A Security Requirements Modelling Language to Secure Cloud Computing Environments

Abstract. This paper presents a cloud-enhanced modelling language for capturing and describing cloud computing environments, enabling developers to model and reason about security issues in cloud systems from a security requirements engineering perspective. Our work builds upon concepts from the Secure Tropos methodology, where in this paper we introduce novel cloud computing concepts, relationships and properties in order to carry out analysis and produce cloud security requirements. We illustrate our concepts through a case study of a cloud-based career office system from the University of the Aegean. Finally we discuss how our cloud modelling language enriches cloud models with security concepts, guiding developers of cloud systems in understanding cloud vulnerabilities and mitigation strategies through semi-automated security analysis.

Key words: Cloud modelling language, meta-model, Cloud Security Requirements, Security Requirements Engineering

1 Introduction

The premise of the cloud computing paradigm is that computing resources are offered by third party providers as a form of commodity accessed through network connections [1, 2]. In comparison to traditional IT solutions, this lowers capital costs and abstracts away implementation and infrastructure details by allowing cloud users to select from pre-configured computing services. However one of the prerequisites for cloud computing; outsourcing data and processes to third parties, raises several security [3, 4] and legal questions [5]. To the cloud user, cloud computing is a black box where the user has little to no control over how or where their data is processed. Multi-tenancy in cloud computing refers to multiple cloud users running independent logical processes but sharing the same physical components, such as CPU, RAM and storage. Virtualisation is the enabling technology for virtual machines, which emulates a physical server and is managed through software known as hypervisors. Therefore a single physical server can host a hypervisor managing one or more virtual machines. Each virtual machine is then allocated to a cloud user and runs cloud services. However from a cloud computing context, the mutual distrust in multi-tenancy environments brings up questions about the security of user data when sharing physical infrastructure [6, 7]. For example consider the scenario where two companies are using virtual machines hosted on the same physical server. A VM escape vulnerability [8] can be exploited, enabling one company to access the sensitive data of another company. This attack has been practically demonstrated in [9], in order to extract information from a target co-residing virtual machine.

In this paper we present a cloud modelling language to capture cloud computing concepts from a security requirements engineering perspective, with enhanced properties and attributes to describe the cloud environment as a system as-is and a system to-be. This work builds upon the modelling language proposed in previous work [**?**], and is part of an on-going research effort to create a framework for holistically modelling secure cloud computing systems, grounded in security requirements engineering and cloud computing security concepts. Our contributions in this paper are:

- *C1*: A cloud meta-model aligning concepts from security requirements engineering and cloud computing.
- *C2*: Definitions for cloud concepts, relationships and properties to holistically model secure cloud computing environments.
- *C3*: Instance syntax notation to facilitate cloud security analysis by enriching cloud models with security vulnerabilities and mitigation strategies.

The rest of the paper is structured as follows. The cloud meta-model, cloud modelling language and security-enhanced cloud computing concepts are defined in Section 2. A motivating case study based on a cloud system for the University of the Aegean Career Office is presented in Section 3. In Section 4 we discuss the respective related work. Finally we conclude the paper in Section 5, noting the on-going work and contributions.

2 The Secure Cloud Modelling Language

In this section we present our secure cloud modelling language, which enables the expression of concepts and relationships of cloud computing systems from a security requirements engineering perspective. We first present the meta-model of the cloud modelling language, which builds upon concepts from the Secure Tropos methodology [12]. Therefore when referring to concepts such as goal, resource and actor, the definitions are found in [12] unless specified otherwise. We then present our novel cloud concepts, relationships and properties in subsection 2.1. Our extensions to the existing concepts in Secure Tropos are expanded in subsection 2.2, and the extensions to existing relationships are described in subsection 2.3. Finally we present the concrete and instance syntax of the cloud concepts in subsection 2.4.

We create a meta-model following UML elements in order to define the concepts and relationships required to model cloud computing systems from a security requirements perspective. Our cloud meta-model is part of an on-going thesis and has been refined through several iterations of published work, the latest appearing in [**?**]. Our cloud meta-model is shown in Figure 1, illustrating the concepts as boxes, relationships as shaded boxes and properties as attributes inside boxes. Specifically we have highlighted the novel cloud computing concepts and relationships proposed in this paper as boxes with thick outlines. Concepts extended from the Secure Tropos methodology is shown as boxes with thin outlines. Boxes with dashed outlines denote existing concepts which has

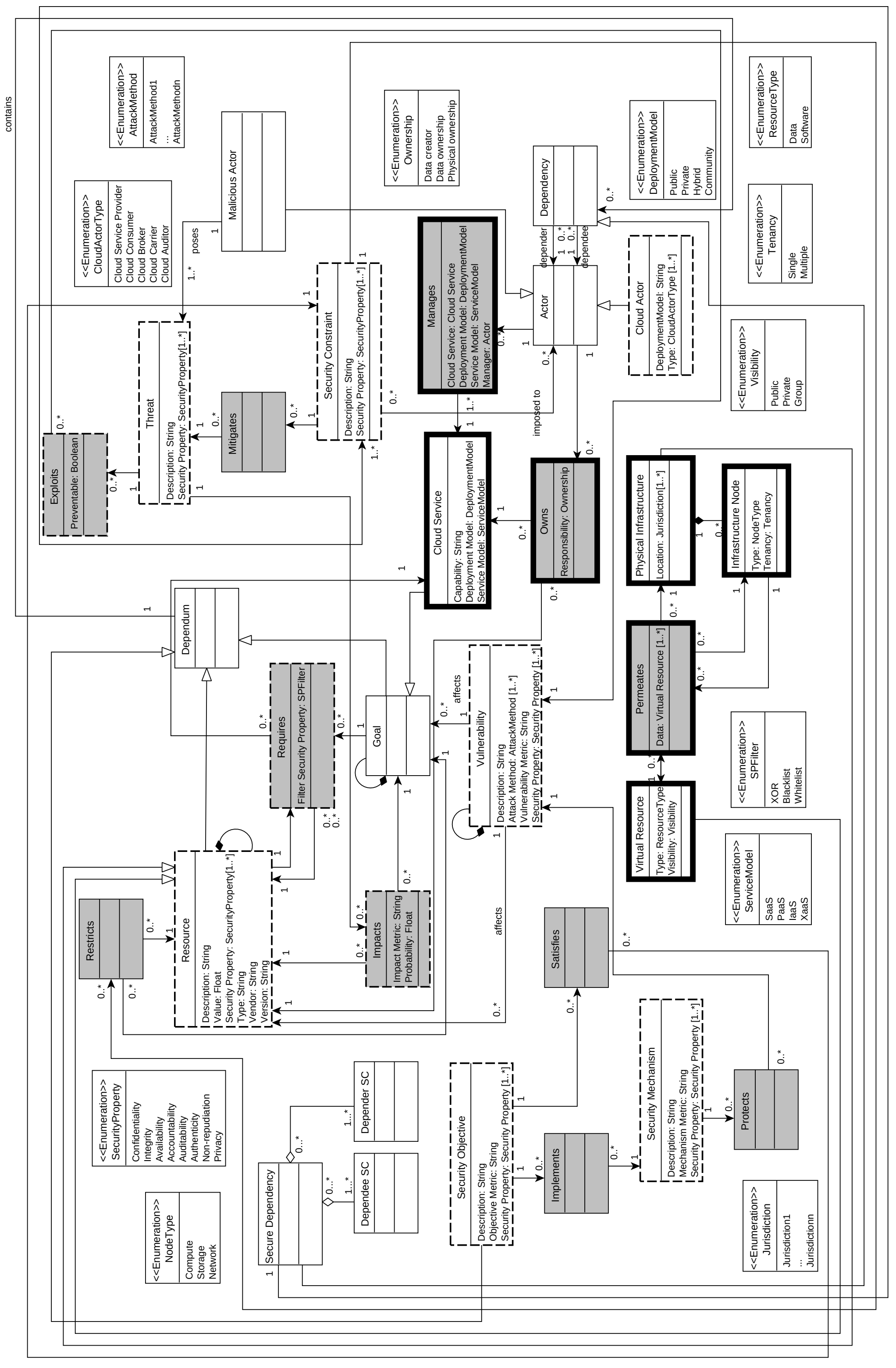

Fig. 1. The cloud meta-model with security requirements engineering and cloud computing concepts

been extended in our research, in order to address the limitations in existing work. We now present our novel cloud computing concepts.

2.1 Proposed Cloud Computing Concepts

In this subsection we introduce our concepts for modelling cloud computing systems, in the context of security requirements engineering.

Cloud Service: *A cloud service provides a specific computing capability, is managed and owned by actors and requires virtual and physical resources in order to deliver its capability.* The cloud service concept is a specialisation of the goal concept found in the Secure Tropos methodology [12], which represents a way to achieve a specific need. The *Cloud service* concept has the properties *Capability*, *Deployment Model* and *Service Model*, indicating the specific computing capability, cloud service deployment model and the service model. The capability property is of type string, which provides a description of the strategic value or work the cloud service is capable of delivering. The *Deployment Model* property specifies which type of cloud service deployment model is deployed by the cloud service, where the enumeration *DeploymentModel* includes the values *Public*, *Private*, *Hybrid* and *Community*. The *Service Model* property specifies the service model of the cloud service, where the enumeration *ServiceModel* includes the values *SaaS*, *PaaS*, *IaaS* and *XaaS*.

Virtual Resource: *A virtual resource represents intangible assets in a cloud computing system.* In order to differentiate between tangible and intangible resources, we create a specialisation of the resource concept to represent intangible resources as virtual resources. An example of an intangible resource is patient data, which has an owner representing an entity that produces the original copy as well as responsible parties representing entities handling the data. The *Virtual Resource* concept has the properties *Type* and *Visibility*. The *Type* property denotes the type of resource using the enumeration *ResourceType*, with values; *Data* and *Software*. The Visibility property denotes the level of visibility of a resource, using the enumeration *Visibility* with values; *Public*, *Private* and *Group*.

Physical Infrastructure: *A physical infrastructure represents a tangible system which, given a geographical location, hosts a group of physical assets within its local proximity.* We define this concept as a specialisation of the resource concept, given that cloud computing resources are hosted in physical infrastructure such as a data-centre. This is essential as properties belonging to the physical infrastructure contain fields such as geographical location, ownership and responsible parties; which is required for performing security analysis. The *Physical Infrastructure* concept has the properties *Jurisdiction*. The *Location* property denotes one or more jurisdictional constraints on the physical infrastructure using values defined in the enumeration *JurisdictionType*, which consists of the values; *GDPD*.

Infrastructure Node: *An infrastructure node represents a single instance of a computing component such as a server, data storage or network connection.* In this case a tangible resource is defined as a specialisation of the resource concept using the notion of an infrastructure node. Each infrastructure node is part of a physical infrastructure, to conceptually represent that a infrastructure node is physically hosted within a structure or area. This allows us to capture the properties of individual nodes through fields such as multi-tenancy, physical location and responsible parties. This is essential for capturing cloud computing concepts at the physical components level, in order to facilitate cloud security analysis from a security requirements engineering perspective. The *Infrastructure Node* concept has the properties *Type* and *Tenancy*. The *Type* property denotes the type of infrastructure node, which is defined through the enumeration *NodeType* using the values *Compute*, *Network* and *Storage*. The *Tenancy* property denotes the tenancy of the infrastructure node, which is enumerated through *Tenancy* with the values *Single* and *Multiple*. The tenancy indicates whether a infrastructure node only involves a single unique user or if multiple users are involved. The NodeID provides a unique identifier for each instance of an infrastructure node in a cloud model. The enumeration *JurisdictionType* consists of the following items: US, UK, EU and Asia. This represents which jurisdiction the asset falls under, and therefore has to adhere to. The enumeration *Tenancy* indicates whether a process is limited to a single tenant or if one or more users are involved.

Permeates: *This indicates the relationship which interrelates data-in-transit and data-at-rest from the virtual resource concepts to the infrastructure node and physical infrastructure, and from the infrastructure node to another instance of the infrastructure node or a physical infrastructure.* A virtual resource is said to be traceable to a infrastructure node if the physical component hosts the virtual resource. For example if user data is stored on a physical hard drive, the data is traceable to the hard drive. A virtual resource is also traceable to a physical infrastructure given the traceable link to a infrastructure node that is part of the physical infrastructure. An infrastructure node can also be traceable to another infrastructure node or physical infrastructure given an exchange of information between the instances, for example a computation node requesting and processing data on a storage node.

Owns: *Owns indicates an actors level of responsibility as a relationship where the initiating actor possess ownership over a physical asset, is the creator of a virtual asset or has data ownership over a virtual asset.* This relationship is used to depict the level of responsibility an actor posses in relation to a cloud service or resource. It is important to define the difference between the data creator, data ownership and physical ownership. The data creator refers to the case where an actor produces data, and thus is the creator of the data in the legal sense. Data ownership refers to the case where data is physically stored on assets owned by third party providers, therefore the third party providers are responsible for the handling of the data. Physical ownership refers to the case where an actor is the owner of a physical asset, such as a server or data-centre. The *Owns* relationship has the property *Responsibility*, which indicates the type of ownership an actor possess in relation to a cloud service or a resource and its specialisations. The enumeration *Ownership* contains the following values; data creator, data ownership and physical ownership.

Manages: *Manages indicates an actors level of responsibility as a relationship, in the configuration and delivery of a cloud service.* An actor can have zero or more *Manages* relationships, indicating that some actors are not involved in the management of cloud services. A cloud service can be the target of one or more *Manages* relationships from a range of actors, indicating that a cloud service is managed by one or more actors.

2.2 Extended Concepts

In this subsection we outline the extensions we make to the existing concepts of the Secure Tropos methodology, linking together concepts from the cloud computing and security requirements engineering domains. These extensions are carried out by adding properties to the existing concepts, which allows the language to express richer levels of information. We now define the extensions to the concepts in our modelling language.

Cloud Actor: *The cloud actor concept has two properties; DeploymentModel representing deployment model and CloudActorType to determine the types of cloud actors an instance plays.* The *DeploymentModel* property represents the type of deployment model, which is of type String. The *Cloud Actor Type* property represents the role the cloud actor plays in a cloud computing context. The enumerated values are based on the five types of cloud actors defined by NIST in [2]; Cloud Service Provider, Cloud Consumer, Cloud Broker, Cloud Carrier and Cloud Auditor. The type of the cloud actor determines the level of responsibility in a *manages* or *owns* relationship. The type of the cloud actor also constrains the validity of relationships with other concepts, based on pre-defined logic or a set of rules defined by the developer. The *Cloud Actor Type* property consists of one or more values from the enumeration *CloudActorType*. This represents the case where a cloud actor may play more than one role simultaneously.

Resource: *The Resource concept has the properties Description, Value, Security property, Type, Vendor and Version.* The *Description* property of type string provides a description of the resource. The *Value* property of type float represents a numerical value associated with user-defined metrics, which assists developers in carrying out semi-automated analysis. The *Security property* describes one or more security properties of the security constraint, which is selected from the enumeration *SecurityProperty* consisting of; *Confidentiality*, *Integrity*, *Availability*, *Accountability*, *Auditability*, *Authenticity*, *Non-repudiation* and *Privacy*. The *SecurityProperty* enumeration also accepts custom values which is optionally added by developers.

Security Constraint: *The Security Constraint concept has the properties Description and Security Property.* The *Description* property is of type string, which provides a description of the security constraint. The *Security Property* property describes one or more security properties of the security constraint, which is selected from the enumeration *SecurityProperty.*

Threat: *The Description property is of type string, which provides a description of the threat. The Security Property property describes one or more security properties of the security constraint from the enumeration SecurityProperties.*

Vulnerability: *The Description property of type string provides a description of the vulnerability. The Attack Method property describes one or more attack methods targeting an instance of the vulnerability, using the enumeration AttackMethods consisting of default values and optionally values defined by the user. The Vulnerability Metric property of type string allows the user to define a numerical value associated with user-defined metrics, which assists them in carrying out semi-automated analysis. The Security Property property describes one or more security properties of the security constraint, which is selected from the enumeration SecurityProperty.*

Security Mechanism: *The Security Mechanism concept has the properties Description, Mechanism Metric and Security Property.* The *Description* property of type string provides a description of the security mechanism. The *Mechanism Metric* of type string allows the user to define a numerical value associated with user-defined metrics, which assists them in carrying out semi-automated analysis. The *Security Property* property describes one or more security properties of the security constraint, which is selected from the enumeration *SecurityProperty.* The *Mechnism Metric* property of type string allows the user to define a numerical value associated with user-defined metrics, which assists them in carrying out semi-automated analysis.

Security Objective: *The Security Objective concept has the properties Description, Security Property and Objective Metric.* The *Description* property of type string provides a description of the security objective. The *Security Property* property describes one or more security properties of the security constraint, which is selected from the enumeration *SecurityProperty.* The *Objective Metric* property of type string allows the user to define a numerical value associated with user-defined metrics.

2.3 Extended Relationships

In this subsection we outline the extensions we make to existing relationships from the Secure Tropos methodology, similar to the extensions carried out in the previous subsection. These extensions to the relationships allows our modelling language to capture the nature of a relationship at the cloud computing level, which isn't supported in existing approaches. We now define the extensions to the relationships in our modelling language.

Requires: *A goal, cloud service or resource requires a cloud service or resource, in order to satisfy a stakeholder need, fulfil a capability or collaborate with other resources or cloud services.* This relationship indicates the resource or cloud service instances required by a goal, cloud service or resource.

The *Filter Security Property* property denotes the type of filter used to determine the security properties inherited in the *Requires* relationship. The purpose of this property is to allow the *Requires* relationship to associate security properties from the source concept to the target concept.

Impacts: *A goal, cloud service or resource is impacted by a threat, which threaten their security properties.* This relationship indicates the resource or cloud service instances impacted by a specific instance of a threat. A threat may impact one or more goals, cloud services or resources.

The *Impact Metric* property defines a metric denoting the consequences of a goal or resource being compromised by the threat. The purpose of this property is to allow the developer to provide a metric for the impact of a threat on goals or resources when performing analysis on the cloud system. The *Probability* property defines a metric representing the likelihood of the threat having an impact on a goal or resource. The purpose of this property is to allow the developer to associate a metric to analysis probability of a threat impacting a goal or resource.

Exploits: *A threat is able to exploit a vulnerability.* This relationship indicates that the specific instance of a threat exploits one or more vulnerabilities.

The *Preventable* property denotes if the relationship of an instance of a threat exploiting a vulnerability is preventable. This allows the analysis to determine if mitigation strategies exist.

2.4 Syntax

In this subsection we present the instance and concrete syntax of the cloud modelling language. We define the instance syntax as machine-readable encoding of instantiated concepts, relationships and properties from our cloud meta-model. We define the concrete syntax as graphical representations of concepts, properties and relationships in our meta-model, which provides an unique one-to-one mapping of a concept from the metamodel to an graphical representation in a cloud model.

The purpose of the instance syntax is to provide a formal and condensed representation of concept instances, which is machine readable in order to perform analysis on cloud models. Thus the instance syntax allows the unambiguous encoding of concepts in a textual format, which describes the instantiated concepts from a cloud model to facilitate security analysis. The general structure of the instance syntax is as follows: an instance of a concept is defined with a lower case abbreviation with parenthesis surrounding the properties and relationships of the instance, the properties and relationships of the instance is defined with upper case abbreviations inside the parenthesis.

The concrete syntax is visualised using graphical notation, where each concept in the modelling language is mapped to a unique graphical notation. The shapes of the graphical notation was chosen arbitrarily in order to distinguish between Secure Tropos notation and our novel cloud computing concepts.

We now describe the syntax of the modelling language using the following format: concept, instance syntax, description and concrete syntax.

Cloud Service: The instance syntax of a cloud service is *cs(D,CAP,DM,SM)*. The instance syntax for a cloud service describes the following: *cs()* describes an instance of a cloud service with associated concepts encapsulated inside the parenthesis, *D* provides a description of the cloud service, *CAP* describes the capability of the cloud service, *DM* is the deployment model selected from a

Cloud Service

Fig. 2. A Cloud Service.

list of enumerated values and *SM* is the service model selected from a list of enumerated values.

Figure 2 shows the concrete syntax of a cloud service, which is represented as a light green rectangle with a solid green outline. The textual description inside the rectangle denotes the properties of the cloud service instance.

Cloud Actor: The instance syntax of a cloud actor is *ca(D,[T])*.

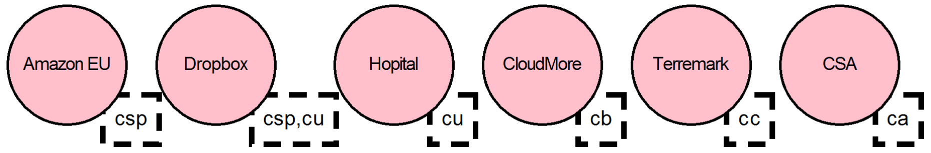

Fig. 3. A Cloud Actor visualised as a light pink circle, with a rectangular box with dashed outlines on the lower right of the circle.

Figure 3 shows the concrete syntax of a cloud actor, which is represented as a light pink circle with a rectangular box with dashed outlines on the lower right of the circle. The text in the centre of the circle denotes the name of the cloud actor instance. The rectangular box with dashed outlines displays the types of roles played by the cloud actor. In Figure 3 a range of cloud actor types are shown, in particular the case of the *Dropbox* cloud actor which is both a cloud service provider and a cloud consumer as indicated by the *csp,cu* textual description. The instance syntax for a cloud service provider describes the following: D is a description of the cloud service provider, *[T]* is a list denoting one or more roles played by a cloud actor. The type of role played by a cloud actor is selected from the enumeration *CloudActorType* with the following list of roles; Cloud Service provider(*csp*), Cloud Consumer(*cu*), Cloud Broker(*cb*), Cloud Carrier(*cc*), and Cloud Auditor(*ca*). For example a cloud service provider with the description *hospital* is encoded as *ca(hospital,[csp])*. In case of cloud actors playing multiple roles, for example *dropbox* who is a cloud service provider and also a cloud consumer is encoded as *ca(dropbox,[csp,cu])*.

Virtual Resource: The instance syntax of a virtual resource is *vr(D,RT,V)*. Figure 4 shows the concrete syntax of a virtual resource, which is represented as a yellow rectangle with a dashed yellow outline. The textual description inside the rectangle denotes the properties of the virtual resource instance. The instance syntax for a virtual resource describes the following: D is the description of the virtual resource instance, RT is the resource type selected from a list of enumerated values, V is the visibility type selected from a list of enumerated

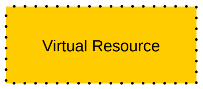

Fig. 4. A virtual resource visualised as a yellow rectangle with a dashed outline.

values. For example the virtual resource *Patient data* with the resource type *data* and visibility type *private* is encoded as *vr(data,private)*.

Physical Infrastructure: The instance syntax of a physical infrastructure is pi(D,L).

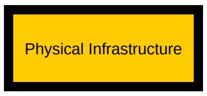

Fig. 5. A physical infrastructure visualised as a yellow rectangle with thick outlines.

Figure 5 shows the concrete syntax of a physical infrastructure, which is represented as a yellow rectangle with a solid black outline. The textual description inside the rectangle denotes the properties of the physical infrastructure instance. The instance syntax for a physical infrastructure describes the following: D is the description of the physical infrastructure, L is an enumerated list of jurisdictions which the physical infrastructure falls under. For example the physical infrastructure *hospital information system* with the jurisdiction *General Data Protection Regulation* (GDPR) is encoded as *pi(hospital_is,gdpr)*.

Infrastructure Node: The instance syntax of a cloud service is in(D,NT,TE).

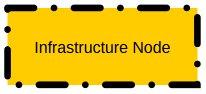

Fig. 6. An infrastructure node visualised as a yellow rectangle with dashed outlines.

Figure 6 shows the concrete syntax of an infrastructure node, which is represented as an aqua cylinder with a black outline. The textual description inside the cylinder denotes the properties of the infrastructure node. The instance syntax for an infrastructure node describes the following: D is the description of the infrastructure node, NT determines the type of the infrastructure node from a list of enumerated values, TE determines the type of tenancy from a list of enumerated values. For example the infrastructure node *ama-*

zon server 1 with the *compute* node type and *single* tenancy is encoded as in(amazon_server1,compute,single).

3 Career Office System Case Study

In this paper we present a case study based on the University of the Aegean Career Office, which is an existing cloud-based system. This case study was presented in a previous collaborative publication by the author in [**?**]. The main objective of the University of the Aegean Career Office system is boundary management, i.e. helping students to manage the choices and transitions they need to make upon completing their studies, in order to proceed effectively to the next step of their careers [11]. The Career Office creates the survey and outsources its hosting and the gathering of responses to a cloud service provider. Once the results are collected, they are send back to the Career Office in order to be analysed and be made available to the university graduates for discussion. There are several existing security needs, specifically that only authorised university members are able to access the survey system and that the data should have guaranteed availability. Thus we are interested in modelling and understanding the potential security issues in this cloud-based system, when the process is outsourced to a third party cloud service provider.

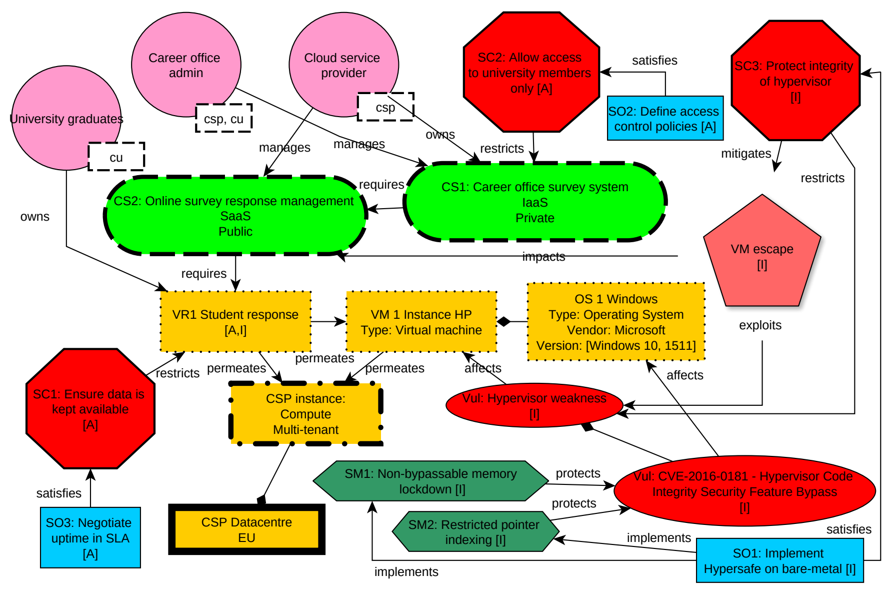
Career office admin
Cloud service provider
SC2: Allow access to university members only [A]
satisfies
SC3: Protect integrity of hypervisor [I]
University graduates
csp, cu
csp
SO2: Define access control policies [A]
cu
manages
manages
owns
restricts
mitigates
restricts
owns
requires
CS2: Online survey response management SaaS Public
CS1: Career office survey system IaaS Private
VM escape [I]
impacts
requires
VR1 Student response [A,I]
VM 1 Instance HP Type: Virtual machine
OS 1 Windows Type: Operating System Vendor: Microsoft Version: [Windows 10, 1511]
exploits
permeates
restricts
permeates
permeates
affects
affects
SC1: Ensure data is kept available [A]
CSP instance: Compute Multi-tenant
Vul: Hypervisor weakness [I]
SM1: Non-bypassable memory lockdown [I]
protects
Vul: CVE-2016-0181 - Hypervisor Code Integrity Security Feature Bypass [I]
satisfies
protects
CSP Datacentre EU
SM2: Restricted pointer indexing [I]
implements
satisfies
SO3: Negotiate uptime in SLA [A]
SO1: Implement Hypersafe on bare-metal [I]
implements

Fig. 7. A security enhanced cloud model of the University of the Aegean Career Office System

Due to space constraints we have modelled the cloud concepts concerning one specific organisational goal of the system, the conduction of a survey of the university's graduates. The cloud model generated using our cloud modelling language proposed in this paper is shown in Figure 7. First we describe actors and their cloud roles, such as *"Career office admin"* who play the role of a cloud service provider and cloud user. The relationship between the actors and cloud concepts include the managers and owns relationships, for example indicating the level of responsibility the cloud service provider has when they own a cloud service. We model the cloud services where one cloud service requires the other, indicating the dependency and therefore propagation of responsibilities to the associated actors. Specifically we are able to see that the cloud service *"CS2: Online survey response management"* requires the virtual resource *"VR1 student response"*, which is owned by the actor *"University graduates"*. The security constraint *"SC1 Ensure data is kept available"* restricts *"VR1"* and due to the requires relationship from CS2, we can infer that the actor responsible for managing *"CS2"* is also responsible for ensuring the security constraint *"SC1"* is addressed. We then describe the resources required by the cloud services, identifying the properties of a virtual machine instance in order to examine vulnerabilities which affects the system given a specification of resources used. Specifically the virtual resource "VM 1 Instance HP" of type *Virtual machine* has the virtual resource *"OS 1 Windows"* associated, which provides additional levels of detail. Thus given the the type, vendor and version properties in the virtual resource *"OS 1 Windows"*, we are able to describe and model the vulnerability affecting this particular configuration. At the high level we model the vulnerability *"Vul: Hypervisor weakness"* which affects the virtual resource "VM 1 Instance HP" due to the type *Virtual machine*. But as the model provides more information through the virtual resource *"OS 1 Windows"*, we are able to model at a lower level of granularity and describe the technical details of the vulnerability, in this case the vulnerability "Hypervisor Code Integrity Security Feature Bypass" with the Common Vulnerabilities and Exposures(CVE) ID *"CVE-2016-0181"* which affects *version 1511* of the type *Operating System Windows 10* by the vendor *Microsoft*. The mitigation concepts are then generated in the model, including the security mechanisms and security objective addressing the specific security property of the vulnerability and the associated security constraint. Specifically we are able to model two alternative security mechanisms *"SM1: Npn-bypassable memory lockdown"* and *"SM2: Restricted pointer indexing"*, which protects the vulnerability *"CVE-2016-0181"*. These vulnerabilities are implemented through the security objective *"SO1: Implement Hypersafe on bare-metal"*, which represents the security policy that should be enforced as part of the mitigation strategy. The *"[I]"* property in the concepts discussed previously indicates that the security property *Integrity* is associated with these concepts. Thus given a threat *"VM escape [I]"*, this threat exploits the *integrity* property of the vulnerability *"Hypervisor weakness"*, which itself affects the *integrity* property.

The contribution here is that by associating security properties to our concepts, we are able to refine the specific security need of each concept in the context of the model. The benefit is two-fold: firstly this provides a guideline for developers to understand the security impact of various concepts, given a set of security constraints embodying the security needs of the system. Secondly this facilitates security analysis, in collaboration with the our proposed properties so that we are able to semi-automate the generation of vulnerabilities and mitigation strategies dependent on the cloud model. Therefore the cloud model of the career office system in Figure 7 illustrates the relationships between the actors, cloud services and required resources, where we enrich the model with security concepts specific to the cloud deployment details. Specifically we are able to model in detail the data required by the cloud service, how the data permeates through the physical components of the cloud system, the jurisdiction due to the physical location, and the specific information of the enabling cloud components including multi-tenancy and virtual machines. Based on the information elicited through these concepts, we are then able to model the cloud security concepts such as vulnerabilities in the cloud system, threats, security constraints and mitigation strategies through security mechanisms and security objectives. Thus our work provides a cloud modelling language enabling developers of cloud computing systems to express and model cloud security needs, understanding the relationship between concepts from a security requirements engineering perspective.

4 Related Work

Research in cloud security primary focuses on mitigating mechanisms and software solutions at the implementation level, which targets software systems that are already implemented and operational [17]. This approach is counter to our argument that security is a factor that is most effective when integrated as early as possible in the software development life-cycle [10]. In existing requirements engineering approaches we are able to capture high level concepts such as stakeholders, goals and resources [13], and security concepts such as vulnerabilities, threats and security constraints [12]. However the approaches in [13] and [12] lack the expressive power to support developers in modelling the relationship between their organisational, business and security needs in a cloud computing environment. Specifically existing approaches lack an language expressive enough to model cloud-specific concepts such as multi-tenancy, virtualisation and cloud services in the context of cloud security. Beckers et al. proposes a pattern-based method to elicit cloud security requirements aimed at guiding cloud customers during the process of modelling cloud systems [14]. However their approach is focused on establishing an Information Security Management System (ISMS) according to the ISO 27001 standard, without considering the propagation of users cloud security needs. Li et al. provides a holistic security requirements-eliciting approach towards socio-technical systems [16]. However their work lacks expressive power for capturing cloud-specific properties, which is essential for inferring cloud security issues, impact on resources and mitigation strategies. Review efforts indicates that while most work covers multiple security sub-areas, they only target cloud computing issues in isolation, for example considering security properties in software systems or human factors on a social level but failing to provide direct correlation between different conceptual layers [18, 19].

Our proposed approach ensures that the system-under-design incorporates security from the early requirements stage, by providing an expressive modelling language able to capture cloud computing concepts and characteristics. We achieve this by building upon existing work in security requirements engineering that lacks the capability to capture or reason about cloud-specific security issues from a holistic point of view [15, 12]. Thus our modelling language captures essential cloud characteristics and security implications in order to progress towards addressing the security problem in cloud computing [5]. Specifically we model cloud characteristics such as multi-tenancy, describe cloud service configurations, identify cloud-specific threats and vulnerabilities, in addition to modelling the impact of attacks on physical and virtual resource within the context of a cloud computing system.

5 Conclusion

The proposed cloud modelling language in this paper enables developers to realise organisational needs in a cloud computing context, capturing the needs of the system as-is and expressing the system to-be. Addressing the contribution points *C1* and *C2*, we have defined a cloud modelling language to capture cloud security concepts. We argue that by enhancing the proposed cloud concepts with detailed relationships and properties, the language is able to represent cloud computing systems through both abstract and fine-grained perspectives. For the contribution in *C3* we have provided the concrete and instance syntax of our cloud concepts. The concrete syntax allows us to visually differentiate between instances of unique components in cloud systems, assisting the developers understanding of a cloud system. The instance syntax facilitates support for semi-automated cloud security analysis, providing a machine-readable format of cloud models in order to identify threats, vulnerabilities and mitigation strategies. Future work will focus on enhancing the support for semi-automated analysis techniques, using information from vulnerability databases and expert security knowledge to generate a security mitigation strategy given a cloud model.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50–58.
2. Mell, P., & Grance, T. (2011). The NIST definition of cloud computing.
3. Subashini, S., & Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing. Journal of network and computer applications, 34(1), 1–11.

4. Sengupta, S., Kaulgud, V., & Sharma, V. S. (2011, July). Cloud computing security–trends and research directions. In Services (SERVICES), 2011 IEEE World Congress on (pp. 524-531). IEEE.
5. Almorsy, M., Grundy, J., & Mller, I. (2010, November). An analysis of the cloud computing security problem. In Proceedings of APSEC 2010 Cloud Workshop, Sydney, Australia, 30th Nov.
6. Lombardi, Flavio, and Roberto Di Pietro. "Secure virtualization for cloud computing." Journal of Network and Computer Applications 34.4 (2011): 1113–1122.
7. Li, Y., Cuppens-Boulahia, N., Crom, J. M., Cuppens, F., & Frey, V. (2016, May). Expression and Enforcement of Security Policy for Virtual Resource Allocation in IaaS Cloud. In IFIP International Information Security and Privacy Conference (pp. 105-118). Springer International Publishing.
8. Luo, Shengmei, et al. "Virtualization security for cloud computing service." Cloud and Service Computing (CSC), 2011 International Conference on. IEEE, 2011.
9. Ristenpart, T., Tromer, E., Shacham, H., & Savage, S. (2009, November). Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In Proceedings of the 16th ACM conference on Computer and communications security (pp. 199–212). ACM.
10. Kissel, R., Stine, K., Scholl, M., Rossman, H., Fahlsing, J., & Gulick, J. (2008). Security considerations in the system development lifecycle [Computer software manual]. (NIST Special Publication 800–64 Revision 2)
11. C. Kalloniatis, E. Kavakli and S. Gritzalis, "Dealing with privacy issues during the system design process", Proceedings of the Fifth IEEE International Symposium on Signal Processing and Information Technology, pp. 546–551, 2005.
12. Mouratidis, H., & Giorgini, P. (2007). Secure tropos: a security-oriented extension of the tropos methodology. International Journal of Software Engineering and Knowledge Engineering, 17(02), 285–309.
13. Eric, S. Yu. "Social Modeling and i*." Conceptual Modeling: Foundations and Applications. Springer Berlin Heidelberg, 2009. 99-121.
14. Beckers, K., Côté, I., Fabender, S., Heisel, M., & Hofbauer, S. (2013). A pattern-based method for establishing a cloud-specific information security management system. Requirements Engineering, 18(4), 343–395.
15. Fabian, B., Gürses, S., Heisel, M., Santen, T., & Schmidt, H. (2010). A comparison of security requirements engineering methods. Requirements engineering, 15(1), 7–40.
16. Li, T., Horkoff, J., Beckers, K., Paja, E., & Mylopoulos, J. (2015). A holistic approach to security attack modeling and analysis. In Proceedings of the Eighth International i* Workshop (2015, to be published).
17. Modi, C., Patel, D., Borisaniya, B., Patel, A., & Rajarajan, M. (2013). A survey on security issues and solutions at different layers of Cloud computing. The Journal of Supercomputing, 63(2), 561–592.
18. Sengupta, S., Kaulgud, V., & Sharma, V. S. (2011, July). Cloud computing security–trends and research directions. In Services (SERVICES), 2011 IEEE World Congress on (pp. 524–531). IEEE.
19. Iankoulova, I., & Daneva, M. (2012, May). Cloud computing security requirements: A systematic review. In Research Challenges in Information Science (RCIS), 2012 Sixth International Conference on (pp. 1–7). IEEE.