The Journal of Systems and Software 163 (2020) 110537

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A novel Security-by-Design methodology: Modeling and assessing security by SLAs with a quantitative approach

Valentina Casola[a,*], Alessandra De Benedictis[a], Massimiliano Rak[b], Umberto Villano[c]

[a] Department of Electrical Engineering and Information Technologies, University of Naples Federico II, Naples, Italy
[b] Department of Computer Engineering, University of Campania Luigi Vanvitelli, Aversa, Italy
[c] Department of Engineering, University of Sannio, Benevento, Italy

ARTICLE INFO

Article history:
Received 19 June 2019
Revised 12 November 2019
Accepted 11 January 2020
Available online 25 January 2020

Keywords:
Security-by-Design methodologies
Security Service Level Agreement
Security metrics
Security models
Security Assessment
Secure Cloud Application Development

ABSTRACT

Recent software development methodologies, as DevOps or Agile, are very popular and widely used, especially for the development of cloud services and applications. They dramatically reduce the time-to-market of developed software but, at the same time, they can be hardly integrated with security design and risk management methodologies. These cannot be easily automated and require big economic investments, due to the necessity of security experts in the development team and to the lack of automatic tools to evaluate risk and to assess security in the design and operation phases. This paper presents a novel Security-by-Design methodology based on Security Service Level Agreements (SLAs), which can be integrated within modern development processes and that is able to support the risk management lifecycle in an almost-completely automated way. In particular, it relies upon a guided risk analysis process and a completely automated security assessment phase, which enable to assess the security properties granted by a cloud application and to report them in a Security SLA. We validated the proposed methodology with respect to a real case study, which showed its effectiveness in improving the awareness of designer and developer teams on security aspects and in reducing the secure design process time.

1. Introduction

In recent years, the business relying on the development of cloud-based services and application has given priority to fast and agile development processes, reducing significantly the time-to-market. Unfortunately, the correct design and assessment of security properties, aimed to properly configure and enforce security mechanisms, are often left out from these processes. In fact, recent studies on the economic aspects of security have highlighted the so-called "information asymmetry" of security, where "from one side, the market players are likely not investing in the right defences with the right amount of money and, on the other side, ill-informed consumers are more likely to buy snake-oil solutions if they are unaware of the full extent of the threats" Brangetto (2015). The situation is going to change, particularly in Europe, due the introduction of the General Data Protection Regulation (GDPR The EU General Data Protection Regulation (2018)) and to the application of the *Directive on security of network and information systems* (NIS Directive) Cybersecurity and Digital Privacy (Unit H.1 Team) (2018), which impose the adoption, in all information systems, of security countermeasures able to cope with security threats and incidents. Nevertheless, the development of secure cloud applications is still an open issue, due to multiple factors.

First, the development of cloud applications is mainly based on the orchestration of cloud services and on the deployment of Commercial-Off-The-Shelf (COTS) software components over cloud resources Benfenatki et al. (2014); Casola et al. (2019). Such an approach implies a lot of additional security issues, since COTS components are likely to introduce new un-predictable threats and, therefore, are commonly considered not suitable for security-critical applications Koch (2012).

Second, addressing security in clouds is an open issue by itself. Developing cloud applications relies on outsourcing every kind of resource to external Cloud Service Providers (CSPs), losing control over their usage and configuration, and making it very hard to verify and enforce security mechanisms across the complete architectural stack Thillaiarasu and ChenthurPandian (2016); Bousquet et al. (2015). To cope with this issue, a recent report by the

* Corresponding author.
E-mail address: casolav@unina.it (V. Casola).

https://doi.org/10.1016/j.jss.2020.110537
0164-1212/

European Union Agency for Network and Information Security (ENISA) European Network and Information Security Agency (ENISA) (2012) suggested the adoption of security Service Level Agreements (*Security SLA*), i.e., of contracts among service providers and service customers stating the level of granted security, as the main way to enforce and monitor security in clouds.

It is worth noting that, despite existing directives and guidelines, actual security-oriented SLAs are not yet adopted by providers, especially in public clouds Casola et al. (2015). To bridge this gap, some initiatives have been recently promoted aimed at assessing the level of security offered by cloud providers, in order to help customers make the best procurement decisions. Among these, the Cloud Security Alliance (CSA) has promoted the Security Trust Assurance and Risk (STAR) Program Cloud Security Alliance (2011b), aimed at certifying the compliance of providers with the security controls included in the Cloud Control Matrix (CCM) CSA (2015) through a self-assessment process based on the Consensus Assessments Initiative Questionnaire - CAIQ Cloud Security Alliance (2011a). The answers given by providers are very useful to evaluate their overall security posture, and can be profitably used to characterize their security offer, but it must be noted that they are quite general and not tailored to a specific service offering.

In fact, as outlined in Casola et al. (2018a); Rios et al. (2017), providers mainly offer SLAs that guarantee the same service terms (not only from the security point of view, but also in terms of availability, response time, etc.) *for all their services, to all their customers*, regardless of their specific needs. In other words, CSPs offer *per-Provider SLAs*, instead of the more desirable *per-Service SLAs*, which should refer to specific services and include guarantees possibly tailored to the needs of specific customers, to take into account their different requirements. In this context, the capability of negotiating Security SLAs with providers would be highly desirable, even if still not considered by current vendors.

To bridge this gap, several European projects such as SPECS SPECS project web site (2016), MUSA MUSA Consortium (2016), SLA-Ready SLA-Ready Consortium (2019), and SLALOM SLALOM Consortium (2019) have recently investigated the adoption of SLAs and Security SLAs in the cloud by proposing models and frameworks for SLA definition and life-cycle management.

A further issue that considerably affects secure cloud application development is linked to the software development approaches most commonly used today, namely DevOps Dyck et al. (2015) and Agile. They dramatically reduce the time-to-market of developed software but, at the same time, they can be hardly integrated with security design and assessment processes. In fact, both Agile methodologies and DevOps processes imply a high number of continuous development iterations, involving development teams that are typically made of five to seven people taking care of functional analysis, development and testing, and that very often do not include security-skilled personnel. Furthermore, while DevOps requires high automation capabilities to be effective, security assessment and testing can be hardly automated Mohan and Othmane (2016); Ur Rahman and Williams (2016). On the contrary, traditional security engineering practices, based on security requirements-oriented software development, typically require the presence of security experts in the development teams, and imply costly and time-consuming activities Anderson (2008); Massacci et al. (2005); Mouratidis and Giorgini (2007); Khan and Ikram (2016). The final result is a dramatically high number of hours of work spent monitoring the security provided by a service and/or remedying to security incidents.

In order to provide more flexible and effective ways of dealing with security in a software development process, the concept of Security-by-Design has recently appeared Cavoukian and Chanliau (2013). It refers to a holistic, anticipatory approach to security (in contrast to the traditional security-by-obscurity principle) based on the adoption of the secure-by-default paradigm in the configuration of both software components and access policies, and of software security assurance processes aimed at identifying, as early as possible, existing threats and vulnerabilities and at including the design of security components in the software architecture from the beginning. Despite their potential, Security-by-Design approaches are not yet widely adopted due to the lack of mature solutions and tools. Available approaches are expensive, many are still at research level and are mainly related to the implementation of specific security controls, which is only one of the steps of the complex risk management life-cycle that should drive the whole development process Joint Task Force Transformation Initiative I (2013), which includes also risk analysis and security assessment activities.

In this regard, it is worth noting that, however, risk management techniques are only partially able to cope with all features affecting the risk management, due to the complexity of properly capturing all threats and vulnerabilities of complex systems, and also because unpredictable threats may be introduced by commercial components or even by insiders Ganin et al. (2017); Collier et al. (2014). Some available techniques provide flexible frameworks in order to take the security evaluators “always in the loop” and cope with non-deterministic and unpredictable aspects that may affect the risks but, at the best of our knowledge, few automated and quantitative security assessment techniques are available.

The adoption of Security SLAs and, in particular, of security metrics, seems very promising in the evaluation of many aspects of cybersecurity, from a technical, economical and business perspectives. In particular, we have investigated how it can support, in a quantitative way, the security design process and almost all phases of risk management life-cycle, namely the *identification* of the critical assets to be protected, the *selection* and *implementation* of the security controls that are needed to protect the identified assets, the *assessment* of such security controls in order to verify that they are properly implemented, and their *monitoring* Joint Task Force Transformation Initiative I (2013).

1.1. Motivation and paper contribution

The motivations of the work presented in this paper are linked to a number of open issues we spotted in the state of the art of secure software development, namely:

- security oriented software development life cycles are costly, time expensive and often need the involvement of security experts;
- the adoption of the widespread Agile and DevOps approaches tend to complicate security assessment processes, and needs *ad-hoc* solutions to address security requirements;
- there is a lack of techniques and tools able to automate security analysis and assessment, which, at the state of the art, rely mainly on human expertise;
- security in cloud environments, which delegate responsibilities to third parties, is an open issue; moreover, the majority of existing security models can hardly be used to take into account the interaction of multiple services.

Our contribution intends to address such issues by leveraging our previous work on cloud applications and security modeling to simplify the security analysis and assessment of cloud applications and reduce the human intervention by security experts. In particular, in this paper we propose:

- a novel **Security SLA-based Security-by-Design methodology** for the development of secure cloud applications that:
  1. relies upon Security SLAs to specify the application security capabilities and to quantify the provided level of security;
  2. is based on novel automation techniques for risk analysis and security assessment;
  3. is able to support all the phases of the software development, taking continuously into account the security requirements;
  4. can be integrated within Agile and DevOps development processes without altering the related application development flow.
- a technique to *partially automate the risk analysis* and
- a technique to *partially automate the security assessment* of cloud applications.

The proposed methodology was implemented by a prototype tool (the *SLA Generator* tool) that was used for validation in the context of the European project MUSA MUSA Consortium (2016), recently closed and focused on the design of secure multi-cloud applications. In particular, we integrated the tool in the DevOps cycle proposed by the project, and conducted a validation on two real-world case study applications. Validation, involving two different evaluators' teams with limited security skills, took into account several quality metrics including *efficiency*, *usability* and *time consumed*, and showed very promising results.

1.2. Paper organization

The reminder of this paper is structured as follows. In Section 2, the related work is presented and discussed with a particular focus on the integration of security aspects in state-of-the-practice software development methodologies and on quantitative approaches to security and risk assessment of cloud applications. In Section 3 we illustrate the security data model that is at the basis of the proposed methodology, by also providing details on the Security SLA model and on the formalism adopt to represent cloud applications. In Section 4, the proposed Security-by-Design methodology is introduced, and its integration within common Agile software development methodologies is discussed. In sections from 5 to 8, we show all the steps at the basis of the methodology and, in particular, we present the risk analysis and security assessment processes, applicable on both single components of the application and on the application as a whole. In Section 9, we validate the proposed approach by illustrating its integration in a DevOps workflow and by evaluating its main benefits in terms of the reduction of the time needed for the design of a secure application. Moreover, we provide a final discussion of the presented approach and of its limitations. Finally, in Section 10 some conclusions and future work are presented.

2. Related work

In this section, we will deal with existing approaches aimed at integrating security aspects within software development methodologies and processes, and with frameworks and solutions devoted to risk analysis and security assessment tasks within such processes. In particular, in Section 2.1 we reported an overview of current security issues in the cloud, in Sections 2.2 and 2.3 we illustrated security approaches in most common development methodologies, in Section 2.4 we discussed in details the relation between Agile methodologies and security and, finally, in Section 2.5 we reported related work on the security assessment problem as important task within software development methodologies.

2.1. Security in cloud

As often outlined by Gartner (e.g., *Gartner Cloud Security Primer 2019*[1]) and ENISA documents European Network and Information Security Agency (ENISA) (2012); Dekker (2012); Catteddu (2011), security is still considered an inhibiting factor for the diffusion of the cloud computing paradigm: cloud customers typically do not trust external cloud providers, and hesitate to delegate the management of their resources and data.

The cloud model inherits all the security issues of a typical datacenter, but introduces additional issues typical of the cloud approach. Interesting surveys on cloud security issues are available in Singh et al. (2016) and Hong et al. (2019), while reference Chen and Zhu (2017) outlines how a cloud system can be used as a starting point for building Advanced Persistent Threats (APT).

Developing secure cloud application is an open problem, and various proposals exist that try to address the issue by adapting Software Development Life Cycles Aljawarneh et al. (2017), in a way similar to the one proposed in this paper. Other works (e.g., Rak et al. (2014)) propose the use of techniques to address security aspects during the development process. The adoption of Security SLA as a basis for development of secure applications is the core of the SPECS Project Rak et al. (2013); SPECS project web site (2016), which has obtained interesting results in the automatic enforcement and continuous monitoring of security, as outlined in Casola et al. (2017a). In particular, this last approach has been taken into account for devising the methodology proposed in this paper and Security SLA have been used to model and quantify security.

2.2. Security in software development methodologies

The consideration of security aspects in the design of systems is the focus of so-called *security engineering* practices Anderson (2008), aimed at building systems that are acceptably robust against possible disruptions, threats and hazards. These practices typically suggest the adoption of processes that must be applied systematically to a target system and carried out during its entire life cycle Ross et al. (2016) and mainly focus on the post-development testing activities, aimed to validate the effectiveness of already enforced security controls or to identify existing weaknesses and guide future security efforts and investments The Software Assurance Forum for Excellence in Code (SAFECode) (2018); Common Criteria (2017). An approach to security engineering widely followed in the literature is to address security in the software life cycle using the principles of *requirements engineering*, which refers to the process of defining, documenting and maintaining requirements and, accordingly, to the software engineering methodologies concerned with that process. In this context, Giorgini et al. Giorgini et al. (2004) proposed an extension to the Tropos agent-oriented software engineering methodology presented in Bresciani et al. (2004). In particular, they presented a formal framework for modeling and analyzing security and trust requirements, which distinguishes actors that manipulate resources or execute tasks, from the actors that own the resources, and that enables to automatically verify security and trust requirements by using a suitable delegation logic. Related approaches have been presented in Mouratidis et al. (2005); Massacci et al. (2005); Mouratidis and Giorgini (2007); Khan and Ikram (2016). It is worth noting that these software engineering approaches are both expensive and time-consuming, since they typically require costly security expertise and rely upon formal and complex procedures that negatively affect the development

[1] https://www.gartner.com/en/documents/3900688/security-of-the-cloud-primer-for-2019.

process time line. For these reasons, they are regularly applied in security-critical contexts, but are often neglected by small and medium enterprises, which often consider security barely as an 'additional requirement' for their products.

2.3. Security-by-Design methodologies

In order to improve the effectiveness of implemented security practices, security should be taken into account as early as possible in the development process. This concept is the basis of *Security-by-Design* Cavoukian and Chanliau (2013). This is an approach to security engineering conceived in contrast to the traditional security-by-obscurity principles, which mainly relies on keeping the secrecy and confidentiality of the internal implementation of a system to keep it secure. The Security-by-Design approach suggests the adoption of proactive measures against existing security threats and the implementation of the secure-by-default paradigm in the configuration of both software components and access policies. Overall, Security-by-Design requires to embed security in the design of systems via the adoption of both *software security assurance processes* and *trusted hardware*. Software assurance processes include, for example, carrying out a comprehensive threat analysis, include the design of countermeasures against existing threats as part of the system architecture, adopting repeated code reviews and audits, and executing a rigorous security testing. Based on this principle, several Secure Development Life Cycles (SDL) have recently been defined. The most common ones are Microsoft SDL, OWASP OpenSAMM and Cisco SDL. All these life-cycles include threat modeling at the beginning of the development process, and continuous security testing and assessment over all the phases of the software product development. The main limits of such approaches is the cost they imply: they assume the involvement of an "expensive" team of security experts over the whole development life cycle and/or of security-skilled developers. As a matter of fact, a few companies are actually applying the methodology as a common procedure, but only for security-critical application development. The interested reader is referred for additional information on these topic to the surveys Jayaram and Mathur (2005); Geer (2010).

2.4. Agile methodologies and security

In spite of the available approaches previously discussed, it is worth noting that neither traditional security engineering practices nor Security-by-Design approaches are well suited to the modern development processes and methodologies, such as *DevOps* and *Agile*, which are pushing toward the rapid and automated deployment of code in subsequent incremental releases, following faster development and testing processes.

Nevertheless, some proposals exist aimed at integrating security management into the most popular agile methodologies, such as Scrum. Azham et al. proposed in Azham et al. (2011) an extension to the Scrum methodology relying upon *security backlogs*, to be used during development, analysis and implementation in Scrum phases. As a matter of fact, several challenges still remain, due to the fact that they do not include specific phases for security requirements definition and risk assessment and imply an incremental development process where code changes frequently, thus breaking security constraints, as also discussed in Wäyrynen et al. (2004); Ghani and Yasin (2013); Ghani et al. (2016); Pohl and Hof (2015).

Mougouei et al. introduced a different extension to Scrum, namely *S-Scrum* Mougouei et al. (2013), which incorporates security analysis and security modeling activities in so-called 'spikes', carried out after release planning and before sprint planning and execution, and producing the (security-enhanced) model of the current release product items. The authors, however, do not propose any security analysis techniques for security analysis and design, while they suggest the adoption of penetration testing techniques in the test stage. Finally, Pohl et al. proposed *Secure Scrum* Pohl and Hof (2015), a different framework that does not require any change in the traditional Scrum process and that is based on the enhancement of Scrum user stories with *security tags*, which relate them to security-related stories (typically representing misuse cases). Secure Scrum is designed to ensure that an 'appropriate' security level is reached (i.e., the cost to exploit a vulnerability is higher then the expected gain of the exploit) and assumes that the vast majority of requirements is handled by the team itself, but admits the inclusion of external security consultants to cope with specific security issues.

The same security challenges found in agile methodologies can be observed in the popular DevOps practices Dyck et al. (2015), which are being commonly adopted in the development of modern systems (including cloud-based systems) and that emphasize the collaboration within and between different teams involved in software development by providing automation and event monitoring at all development steps. According to such practices, software is deployed iteratively, typically without a deep involvement of a security team: rapidly deployed software changes are more likely to contain vulnerabilities due to the lack of adequate reviews Ur Rahman and Williams (2016). Recent studies outline that the DevOps automation tools, if correctly used, may help to correctly address security assessment procedures Mohan and Othmane (2016); Ur Rahman and Williams (2016), leading to the introduction of more security-oriented processes, named SecDevOps or DevSecOps Mohan and Othmane (2016). However, as highlighted by Mohan et al. in Mohan and Othmane (2016), suitable methodologies able to address automated security assessment processes are still lacking.

Also the previously mentioned Microsoft SDL applies the secure development principles in the context of agile programming, adjusting the life cycle to the SDL-Agile methodology. In practice, the security requirements are reported in the blacklog and addressed in the Scrum sprints, applying the activities that SDL prescribes. As a consequence, the Scrum planning is enriched with security-oriented practices. Microsoft SDL-Agile integrates the security-by-design principles in the agile paradigm, but does not tackle the issue of the role of the security experts, that must be part of the team and be continuously involved in the activities.

The above discussion can be summarized as follows: modern development methodologies urge novel methods and techniques that (i) enable to take security into account in each development iteration, possibly based on a Security-by-Design approach, (ii) do not affect negatively the rapidity and flexibility of the development process, and (iii) are made of short, mostly automated tasks, which can be executed even by personnel provided with basic security skills.

It is worth noting that both Agile and common SDLs, even if are able to introduce security aspect in the flow, are not able to address one of the key issues: the lack and cost of security experts. The proposed methodology aims at coping with this issue by introducing automatic techniques and tools to be used by not expert users.

2.5. Security analysis and assessment in development processes

Security analysis and risk assessment processes have been addressed widely in the literature. As a matter of fact, several approaches have been proposed for the identification of possible threats and the evaluation of the associated level of risk, and some of the proposed approaches also devise automated techniques and tools to carry out these tasks. As widely discussed in

Collier et al. (2014); Gisladottir et al. (2016); Ganin et al. (2017), there are several challenges related to the identification and evaluation of threats, vulnerabilities and their consequences on the systems due to the high variability of applicable threats over time, depending on the system deployment context and on other aspects, to the need to cope with organizational and human factors, which may provide additional insider threats, and also to the lack of a shared terminology for the characterization of cyber risk. In this regard, Mateski et al. Mateski et al. (2012) well highlighted the difficulty of identifying and characterizing threats, and introduced a generic threat matrix with the aim of enabling government entities and intelligence organizations to categorize threat into a common vocabulary (threats are classified based on different criteria including stealth, time, technical personnel, cyber knowledge, and access). To overcome some of the challenges facing cyber risk assessment and also preserve system resiliency, current approaches usually identify subsets of threats and vulnerabilities that may apply to their systems and ask security experts to validate them but, at the best of our knowledge, few works are available that provide a fully automated process. Among these, Ganin et al. Ganin et al. (2017) proposed a decision-analysis-based approach to quantify threats, vulnerabilities, and consequences through a set of criteria, which enables to compute an aggregate risk score based on a multi-criteria decision process, useful to assess the overall utility of cybersecurity management alternatives.

While several tools and techniques exist that focus on the security analysis of a system, to the best of our knowledge the tools and techniques for the automated security assessment of systems are still lacking. Pfleeger and Cunningham Pfleeger and Cunningham (2010) identify some of the challenges associated with measuring security in the cyber domain, including the inability to verify the security requirements and the dynamic features of systems, given the strict connection of virtual, heterogeneous domains.

In this paper, we aim at bridging this gap by proposing a Security-by-Design process that can be integrated within the most common agile and DevOps methodologies thanks to the automation of both the security analysis phase and the security assessment phase. In particular, we adopt a reference security model built upon standards to retrieve the knowledge on threats and vulnerabilities of interest for a system, use well-known processes for risk analysis and assessment, and adopt a novel technique for automated security assessment.

3. The system and security model

This paper relies on some work that has been done in the context of two European research projects (SPECS and MUSA) where we defined a Security SLA model and proposed a Multi-cloud application model. Sections 3.1 and 3.2 briefly summarize these results that are the basis for the definition of the novel security data model that will be presented in Section 3.3.

In fact, the methodology proposed in this paper is based on a reference security data model that puts together the main aspects involved in the security characterization of a system and on a specific description of cloud applications, which enables to capture the most relevant aspects needed for automated security analysis and assessment.

3.1. The Security SLA model

The Security SLA model adopted in this paper has been introduced in the SPECS project and has been widely discussed in Casola et al. (2017a). The model, whose graphical representation is presented in Fig. 1, is based on the WS-Agreement standard, which has been extended to include cloud provider-specific information and security-related guarantees.

A Security SLA includes both *service description terms* and *guarantee terms*. The former section specifies the functional and security features of the cloud service object of the agreement, which may be possibly built upon multiple *services* offered by different *resource providers*. The security policies associated with the service are declared in the form of *capabilities*, which are expressed in terms of standard *security controls*, defined by NIST as *safeguards or countermeasures prescribed for an information system or an organization designed to protect the confidentiality, integrity, and availability of its information and to meet a set of defined security requirements* Joint Task Force Transformation Initiative I (2013).

Several security control frameworks have been proposed, both by standardization bodies and industry-oriented organizations, such as the NIST Security Control Framework Joint Task Force Transformation Initiative I (2013), the ISO/IEC 27002 specification International Organization for Standardization (2013) and the Cloud Security Alliance's Cloud Control Matrix CSA (2015). They list several security controls addressing different security domains and are related to both technical and organization aspects. As an example, the NIST Security Control Framework (currently supported by our process) lists more than 900 controls belonging to 18 different *control families*, including access control (AC), identification and authentication (IA), physical and environmental protection (PE) and awareness and training (AT).

Service description terms include also a list of available *security metrics*. Metrics considered by our Security SLA model belong to the Security Metric Catalogue discussed in Casola et al. (2017b), which was initially released by the SPECS project and that has been later refined and enriched based on several sources including European projects, international standards and initiatives and scientific papers (e.g., A4Cloud and MUSA projects, Center for Internet Security and NIST 800-55r1 Special Publication).

Security metrics included in the catalogue differ significantly from one another. Some metrics, referred to as technical metrics, provide a measure of how well a service or system has been configured to enforce a given set of controls, and can be monitored via classic monitoring tools or by checking the active system configuration. Examples of such metrics are `Vulnerability Scanning Frequency` (frequency of vulnerability scan processes triggered in a system), `HTTPS Redirect Activation` (activation of the feature of the HTTP service that forces clients to use only secure HTTP protocol), and `Identity Assurance` (type of authentication mechanism in place, ranging from the simple user-password mechanism to much more complex two-factor authentication mechanisms). Other metrics, referred to as management metrics, refer to the way an organization sets-up its security policies and procedures and handles security incidents and are less related to a specific service or system. Each metric in the catalogue is associated with one or more security controls and helps verify if such controls have been correctly implemented and at which level.

Security metrics are used to define security guarantees in the *guarantee terms* section of the Security SLA. In particular, security guarantees are expressed by means of security Service Level Objectives (*SLOs*), specified in terms of boolean conditions built upon security metrics (e.g., `Identity Assurance > user_pass` specifies that an authentication mechanism based on username/password is not admissible but some more complex mechanism must implemented by a service, such as for example certificate-based authentication).

Once implemented, the Security SLA of an application should be continuously monitored in order to guarantee that the conditions defined by SLOs are always verified. In case of violation, or whenever needed, a re-negotiation process should be devised to establish a new agreement with updated conditions. The interested reader can find the details on the complete SLA life-cycle management in Casola et al. (2015).

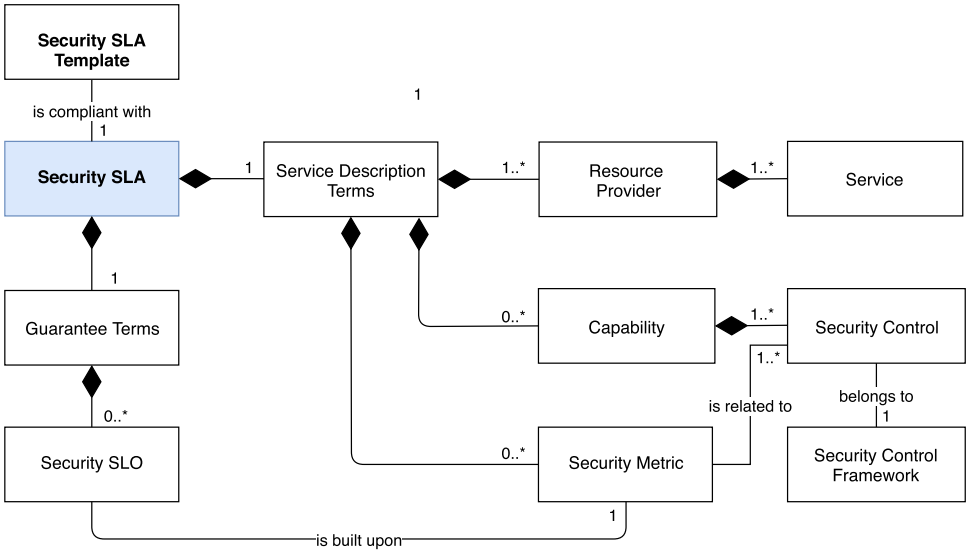

Fig. 1. The Security SLA model.

Finally, as shown in the figure, the model includes the concept of *Security SLA Template*. The WS-Agreement specification defines an SLA Template as a document that complies with the general structure of the agreement but enables to specify customizable aspects, called constraints, that make it possible to express the valid ranges or distinct values that the terms may take. Templates are used to create an initial agreement to negotiate, which differs from an actual SLA only because the parties have not yet reached an agreement on the guarantees that will actually be granted. In the remainder of this paper, we will use the term SLA Template to refer to an agreement that formally complies with the model illustrated above but that includes guarantees that have not yet been verified.

3.2. The cloud application model

As mentioned in the Introduction, the cloud applications considered in this paper are built by suitably orchestrating a set of components. For simplicity's sake, we assume that application components are either represented by cloud-based services invoked in a Software-as-a-Service (SaaS) fashion from a third-party provider, or by custom/COTS pieces of software that need to be deployed and executed on a cloud-based compute resource (i.e., a virtual machine). In both cases, components are considered as services, either offered *as-they-are* by a provider or set-up and executed on top of leased cloud Infrastructure-as-a-Service (IaaS) resources. The formalism adopted to model such an application is the MACM formalism Casola et al. (2018b), a graph-based formalism which enables to describe the architecture of a cloud application in terms of its components and their relationships, and to specify deployment and implementation details (e.g., IP address or port, communication protocol) along with security properties.

In MACM, application components are modeled as graph nodes belonging to the *SaaS* node type. Virtual machines used for the deployment of components are modeled as graph nodes belonging to the *IaaS* node type. Graph nodes are used also to represent cloud service providers (*CSP* node type), which may *provide* any of the cloud service types defined above. An IaaS node may *host* or *use* an SaaS nodes, while an SaaS node may *use* another *SaaS* node. Finally, both graph nodes and edges may be assigned a set of *properties*, which provide several information related, for example, to the type of component (i.e., the type of service offered including

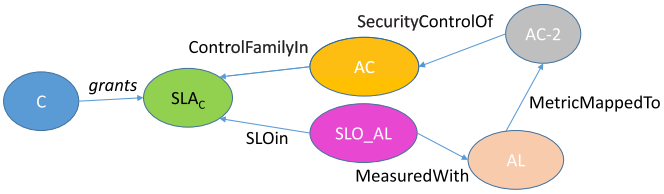

Fig. 2. An example MACM representation of a Security SLA.

web applications, storage services, etc.) or to the protocol used for communication, to deployment configuration information, to generic constraints, etc.

Besides the application architecture, the MACM formalism enables to represent security properties of components and providers in terms of Security SLAs, in compliance with the model discussed in the previous section. In particular, two node types are included to model both Security SLAs and Security SLA Templates (i.e., *SLA* and *SLAT* node types), depending on whether the guarantees stated in the agreement have been assessed or not. We assume that an assessed SLA is available for each provider, while individual components of the application are initially assigned a Security SLA Template based on their internal implementation and behavior, which must be properly assessed through a suitable composition procedure (described later in the paper). Hence, any service (belonging to either SaaS or IaaS node type) may *support* an SLAT, while any service or provider may *grant* an SLA. Table 1 summarizes the relationships available among MACM nodes.

As an example, Fig. 2 shows the SLA granted by a node `C` (i.e., SLA_C), declaring a control belonging to the Access Control family (AC), i.e., `AC-2 - ACCOUNT MANAGEMENT` and including an SLO (`SLO_AL`) built on top of metric `AUTHENTICATION_LEVEL` (AL). In the example, node `SLO_AL` belongs to the *SLO* node type and is in relationship *SLOin* with SLA_C. Node `AC`, representing the Access Control control family, belongs to the *ControlFamily* node type and is connected with SLA_C by means of the relationship *ControlFamilyIn*). The security control `AC-2 - ACCOUNT MANAGEMENT` is represented by the node `AC-2`, belonging to the *SecurityControl* node type, which is in *SecurityControlOf* relationship with node `AC`, representing the associated control family. Node `AL`, belonging to the *SecurityMetric* node type, represents the `AUTHENTICATION_LEVEL` security metric and is linked with

Table 1
MACM verbs.

Relationship	Start Node	End Node	Description
provides	CSP	SaaS, IaaS	identifies the software and compute services offered directly by a CSP
hosts	IaaS	SaaS	identifies the services hosted on a compute cloud resource
uses	SaaS	SaaS	describes a dependency relationship between two services
support	SaaS, IaaS	SLAT	identifies the security guarantees that a service is potentially able to grant based solely on its internal implementation and behavior
grant	CSP, SaaS, IaaS	SLA	identifies the security guarantees that a provider or service is actually able to grant based on the policies enforced locally and even considering the interconnections among components and the impact of cloud deployment

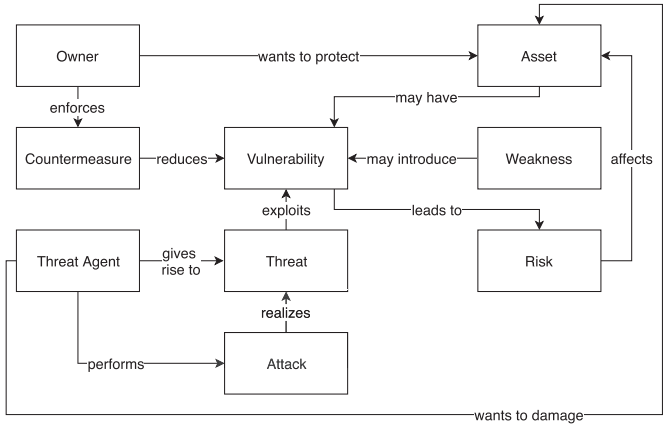

Fig. 3. Basic security concepts.

node AC-2 by means of the *MetricMappedTo* relationship. Moreover, AL is also linked to the SLO where it is used, i.e., SLO_AL, by means of the *MeasuredWith* relationship.

It is worth noting that several languages and tools exist for modeling cloud applications, such as CAMEL PaaSage Consortium (2016), CloudML SINTEF (2016) or Tosca OASIS (2013), and a MACM representation can be easily generated from such languages, as demonstrated for instance in the MUSA project, where CAMEL representations were considered in input to the development process. The MACM model can be managed by using graph-based databases, which enable to easily maintain and query complex graphs using graph-oriented languages (like Cypher). In Section 8.1, we will show the basic operations that can be performed on a MACM-based SLA model and that enable the security reasoning needed for the SLA Composition task, discussed in detail in Section 8.2.

3.3. The security data model

The model shown in Fig. 3 illustrates the main aspects involved in the security characterization of a system, merging together and extending different concepts introduced by several security standards and initiatives, including the ISO Common Criteria for Information Technology Security Evaluation (ISO 15408) Common Criteria (2017), the ISO 27000 standard family International Standard Organization (2018) and the Common Weakness Enumeration (CWE) project Mitre Corporation (2018b) by Mitre Corporation (Mitre Corporation (2018a)). According to the ISO terminology, a **threat** is *a potential cause of an unwanted incident, which can result in harm to a system or organization (an **asset**)*, a **vulnerability** is *a weakness of an asset or control that can be exploited by one or more threats*, and an **attack** is an *attempt to destroy, expose, alter, disable, steal or gain unauthorized access to or make unauthorized use of an asset* International Standard Organization (2018). A **threat agent** is the entity that gives rise to a threat and that concretely performs an attack against the asset to cause harm or damage. Existing vulnerabilities expose the asset to a **risk**, and their potential damage can be contained by enforcing proper **countermeasures**. The concept of **weakness**, not formalized explicitly in the ISO standard, has been highlighted by the Mitre Corporation, an American private, not-for-profit corporation that manages the Common Vulnerabilities and Exposures (CVE) system Mitre Corporation (2018a) and the CWE project Mitre Corporation (2018b): Mitre defines weaknesses as *a type of mistake in software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software*.

The above concepts are all involved in the design and assessment phases of the proposed process, since the knowledge of existing threats, vulnerabilities and weaknesses and of related risks is a prerequisite to first identify the actual security requirements (and the related countermeasures to apply to mitigate existing risks) of application components (i.e., the assets to protect), and then to determine which security parameters to assess and how to carry out the assessment.

In order to manage such concepts and enable the automation of the whole assessment process, we developed a *security data model* enriching the theoretical model reported in Fig. 3 with additional information related to assets, threats and countermeasures. The reference security data model is depicted in Fig. 4, where new introduced concepts and relationships are identified by dashed borders and lines, respectively.

The data model is built around the concepts of *component* and *component type*, which specialize generic assets based on the considered model of cloud application and that are directly mapped to the concept of threat. Basically, we associate component types (e.g., web applications, storage services, etc.) with a set of well-known threats that potentially affect all components of that type (e.g., threat Cross-site scripting potentially affects any web application). It is worth noting that the actual exposure of a given component to a threat depends on how the component behaves (e.g., a web application component is actually exposed to Cross-site Scripting only in presence of non-validated user-supplied data). To take this issue into account, we introduced the *Threat Condition* concept, modeling the set of conditions observed in a component behavior and internal implementation that may actually lead to the realization of a threat. Finally, as will be illustrated in detail in Section 7, the presented data model explicitly considers security controls as possible countermeasures against existing threats, and adopts the concept of risk level associated with a threat to select the proper controls to enforce at each component of the application.

The above discussed data model was implemented by a *Threat Catalogue*, available as open source at the address http://bitbucket.org/cerict/sla-model. In the current implementation, the catalogue includes almost 100 well-known threats against different component types, mainly gathered from standards and open repositories (e.g., the OWASP top 10 Threats OWASP (2016b), the OAUTH Threat Model Internet Engineering Task Force (IETF) (2013), the SSL Threat Model from SSLabs SSL Labs (2018), the CSA Top Threats Cloud Security Alliance (2019)) and from scientific papers. In particular, threats in the catalogue:

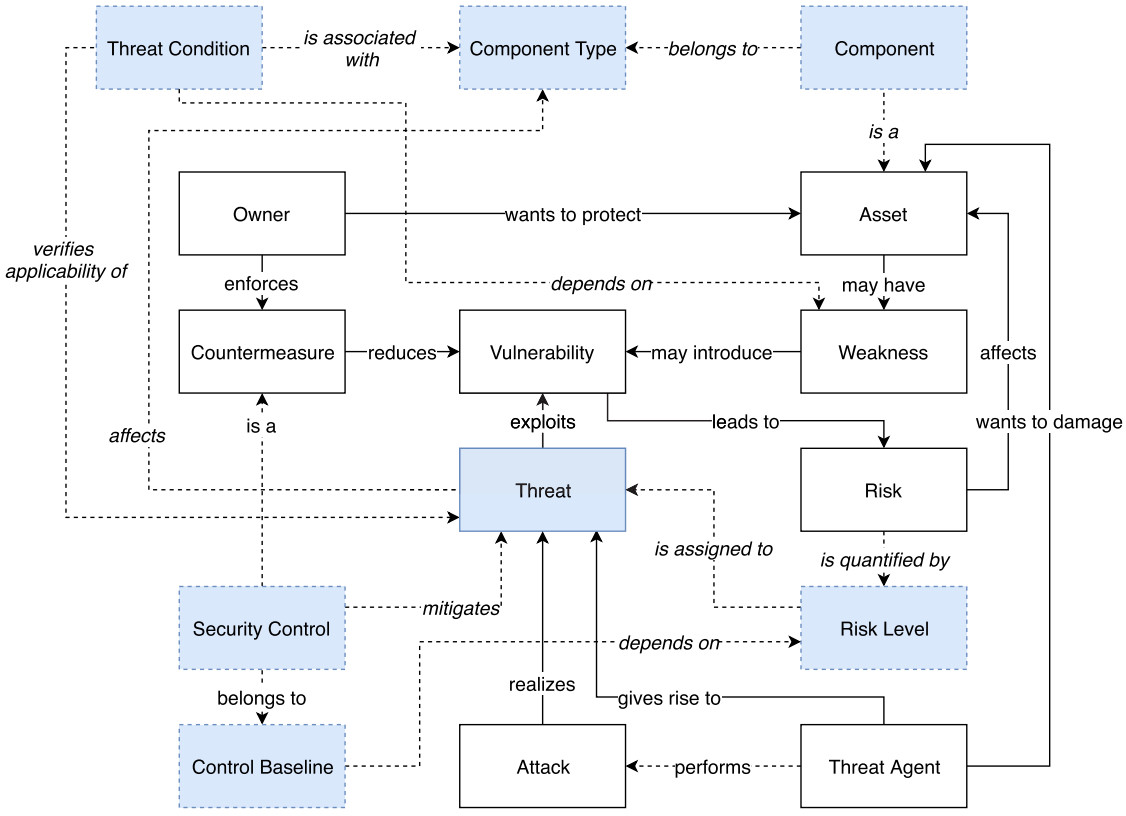

Fig. 4. The reference security data model.

- are mapped to the set of software component types to which they may potentially apply;
- are classified based on the STRIDE threat categories Microsoft Corporation (2016) (Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, Elevation of Privilege) and associated to the Confidentiality, Integrity and Availability requirements;
- are associated with a set of questions aimed at verifying whether they are actually applicable to a component, based on the component behavior and implementation. The conditions checked by these questions are aimed at identifying possible weaknesses in a component implementation, and are mainly derived from the CWE and other sources such as existing security guidelines and best practices;
- are mapped to a set of standard security controls (the NIST Security Control Framework is currently supported Joint Task Force Transformation Initiative I (2013)), which represent possible countermeasures to adopt in order to prevent their exploitation.

In addition to threat-related information, the catalogue includes further information related to the enforcement and monitoring of security controls (i.e., NIST security control baselines and security metrics), which is used to automate the identification of security controls and the construction of Security SLAs. In the following sections, we will present the development methodology and detail the use of the discussed data model in all the phases of the development process.

4. A Security SLA-based Security-by-Design methodology

In this section, we will introduce our *Security SLA-based Security-by-Design* (SSDE) cloud application development methodology, by providing on overview of the development phases and by also illustrating how these phases can be easily mapped onto the Scrum methodology.

As anticipated, the proposed methodology aims at coping with the challenges outlined in the analysis of the state of art by implementing the following requirements:

1. provide support for the security assessment of cloud applications at the design phase, when the system is only partially defined;
2. simplify and reduce the cost of security assessment, in order to make it affordable even to developers with basic security skills;
3. help identify the additional security mechanisms to integrate within the application in order to meet security requirements;
4. define an easy-to-automate secure software development process;
5. design security mechanisms by adopting, as much as possible, standard solutions and available frameworks.

As it will be clarified in the following, the SSDE methodology adopts Security SLAs to model the security properties of a cloud application, and makes it possible to carry out an evaluation of the existing risks and an automated assessment of the actually-granted security level, based on the deployment environment and conditions.

The SSDE methodology devises the process shown in Fig. 5, which involves three main phases, namely *Modeling*, *Per-Component Security Assessment* and *Per-Application Security assessment*.

In the *Modeling* phase, the functional architecture of the cloud application under development along with its deployment layout are specified by using the MACM formalism. In particular, as anticipated in Section 3.2, a cloud application is modeled as a collection of cooperating software components that provide a

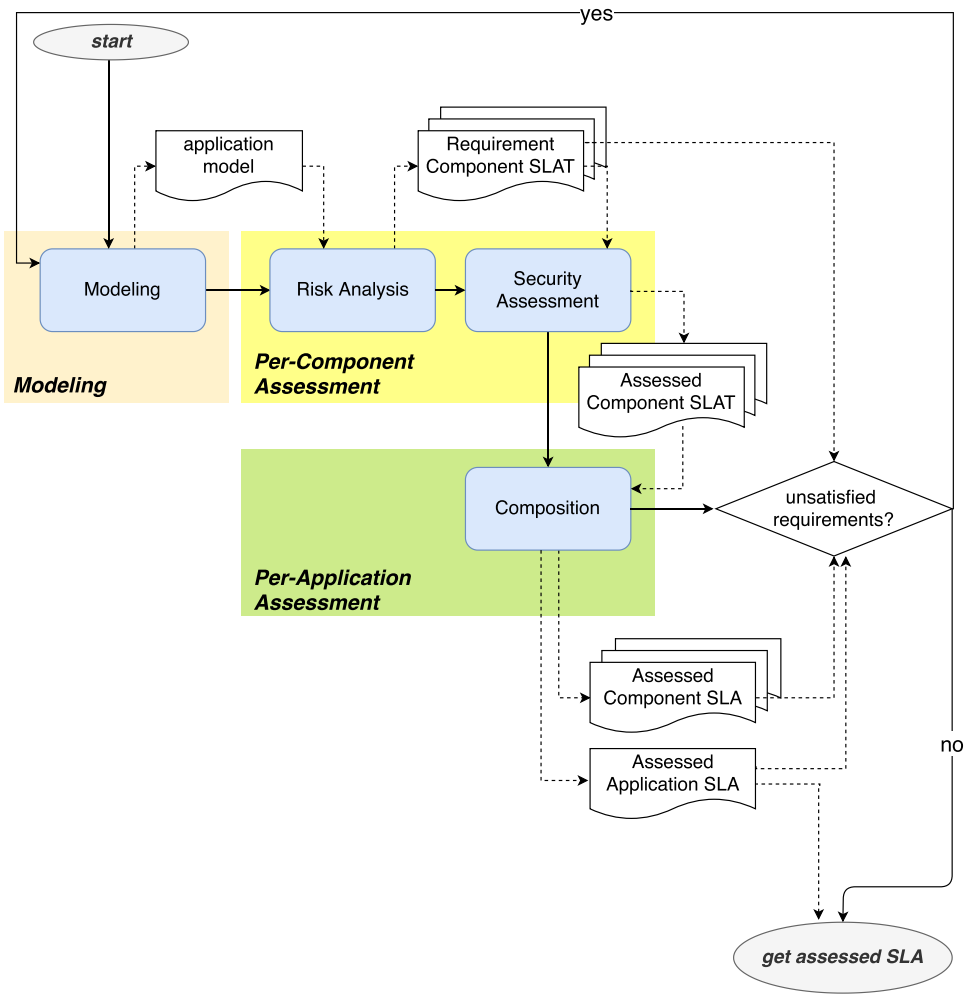

Fig. 5. The SLA-based Security-by-Design methodology flow.

given high-level functionality in an as-a-service fashion. Each of these components is directly mapped to a cloud resource, represented by either a SaaS service (for example if the functionality implemented by the component is directly offered by a provider in terms of remote APIs) or an IaaS service (if the component is a piece of software that must be deployed and executed in a virtual machine offered by a provider). By using the MACM formalism, a developer is able not only to define the main building blocks of an application (both on the functional and deployment levels), but also to specify some information on the nature of the application components that, as illustrated in detail in the following sections, will be fundamental for their security characterization.

In the *Per-Component Security Assessment* phase, a design and assessment process is conducted for each component of the application, independently of the others. In particular, in the *risk analysis* task, the security requirements for each application component are identified based on the evaluated risk while, in the *security assessment* task, an assessment of the security policies implemented internally is performed. More detailed, the risk analysis task aims at identifying the existing threats against the application and the countermeasures to apply in order to mitigate them. The security assessment task takes in input the results of risk analysis and verifies through a detailed assessment procedure the correct enforcement of required countermeasures in the involved components/services, based solely on the knowledge of their internal implementation.

The countermeasures identified by risk analysis to cope with existing threats, which represent the actual security requirements of a component, are specified in terms of standard security controls, and are collected in an SLA document, as discussed in Section 3.1. This document is referred to as a *Requirement Component SLA Template* (SLAT), since the included security terms comply with the SLA format and represent the required security guarantees, but not those actually granted. Similarly, also the security policies resulting from the security assessment of each component are specified in terms of security controls, and are enclosed in an SLA document (named **Assessed Component SLAT**). Even in this case, the document must be considered as a template, since it summarizes the security features theoretically implemented by each component without taking into account the impact of the interconnection and dependencies of components and of the deployment configuration.

It is worth pointing out that the risk analysis and security assessment tasks typically require strong security expertise. As discussed in Section 6.2, in the proposed process these tasks are partially automated, thanks to the adoption of the security data model introduced in Section 3.3, which collects all the relevant security information, thus requiring a limited effort from security experts and lower security design costs.

The last step of the process is the *Per-Application Security Assessment*, which aims at performing the overall application security assessment and produces the **Assessed Application SLA**.

This is the SLA that the application is actually able to grant, when deployed and running. This step assumes as input all the Assessed Component SLATs and, based on the information available from the application model (components/services interactions, CSPs involved, deployment details), evaluates the security controls actually implemented and the security level granted in production (i.e. in the operative environment). In particular, the effectiveness of a control is evaluated through an *SLA Composition process*, introduced in Rak (2017) and summarized in Section 8.2. The results of the *Per-Application Security Assessment* phase are commonly used as a feedback to the *Modeling* phase, in order to correct and adjust the application design, if needed, in an iterative process.

For example, if suggested security controls are not already available in the application architecture, the application can be enriched by either enforcing specific security mechanisms, provided by software libraries or third parties, or by updating security configurations or by updating the component-cloud resource mapping.

Before examining the details of the above phases, it is worth mentioning that the SSDE process can be easily integrated within the most popular software development methodologies. To support this claim, we will illustrate in the following subsection how the above phases can be integrated in the Scrum development methodology, which is probably the most popular approach adopted nowadays to manage the development of software products in an iterative, incremental and flexible way.

4.1. The SSDE process integrated in Scrum

According to the Scrum methodology, the development process is structured in a number of subsequent *Sprints* (i.e., short time intervals of one month or less), during which a potentially releasable product (called *Increment*) is created by the *Development Team*. The requirements of the software are summarized in a *Product Backlog*, controlled by the *Product Owner*, which includes the list of all the actions to be done. During the *Sprint Planning* phase, the Scrum team builds the *Sprint Backlog*, which includes the list of actions to perform during the next *Sprint* based on existing priorities: at the end of each *Sprint*, a *Sprint Review* takes place, involving the *Product Owner* and the *Development Team*, in order to inspect the *Increment* and adapt the *Product Backlog*, if needed.

With regard to the SSDE phases, it is worth remarking that the whole process relies on the application model, which is built in the *Modeling* phase. Therefore, *Modeling* should be considered as one of the primary activities to plan during the Sprint Planning phase, and should be assigned high priority. Once the model is available, the assessment tasks of the SSDE process can take place in the subsequent Sprint Review phases. In particular, the *risk analysis* task is performed as soon as the application model is ready, to obtain the list of initial security requirements of application components in terms of Requirement Component SLATs. Such requirements are included in the Sprint Backlog during the subsequent Planning phase, and are used to develop components from scratch or to identify possible COTS to integrate within the design. During the Sprint Review phases, as anticipated, the developer team performs also the other assessment tasks of the SSDE process. In particular, apart from the *risk analysis* activity that is launched at each development iteration to keep always updated the threat model of the system, the per-component and per-application *security assessment* tasks are also performed whenever the internal implementation of the components or their interconnections are updated, or in case a new component is introduced. As mentioned before, these activities allow the team to determine the actual SLA that the application and its components, in their current state, are able to guarantee. By comparing such SLAs with the Requirement Component SLAs (i.e., with the security requirements in the Backlog), the team can easily identify the requirements that are still not implemented and get an idea of the (security) issues that may arise. As a consequence, the development team may update the implementation of a component or even the global application architecture. In this case, a new high-priority *Modeling* activity will be launched in the subsequent Sprint, to have an up-to-date model of the application to be used for later assessment.

It is worth remarking that the above discussed development methodology enables to take into account the application evolution during subsequent Sprints: when a development action affects a component, the assessment tasks performed during the *Sprint Review* phase enable to immediately (and easily, since mostly automated processes are involved, as illustrated in detail in the next section) evaluate the effect of the change in terms of security, both on the component and on the overall application architecture.

As outlined in Section 2, there are only few proposals that alter the Scrum methodology in order to integrate security aspects. Our approach is different from the ones cited, in that our actions can take place during the traditional Scrum workflow without altering the Scrum principles, and can take place with the involvement of developers with very limited security skills. A Security Expert may be involved in the Sprint Review in the first phases on the project, to help the developers correctly understand the results of the analysis, and/or when decision are needed to address specific security issues (e.g., to select a security technology, to address specific security requirements, etc.).

5. Modeling phase

The *Modeling* phase is devoted to building the model of the cloud application and of its security properties, which will drive the activities of all steps of the SSDE process. Indeed, it entails a more general modeling activity aimed at supporting the automation of risk analysis and security assessment processes, and at providing a representation of security properties by means of Security SLAs. For these reasons, different models were developed and integrated.

In particular, as discussed in Section 3.3, we developed a reference security model extending existing security standards and including the security concepts that enable a guided identification (i) of the security threats against the system assets, (ii) of the countermeasures needed to mitigate such threats, and (iii) of the means to verify the countermeasures' correct implementation.

Moreover, we adopted the Security SLA model described in Section 3.1 to describe, in a structured way, what security capabilities a service is able to grant to customers and how security guarantees can be measured during operation in order to identify possible violations.

For what regards cloud application modeling, we adopted the MACM formalism illustrated in Section 3.2, which not only enables to describe the architecture of a cloud application, but that can also be profitably used to evaluate how the composition of different services and their deployment in different environments may affect the security granted by the application.

5.1. Example

To provide a concrete example of application of the proposed methodology, let us discuss a simple cloud application that will be used as a running example throughout the paper. The application (referred to as the "messaging cloud application" hereafter) allows users to upload and share personal documents through a web interface and to talk with one another thanks to a live chat service. It is made of three components: (i) the application core `W`, which is the web application that offers the upload and visualization functionalities to the users, (ii) the application database `DB`, which is the database used by the application core to store the users'

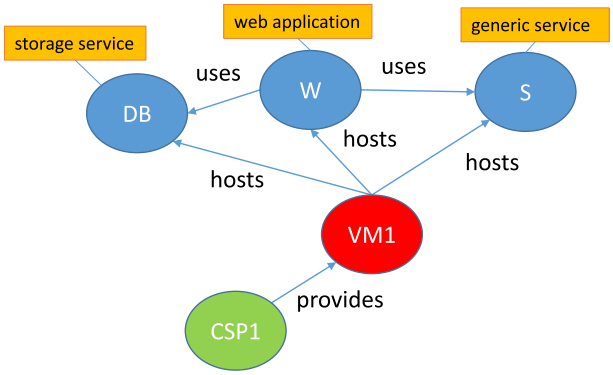

Fig. 6. An example MACM representation of a cloud application.

data, and (iii) the application chat service S, which is a live chat service used by the application core to provide its users with instant messaging functionalities.

Fig. 6 shows the MACM of the example messaging cloud application. As shown in the figure, the three application components are all modeled as *SaaS* nodes deployed on the same VM. Component W *uses* both DB and S, and virtual machine VM1, offered by CSP1, *hosts* the three application components. W is labeled as a web application, DB as a storage service and S as a generic service.

6. Per-Component Security Assessment phase - Risk Analysis

As already outlined in Section 4, the *Per-Component Security Assessment* phase of the SSDE process has two goals, namely (i) identify the security requirements of each component of the application and (ii) assess the security properties that application components are able to enforce. The first goal is accomplished through a risk analysis process, described in this section, which is devoted to identifying the main threats posed to the application in order to be able to elicit the related security needs and countermeasures, in terms of security controls to enforce. The second goal, discussed in Section 7, aims at assessing the security properties of the application, without taking into account yet possible influences due to component interconnections and deployment configurations.

As shown in Fig. 7, the risk analysis process includes a set of sub-tasks, namely *Threat Selection, Risk Evaluation, Security Control Selection, SLO Selection* and *SLAT Generation*. These sub-tasks can be almost completely automated thanks to the data model introduced in the previous section. In fact, as suggested by the picture and discussed in the next sub sections, only threat selection and risk evaluation require some - limited - user intervention, while the others can be carried out by only leveraging the information provided in the Threat Catalogue.

The next sub-sections show in detail the steps that lead to the definition of a threat model for the application (Section 6.1) and to the evaluation of the risk associated with each identified threat (Section 6.2), and illustrate the process adopted to select the appropriate countermeasures for existing risks (Section 6.3) and to build the Requirement SLA Templates for each component (Sections 6.4 and Sections 6.4).

6.1. Threat Selection

The threat selection sub-task aims at producing a threat model of the system in a (semi-)automated way. The proposed process involves only a limited human-interaction, as outlined in the following description, and mainly aims at identifying only threats to technical assets, taking into account the system architecture and the type of components involved. The automation relies on the availability of a complex knowledge base named *Threat Catalogue*, introduced in Section 3.3, which collects several threats along with associated information, including: (i) the assets potentially affected by the threat and (ii) the conditions that make the threat applicable to the asset, (iii) the STRIDE categories and (iv) a list of security controls (our current implementation supports controls belonging to the NIST framework). It is worth outlining that the Threat Catalogue is continually enriched by security experts, new best practices, new standards, and the conditions are described in natural language and in the form of questions.

The portion of the Threat Catalogue involved in the *Threat Selection* task is conceptually represented in Fig. 8, which shows an extract of the reference data model depicted in Fig. 4.

Hence, the (semi-) automated threat selection relies on a questionnaire-based approach: for each asset under analysis (i.e., each application component), we first list all the threats that may affect the asset, and then we generate a questionnaire that helps in discarding the non-applicable threats. In the practice, even for a not-expert user, threat selection results in a simple two-step task consisting of: (i) listing the components under analysis (automatically extracted from the MACM model), (ii) listing the associated threats (automatically retrieved from the database) and (iii) replying to the associated questions.

As already said, the Threat Catalogue maps the threats to STRIDE categories and to a list of security controls (belonging to the NIST framework) that can be used as countermeasures to mitigate the threat.

6.1.1. Example

In order to exemplify the above process, let us discuss its applicability to the simple cloud application introduced in the previous section.

The application components W, DB and S belong to the *SaaS* node type and, in particular, W belongs to the web application component type, DB is a storage service and S is a generic service. Let us focus on the web application component W and sketch the whole process. As mentioned, the SSDE user must first select the web application type for component W: this will automatically retrieve all the threats associated with web applications from the Threat Catalogue. In Table 2, a small subset of these threats is reported: as shown, each threat is associated with the related STRIDE category and with a set of questions aimed at determining whether the threat is of interest based on the behavior of the component.

Referring to the threats in the table, let us assume that, from an analysis of the behavior of component W, it results exposed to threats “Account Hijacking” and “Missing Function Level Access Control” (i.e., the SSDE user replied *yes* to at least one question associated with these threat).

6.2. Risk Evaluation

The risk evaluation sub-task enables to rate the level of risk associated with each threat identified by the previous task by automating the OWASP Risk Rating Methodology OWASP (2016a), which represents an easy-to-apply technique, widely used for cloud web-oriented applications. Such an approach evaluates risk based on **likelihood** and **impact** levels, both represented by means of 8 different parameters. In particular, likelihood parameters are classified into *threat agent factors* (including Skill level, Motive, Opportunity and Size) and *vulnerability factors* (including Ease of discovery, Ease of exploit, Awareness and Intrusion detection), while impact parameters take into account both *technical factors* (related to Loss of confidentiality, Loss of integrity, Loss of availability and Loss of accountability) and *business factors* (related to Financial damage, Reputation damage, Non-compliance and Privacy violation). Each of the above parameters is assigned a value in a range

Table 2
Extract of the threats selected for component W of the example messaging application.

Threat	STRIDE cat.	Condition	Countermeasures
Account Hijacking	SPOOFING	Does the component maintain account information?	AC-2, AC-2(12), AC-3, AC-7, AC-9, AC-17, AC-23, AC-9(2)
Cross-Site Request Forgery (CSRF)	TAMPERING	Does the component accept requests without checking their origin and trustworthiness?	SI-4,SI-10
Cross-Site Scripting (XSS)	TAMPERING	Are the inputs from users (forms, http requests) directly used without validation?	SI-10, RA-5
Man in the Middle attack	SPOOFING	Does the component communicate with other components without ensuring the authenticity and integrity of the communications?	SC-23, IA-2(13)
Buffer overflow exploitation	INFORMATION DISCLOSURE, DOS, TAMPERING	Does the component allow to write on memory without restrictions?	SI-16
Missing Function Level Access Control	ELEVATION OF PRIVILEGES	Does the component expose different functions to different users based on their access rights by only differentiating the user interface (without performing an actual authorization at each request)? Does the component manage functions/data with different access rights?	AC-6, AC-6(1), AC-3

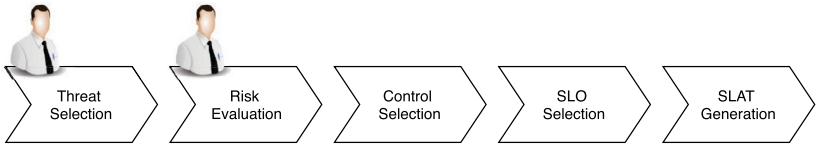

Fig. 7. Risk Analysis Process to build the Requirement SLA Template.

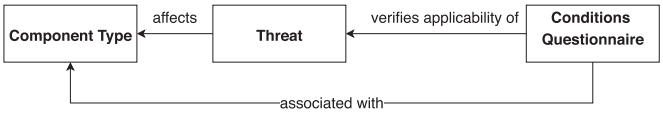

Fig. 8. Threat selection data model.

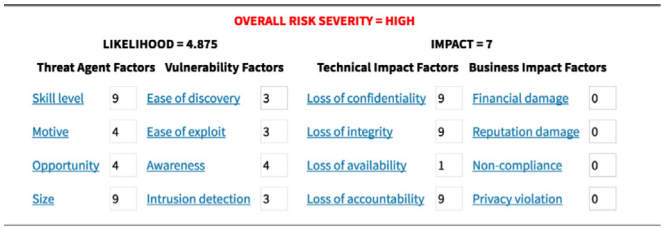

Fig. 9. The OWASP Risk Rating Methodology applied to the example messaging application.

from 0 to 9, and the overall likelihood and impact levels are computed as the average of the values of respective parameters. Values in [0;3[correspond to a LOW level, values in [3;6[correspond to a MEDIUM level, and values in [6;9] correspond to a HIGH level. The final risk is obtained by suitably combining respective likelihood and impact levels based on a pre-defined match table.

It is worth noting that, while the 4 business factors represented by *Financial damage*, *Reputation damage, Non-compliance* and *Privacy violation* are dependent on the specific application business model and must be set necessarily by the system owner, the other 12 parameters relate to technical aspects mainly depending on the considered threats and the involved assets. Based on this consideration, the assignment of proper values to likelihood and technical impact parameters can be simplified with the introduction of default values, suitably set by security experts based on the knowledge of the specific asset[2]

Moreover, in order to limit the human interaction, such an evaluation will be done not for each single specific threat, but for a set of threats grouped based on the STRIDE categories: as an example, the evaluator will be asked: *How do you consider a reputation damage due to a successful Denial of Service attack to your system?*. The developer may reply, for example, with the following values: minimal damage (assigning value 1 to the parameter), lost of big customers (value 4), lost of resources (value 5), brand damage (value 9). It is worth noting that our choice of using the OWASP Risk Rating Methodology is related to its simple and fast applicability in the context that we are taking into account, i.e., the integration of security design practices into agile methodologies even for SMEs and developers with minimal skills and resources. In fact, OWASP provides a means to express, in a way that can be easily understood by even non-experts (i.e., in terms of simple questions), a set of risk factors that enable to perform a quantitative risk evaluation.

6.2.1. Example

Let us consider again the example messaging application and apply the risk rating methodology to the two threats identified in the previous step, namely “Account Hijacking” and “Missing Function Level Access Control”. Fig. 9 reports the likelihood and impact values selected for the two threats: the last 4 values can be defined only by the user, so we left a 0 value as default.

With the values reported in the figure, likelihood level is MEDIUM and impact level (limited to technical impact) is HIGH. The resulting overall risk severity is therefore equal to HIGH for both threats affecting component W.

6.3. Security Control Selection

The security control selection sub-task is devoted to identifying the countermeasures to adopt, in terms of the security controls to enforce, in order to mitigate existing threats against considered assets, based on the actual risk level.

As anticipated in Section 3.3, the Threat Catalogue reports the association between threats and all possible countermeasures. This allows to automatically retrieve the full list of security controls

[2] Let us recall that threats in the catalogue are classified based on the asset they apply to, hence the same threat applied to different assets results in different catalogue entries. Default values for likelihood and technical impact parameters are assigned by security experts for each entry in the catalogue.

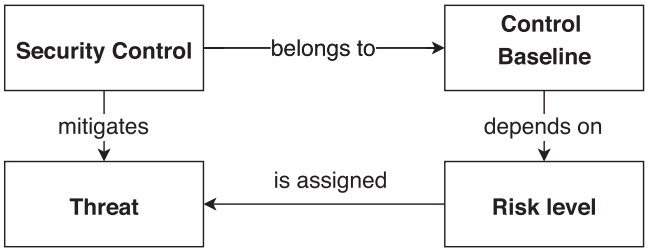

Fig. 10. Security control selection data model.

associated with the identified threats. Afterward, this set of controls, which is typically large, is refined by taking into account the actual risk level assigned to each threat in the risk evaluation task.

In order to perform this refinement, we adopt an approach similar to the one suggested by the NIST Control Framework, which suggests the adoption of security control baselines for low-impact, moderate-impact, and high-impact information systems, respectively, based on the result of the risk analysis. Given a certain level of risk for a system under observation, the respective baseline identifies the security controls to implement depending on that risk severity. Note that the three baselines are hierarchical: if a control belongs to the LOW impact baseline, then it will also appear in MEDIUM and HIGH impact baselines, too. As an example, NIST security control AC-12 belongs to the MEDIUM impact baseline (while it is not selected in the LOW-impact baseline): this means that this controls must be implemented, if required, not only in the systems with MEDIUM risk severity, but also in systems with HIGH risk severity.

Our process adopts a similar approach while relying on a different risk analysis process than the one used by NIST, and while assigning a level of risk to a single threat rather than to a system. In practice, given the level of risk assigned to a threat, the process allows to automatically select the controls that have been previously identified and that, at the same time, appear in the control baseline related to the same level of risk (we currently use the NIST baselines but ad-hoc baselines may be built).

Fig. 10 shows the view of the Threat Catalogue data model involved in the security control selection task, where the relationships among the concepts of threat, security controls, control baseline and risk level are outlined. Note that, thanks to this data model, the task can be completely automated.

6.3.1. Example

Let us consider again the example messaging application, whose component `W` has been found exposed to threats "Account Hijacking" and "Missing Function Level Access Control". In the first step of the task, the security controls associated with these two threats are retrieved from the catalogue, i.e., AC-2, AC-2(12), AC-3, AC-6, AC-6(1), AC-7, AC-9, AC-9(2), AC-17, AC-23, belonging to the Access Control family. In the second step, this set of controls is refined by selecting only the controls that were identified by NIST for the baseline related to high-impact systems. In this case, only AC-2, AC-2(12), AC-3, AC-6, AC-6(1), AC-7, AC-17 are selected Joint Task Force Transformation Initiative I (2013).

6.4. SLO selection

The SLO selection sub-task is devoted to identifying the concrete security guarantees that are required by application components based on the conducted risk analysis activity. These guarantees are expressed in terms of security SLOs and will be included in an SLA Requirement Template, which represents the output of the risk analysis process.

Similarly to the previous task, even this task can be completely automated thanks to the information included in the Threat Catalogue. In particular, as anticipated in Section 3.1, the catalogue includes a collection of security metrics that are mapped to security controls: basically, this mapping identifies which security metrics offer an evidence to customers about the correct implementation of a given security control. Moreover, in order to enable the automation of the SLO selection, for each metric, the catalogue reports also a corresponding default SLO (i.e., a boolean expression involving the metric and default thresholds).

Hence, the SLO selection task simply consists of (i) retrieving the metrics associated with the controls resulting from the security control selection sub-task, for each threat identified for the component under assessment, and in (ii) retrieving the corresponding SLOs as default values. Clearly, the user can update these SLOs according to his/her requirements if needed, in order to define the actual required security guarantees. Otherwise, the process will not need any user intervention.

Table 3 reports the metrics retrieved for the example messaging application, which are related to the security controls selected in the previous phase. As shown, some controls are mapped to more than one metric, which can be used together to evaluate, from different points of view, whether the control is correctly implemented. Moreover, one metric may be associated with more than one control. This condition, not shown in the table, means that the same metric is useful to evaluate the effectiveness of different controls. Finally, some controls are not mapped to any metric: this means that they cannot be monitored directly, and their effectiveness is to be verified in a different way (e.g., by certification).

6.4.1. Example

Related to the simple messaging application, let us assume that the following SLOs are set:

- `AUTHENTICATION_LEVEL > 1`: grants that an authentication mechanism stronger than the simple username/password is in place;
- `ACC_CONTR_LOGGING = yes`: grants that all access attempts are logged.

6.5. SLAT generation

Finally, the SLAT generation sub-task essentially consists in putting together the information collected in previous sub-tasks in order to build a **Per-Component Requirement SLAT** for each component, which summarizes the requested security controls and SLOs. We do not report here the complete SLA for the example messaging application for brevity sake, but the interested reader can have a look at the resources published on SPECS project web site (2016), including videos and presentations.[3].

7. Per-Component Security Assessment phase - Security Assessment

The security assessment task of the *Per-Component Security Assessment* phase has the goal of verifying that the required security controls (listed in the *Requirement SLA Template* produced in the previous task) are correctly implemented and configured in the component. The task produces the **Per-component Assessed SLA Templates**, i.e., the SLA Templates potentially *supported* by

[3] An example SLA for a secure web container service is available at https://bitbucket.org/specs-team/specs-utility-xml-sla-framework/src/master/examples/specs-SLATemplate-SecureWebServer_M30_NIST.xml.

Table 3
Metrics and SLOs selected for component W of the example messaging application.

Control	Metric	Description	SLO
AC-2	AUTHENTICATION_LEVEL	Level of authentication mechanism in place: 0:no authentication, 1:Login/Password, 2:two factor auth, 3:white-list of IPs + credentials, 4:combination of mechanisms (IP, port number, certificate or public keys, credentials etc.)	> 0
	SHARED_ACCOUNTS_PERC	Percentage of users with access to shared accounts	≤ 50
AC-3	ACC_CONTR_CORRECT	Total access control rules fulfillment in place	$=$ *yes*
AC-6	*none*	*none*	*none*
AC-6(1)	*none*	*none*	*none*
AC-7	ACC_CONTR_LOGGING	Logging of all tentative of access in place	$=$ *yes*
	USR_AUTH_BEHAV_CHG	More than one station or browser used for connection by the user agent	$=$ *yes*
AC-17	REMOTE_ACCESSES_PERC	Percentage of remote access points used to gain unauthorized access	≤ 10
	SRC_REQ_LOCATION	List of countries allowed to make requests	$=$

each component of the application (before considering other dependencies that will be verified and assessed in the last step).

According to initial methodology requirements, also developers not specifically skilled in security should be able to perform security assessment. Consequently, we chose to perform the assessment through a code-review approach, by discarding in the current implementation other techniques, like penetration tests, which need specific expertise, are time-costly and can hardly be automated. However, some research toward the integration of such techniques in the development process has been conducted in recent work Casola et al. (2018c).

Code review relies on the idea of submitting to the developers suitable questionnaires that list the checks to perform on the code of the application. Accordingly, our approach relies on the idea of generating a dedicated checklist for each component of the application, based on its type and on the requirements (i.e., the security controls) listed in the associated *Requirement SLA Template*. In order to enable this approach, we collected well-known assessment questionnaires and best practices and mapped (by means of a dedicated relationship in our data model) each question (or best practice) to one or more security controls as a means to verify their correct implementation. At state of art, the following sources have been used:

- Security Controls definition from NIST SP-800-53;
- Application Security Verification Standard (ASVS) questions by OWASP OWASP (2019);
- Berkley DB Best Practices Berkeley DB Best practice (2017);
- CAIQ by Cloud Security Alliance (CSA) Cloud Security Alliance (2011a).

The questions and best practices are formulated, commonly, in a way that does not need very high security skills, so developers with basic security skills are usually able to correctly reply, offering a concrete evaluation of the security assessment. The resulting *Per-Component Security Assessment* task is probably the most time-expensive activity in the SSDE process, but the adoption of the proposed security models and assessment techniques has the great advantage that it can be executed even partially during common development activities, and regularly updated during the execution of other tasks.

7.1. Example

To provide a concrete example, let us assume that an SSDE user aims at assessing the web application W for the security control SI-10 INFORMATION INPUT VALIDATION. Associated with this control, our assessment questionnaire reports the conditions listed in Section V5.1 - Input Validation Requirements - of the OWASP Application Security Verification Standard 4.0 OWASP (2019), namely:

- Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, or environment variables).
- Verify that frameworks protect against mass parameter assignment attacks, or that the application has countermeasures to protect against unsafe parameter assignment, such as marking fields private or similar.
- Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc.) is validated using positive validation (whitelisting).
- Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g., credit card numbers or telephone, or validating that two related fields are reasonable, such as checking that suburb and zip/postcode match).
- Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content.

The control is considered as correctly implemented only if all the above conditions are satisfied in all the software modules to which the above conditions apply. It is worth noting that all the replies to the presented questionnaire can be stored and easily used to keep track of partially implemented controls, in order to identify the conditions and checks that result still missing, and drive the developer to the correct implementation of the control in the component as a whole.

8. Per-Application Security Assessment phase

The *Per-Application Security Assessment* phase is devoted to building the final SLA of the application and of its components. It relies upon the *SLA Composition* technique, which suitably combines the security policies implemented by each component by taking into account the existing component dependencies and the impact of deployment choices, in order to identify the set of security controls that can be declared as correctly implemented and that can be actually granted in the SLA.

The SLA Composition process takes in input the MACM representation of the application and the MACM representation of all the involved SLAs and (Assessed Component) SLATs, and operates a set of manipulations that allow to obtain an equivalent representation allowing to express and analyze more easily the logical conditions involving security controls. In particular, the process entails the construction of a first-order logic Prolog knowledge base (discussed in Section 8.1) that includes (i) a set of *facts* regarding the declaration of security controls in respective SLAs and SLATs, and (ii) a set of logic *rules* tailored to each specific security control and aimed at determining when a control can be considered as correctly implemented in a component. The algorithm used for SLA Composition, illustrated in Section 8.2, dynamically generates the composition rules, and queries the

knowledge base to build, step-by-step, the SLA granted by each node in the application's MACM model.

8.1. SLA Composition knowledge base

Assertions and rules used for SLA Composition are built by querying the MACM representation in input in order to retrieve the needed information on the nodes composing the application, their mutual relationships and the associated SLAs and SLATs. In particular, the Cypher language can be used to perform SQL-like queries such as:

- retrieve all the nodes belonging to the IaaS cloud service type: `$match (n:IaaS:service) return (n)$`
- list all the components hosted by the vm with id equal to '23': `$match (vm {id:'23'}), (n:service), (vm)-[:hosts]->(n) return (n)$`
- retrieve the SLAT supported by a node S: `$match (n:service {id:'S'}), (n)-[:supports]->(slat) return (slat)$`
- retrieve the SLA granted by a node S: `$match (n:service {id:'S'}), (n)-[:grants]->(sla) return (sla)$`
- list all the controls included in the SLAT associated with node 'S': `$match (sc:SecurityControl), (sc)-(:SecurityControlIn)-> (cf)-(:ControlFamilyIn)->(slat:SLAT)<- (:supports)-(n:service {id:'S'}$`

where n is the number of nodes in the MACM graph.

As anticipated, the SLA Composition knowledge base includes *assertions* on the presence of a given security control in an SLA or SLAT associated with a node. We use the boolean variables $SLA(c, S)$ and $SLAT(c, S)$ to indicate the presence of control c in the SLA and in the SLAT associated with a node S, respectively. At the beginning of the *Per-Application Security Assessment* phase, the SLAs and SLATs in the application MACM are translated into facts: as an example, if the SLA (or SLAT) of node S contains the control c, we assert $SLA(c, S)$ (or $SLAT(c, S)$).

In addition to facts, the knowledge base includes a set of rules, used to assess the correct enforcement of security controls in each node of the application. Rules depend on specific controls and are built dynamically based on the structure of the MACM representation in input. In fact, as already said, the correct implementation of a security control in a component S depends not only on how S *implements* the control internally, but also on the interactions with other components and on how these components implement, in turn, the control under consideration.

It is worth outlining that it is not possible to identify a rule that can be applied in general to all security controls, but the analysis must be carried out control-by-control. For this reason, we analyzed one-by-one the controls belonging to the NIST framework and built different *composition functions* for each control, taking into account the possible relationships in which the component declaring that control may be (e.g., the node may be a VM *hosting* another component, or it may be a generic service *using* another component, etc.), along with the role taken in each relationship (e.g., the component may use or be used by another one, it may host are be hosted by another one, etc.).

In most part, we found that a composition function may be applied to an entire family of controls rather than to only a single control (e.g., it is the case of the Access Control - AC family). The result of this analysis activity is a set of *composition tables* including pre-built composition functions for each security control or control family. As discussed in the next section, the specific composition functions to apply are identified from these tables and the corresponding composition rules are generated based on the MACM of the application under consideration.

All things considered, a security control c_j belonging to the set of controls of interest SC is considered as correctly enforced by an application app if and only if it is declared in the SLA granted by each of the application components. Let N be the set of the n components N_i of the application, the above condition can be formalized as follows:

$$SLA(c_j, app) = \bigwedge_{N_i \in N}^{n} SLA(c_j, N_i) \tag{1}$$

Summarizing, we state that the SLA of the overall application depends on the SLAs of the components that, in turn, depend on the respective SLATs and on the SLAs of components they are in relationship with.

8.1.1. Example

In order to show with an example the relevance of composition and the impact of deployment on the enforced security policies, let us consider the assessment of control AC-3 (that states the enforcement of an access control mechanism) for the cloud messaging application introduced before. As shown in Fig. 6, it is made of a web application W, a database DB and a chat service S, all hosted by a virtual machine VM and provided by one CSP.

It is worth noting that if DB implements a security control related to Access Control, but VM does not have any access control (e.g., allowing access to everyone), we can assess that the deployed DB does not correctly implement the security control.

Moreover, we can assess that if both DB and the underlying VM correctly implement an access control policy, it is not relevant whether it is also implemented by W, that *uses* the DB.

This result can be easily exploited by applying the following rule:

$$SLA(AC-3, DB) = SLAT(AC-3, DB) \wedge SLA(AC-3, VM) \tag{2}$$

Regarding the node W, it is hosted by a VM and uses a DB. So, in this case, the assessment of the AC-3 control on W depends on the implementation of the control on W, on the DB and on the Virtual Machine (SLAT of W, SLA of the VM and SLA of DB).

This rule can be expressed by composing the different available SLAs and SLATs as follows:

$$\begin{aligned} SLA(AC-3, W) = {} & SLAT(AC-3, W) \wedge SLA(AC-3, VM) \\ & \wedge SLA(AC-3, DB) \end{aligned} \tag{3}$$

The example illustrates that the assessment for the same control may vary depending on the relationships that affect the nodes under analysis in many different ways. Accordingly, it is impossible to write a single rule that can be applied in all cases, but we need to generate different composition rules taking into account the security control, the relationships and the role that the nodes have in such relationships. In the next sections we will present in detail the SLA composition algorithm.

8.2. SLA Composition algorithm

As anticipated, the SLA Composition rules are dynamically generated based on the input MACM representation, which includes all the SLAs and SLATs granted and supported by the application components. In particular, the proposed model assumes that, after *Per-Component Security Assessment*, only CSP nodes have an associated SLA, while all the other nodes just support SLATs.

Composition rules are generated with a string manipulation approach and using a composition table; the information to manipulate is organized as shown in Table 4 in the case of the AC control family. Each row of the table contains a string template in the three variables <X>, <S>, <T>. <S> and <T> represent the id of the source and target nodes of the relationship under

Table 4
Composition rules for the AC control family.

Relationship	Role	function
start	-	`slat(sc,<S>) AND <X>`
provides	source	`1`
provides	target	`sla(sc,<S>) AND <X>`
hosts	source	`<X>`
hosts	target	`slat(sc,<S>) AND <X>`
uses	source	`slat(sc,<T>) AND <X>`
uses	target	`<X>`

analysis. Variable <X>, instead, represents the remaining part of the composition rule to be defined in the following iterations.

As already mentioned, we generate a composition rule for each node in the MACM and for each security control (family). Given a security control (family), we select one node of the MACM and generate a string that represents the composition function. The starting string is the one associated to the row labeled as `start` in the table. The composition algorithm iterates over all relationships in which the node is involved. In the table there are the composition functions to be adopted for each relationship, depending on the role the node under analysis has in the relationship. The starting string is modified at each iteration, substituting the <X> with the new composition function and the <S> and <T> with the name of the id of the source and target nodes in the relationship, respectively. At the end of the iterations, when all relationships have been taken in consideration, the last <X> is removed.

As mentioned, the composition algorithm is the result of previous work and its description, which is pretty complex, is out of the scope of this paper. Hence, we invite the interested reader to check reference Rak (2017) where all details are discussed.

8.2.1. Example

As an example, let us consider the case of the node VM of the web chat application example, and let us build the composition rules for the control AC-3. The security control belongs to the AC family, so we can use the Table 4, already presented. The starting template string is `slat(AC-3,<S>) AND <X>`. In this example <S> is the node under analysis VM, and so the resulting initial string is `slat(AC-3,VM) AND <X>`. As shown in Fig. 6, the node VM is involved in the relationships *provides* with the role of *target* (having as *source* the CSP), and in three relationships *hosts* with the role of *source* (the corresponding targets are W, DB and S). Accordingly, the composition rule will be modified as follows:

- **start** the starting string is: `slat(AC-3,VM) AND <X>`.
- **step 1** the *provides* relationship with role "target" modifies the rule into: `slat(AC-3,VM) AND sla(AC-3,CSP) AND <X>`.
- **steps 2,3 and 4** the *hosts* relationship with role "sources" leaves the string <X> unmodified: `slat(AC-3,VM) AND sla(AC-3,CSP) AND <X>`.
- **end** all the relationships have been taken in consideration; at the end of the process, the final rule is: `slat(AC-3,VM) AND sla(AC-3,CSP)`.

The composition function obtained (`slat(AC-3,VM) AND sla(AC-3,CSP)`) asserts that the security control AC-3 is granted by the SLA offered by the VM if it is supported by the VM SLAT and granted by the CSP. Such string is then translated into a Prolog predicate. It is worth noting that rules for the same control, but applied to different nodes, may be completely different. At the state of the art, we have built the composition rule for all the NIST families.

9. Validation

The methodology was adopted in the context of the H2020-MUSA European Project that uses a DevOps methodology to design multi-cloud applications.

In order to validate the approach, we developed the SLA Generator Tool[4] to support all the phases of the process. The validation was carried out by testing the methodology within two MUSA projects' case studies. We had the chance to use it during the development of two real-world applications (i.e., an airline flight scheduling application and a smart mobility service) with a large team of heterogeneous developers. Some public details on these two multi-cloud secure applications are available in the project deliverable D5.1-MUSA case studies work plan D5.1 MUSA case studies work plan (2018).

During this experimental phase, we evaluated the economic benefits of the methodology in terms of its efficiency, its usability and in terms of the time spent to design security solutions.

In the next sections, we will first discuss how the proposed methodology can be mapped onto the steps of the MUSA DevOps workflow (Section 9.1), and then we will illustrate the validation methodology (Section 9.2.2) that includes a presentation of the evaluation team and an illustration of the validation process, then the obtained results will be presented (Section 9.3). Finally, in Section 9.4 we will provide a conclusive discussion of the benefits introduced by our proposal and of its main limitations.

9.1. The SSDE process in MUSA DevOps workflow

The Security-by-Design process based on Security SLAs introduced above was firstly devised in the context of the MUSA project, where a DevOps workflow was identified for the development of secure (multi-)cloud applications. The MUSA DevOps workflow includes seven main phases: *Modeling*, *Risk Assessment*, *Cloud Service Selection and Decision Support*, *SLA Generation*, *SLA Composition*, *Deployment Planning*, *Deployment* and *Execution*. The details of the full solution are available on the project website and in Rios et al. (2015). Fig. 11 shows the overall MUSA workflow and outlines the mapping with the SSDE phases.

The *Modeling* phase is the same in both MUSA and SSDE processes, while the *Risk Assessment* phase of the MUSA workflow corresponds to the *risk analysis* task of the *Per-Component Security Assessment* of the SSDE process. As previously mentioned, *risk analysis* requires an interaction with human users that are asked to reply to simple questions.

The SSDE *Per-Component Security Assessment* activity takes place in the MUSA *SLA Generation* phase, where the results of the risk analysis are translated into the Requirement Component SLATs and assessed according to the methodology presented in Section 7. The SSDE *Per-Application Security Assessment* phase is explicitly mapped onto the MUSA *SLA Composition* phase, by applying the techniques illustrated in Section 8.2.

It is worth noting that, in MUSA, the feedback takes place after the *SLA Composition* phase, providing the developer with a summary of the security requirements still uncovered, and outlining the need of modifying the deployment architecture and/or of changing some of the choices made during the previous steps.

9.2. Validation methodology

The SLA Generator was evaluated with respect to different quality metrics that include: Availability, Efficiency, Usability, Interoperability, Reusability, Testability and Time consumed.

[4] Available at: https://musa-project.eu/content/service-level-agreement-support.

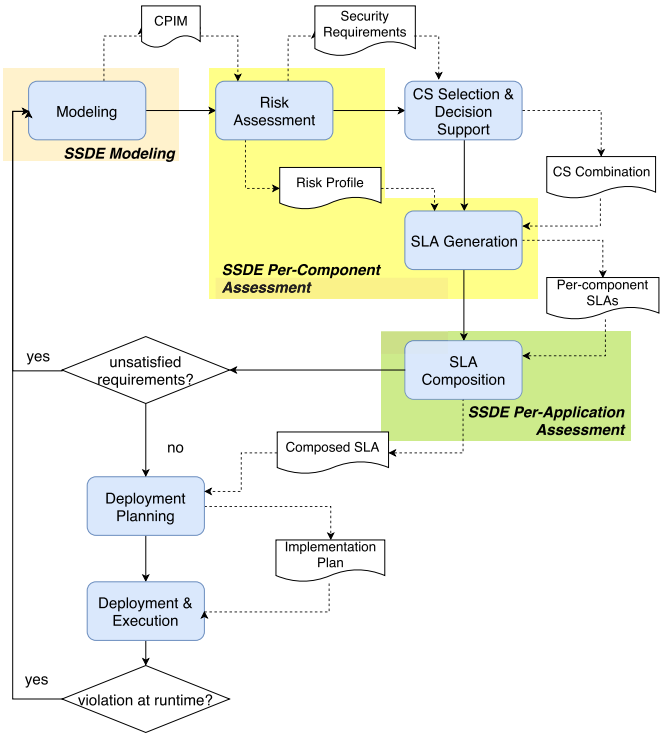

Fig. 11. The MUSA DevOps Workflow and mapping with the methodology.

Due to space limitations, in the following discussion, we will focus on three of those metrics, namely *Efficiency*, *Usability* and *Time consumed*, in order to highlight how the process can be easily integrated into existing development methodologies with a minimum effort, in compliance with business and economic requirements. The interested reader can read the MUSA public deliverable D5.5 Results of final evaluation of MUSA framework (2018) with all evaluations.

In order to carry out the evaluation, we used online questionnaires and informal interviews to collect and analyze results and feedback. Furthermore, we selected a team of evaluators that was not skilled on the functional and security aspects of the two case studies, and we measured the effort needed to use the tool.

9.2.1. Composition of the evaluator teams

We adopted different criteria to select the ideal team, mainly to avoid the influence of existing knowledge when the questionnaire is answered. First of all, the respondents should not be familiar with the project concepts, in order to avoid the influence of existing experience with the project. Second, the respondents should have a heterogeneous level of experience, as the subjects of the questionnaires should be equally understandable and usable both for junior employees and senior professionals. Lastly, the respondents should not have significant knowledge about security. Nevertheless, since most employees in IT companies are directly or indirectly involved with security issues at different levels during their career, this factor cannot be completely excluded.

Two different evaluator teams were involved in the evaluation process. Overall, we had nine evaluators for the final tool evaluation, including a software architect, a business analyst, a system administrator, a system operator and some developers. Fig. 12 briefly summarizes the composition of the teams.

9.2.2. Validation process

As already mentioned, we prepared an online questionnaire. Questions were organized according to multiple objectives. On the one hand, we asked for an overall evaluation of the tool usability

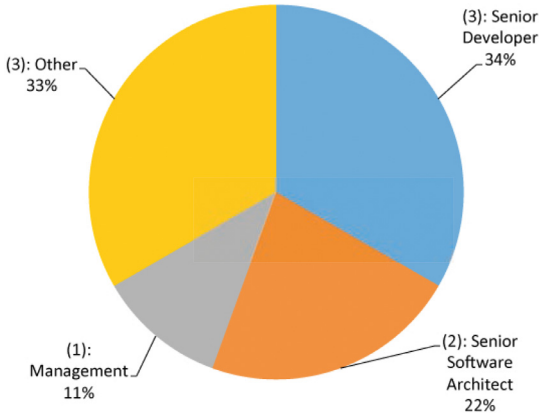

Fig. 12. Evaluator Teams composition.

and users' interest and, on the other hand, we asked to evaluate specific quality metrics (e.g., *Efficiency* and *Usability*). For each metric, multiple-choice questions were proposed. We adopted the following approach to provide a quantitative evaluation of each criteria analyzed: being n the number of answers received (excluding the not applicable answers, denoted by *n/a*), being i one of the five agreement categories (from `Strongly agree` to `Strongly disagree`), and P_i the number of answers replied for the agreement category i, the fulfillment degree of each category is evaluated by weighting each reply with the following weights w_i:

- Strongly agree: $w_5 = 5$
- Rather agree: $w_4 = 4$
- Difficult to say: $w_3 = 3$
- Rather disagree: $w_2 = 2$
- Strongly disagree: $w_1 = 1$

The formula to compute the metric category factor is therefore:

Fulfillment degree $= 100 * (w_5 * P_5 + w_4 * P_4 + w_3 * P_3 + w_2 * P_2 + w_1 * P_1)/(n * 5)$

Answers *n/a* (not applicable) are neglected in the formula.

In addition, we asked each evaluator to measure the time spent on each tool, in order to fully correlate their replies with the issues met during the adoption of the approach.

According to the agile methodology philosophy, all activities started with an informal planning. In particular, the members of the evaluator team had a teleconference of about 1 hour with the developers of involved tools (in fact, even other tools were experimented together with the SLA Generator) before answering the questionnaires, and received both project deliverables and demo videos to consult, if needed. Moreover, they were able to send emails and other requests to the developers during the evaluation. Each evaluator performed the whole process, and all of them were able to perform the SLA assessment by themselves with the only aid of the offered documentation, videos and email support.

9.3. Validation results

Figs. 13 and 14 report the questions and answers we collected respect to the Efficiency and Usability metrics, respectively. Table 5 summarizes the time spent by evaluator teams on the activities related to security assessment.

Regarding the tool's Efficiency, the proposed questionnaire includes five specific questions. In particular, two of them focused on the proper selection of security metrics (as described in Section 5), the other three questions aimed at verifying if evaluators were actually able to catch all the security aspects within the process and to perform the security assessment at both component and application levels.

Fig. 13. SSDE process and SLA Generator Tool Efficiency.

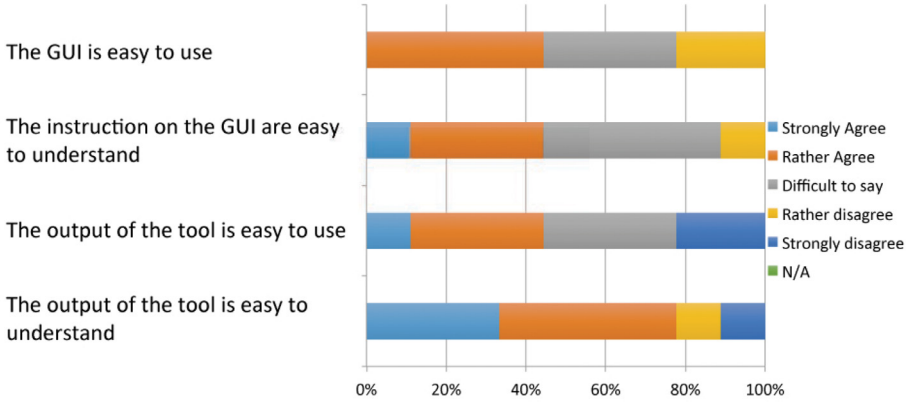

Fig. 14. SSDE process and SLA Generator Tool Usability.

Table 5
Time spent on security assessment phases.

Time spent to:	Case Study	Avg	Median
Create a component SLA template	CS1	100 mins	30 mins
Create a component SLA template	CS2	32 mins	20 mins
Perform a self-assessment of a single component	CS1	78 mins	60 mins
Perform a self-assessment of a single component	CS2	24 mins	10 mins
Perform self-assessment of all components	CS1	326 mins	240 mins
Perform self-assessment of all components	CS2	154 mins	77 mins
Create a composite SLA	CS1	380 mins	60 mins
Create a component SLA	CS2	16 mins	10 mins

As shown in Fig. 13, the results of the evaluation outline that all the evaluators gave a positive reply to the efficiency of the proposed process and to its applicability. According to evaluators' opinion, the adoption of the SLA Generator gives to end users a lot of well-structured and motivated information on threats, security controls and metrics, and is able to draw the attention to unknown security aspects for application design and help the users to take more informed decisions. The answers regarding Efficiency of the tool got a very good acceptance degree and we can see fairly positive results for all questions, which means that the SLA Generator got a very good result on efficiency and usefulness.

Usability-related questions, reported in Fig. 14, aim at evaluating the difficulties that the users may meet when applying the methodology and using the tool. As already mentioned, the evaluators were not security-experts, and so the capability of the tool to perform security assessment in an easy way becomes critical. For this metric, the replies also contain some negative feedback: evaluators outlined that the large amount of information, produced and presented, may be a drawback (a chore to scroll through, for an experienced user, and information overload for an unexperienced user). As a consequence, the users tend to overlook/over-click important parts and this, in general, is one of the factors that may result in increasing security risks.

This consideration, from our point of view, is very relevant: the tool should be able to hide as much as possible information not directly of interest for customers but, at the same time, it should give the right amount of information to take informed decisions and avoid other kind of risks Ganin et al. (2017). It is worth noting, however, that the prototype nature of the tool (Technology Readiness Level 5) may have affected such evaluation. In the future versions of the tool implementation, we will retain the same information, but we will show it only on user request.

Efficiency and Usability have shown that the proposed process is effective, and that even teams with limited security expertise are able to use it. To conclude our analysis on the concrete impact that the process may have over the development costs, during the evaluation process the evaluators registered the time spent for the full process.

In particular, we registered the time spent in creating a single (Requirement) SLA Template, the time spent to perform a security assessment for a single components and for a whole application. Table 5 summarizes the time spent by evaluators during the different phases of the process.

There is some dispersion of the results for the two case studies (named CS1 and CS2), ranging from 10 minutes to 10 hours for the whole application. In particular, the longest time response, 10 hours, is related to an evaluator un-experienced with security, but

very inquiring. This time includes his in-depth analysis into security terms, threats, security controls and assessment expressions; he wanted to gain some knowledge to be able to understand and fill the forms of the tool. Because of missing explanations, tool-tips, etc., a lot of time was spent on googling and reading Wikipedia. The massive information overload was hard to overcome. Removing the extreme values, the average time spent is 30 minutes to create one component and 60 minutes to complete the assessment. Different values for the two case studies also depend on the number of security requirements and on the total number of software components, not discussed here for brevity's sake but available in the MUSA Deliverable D5.5 MUSA project (2017).

9.4. Discussions and limitations

As discussed in the paper, integrating security design in modern software development methodologies, as Agile and DevOps, is quite complicated, mainly due to the difficulties in designing and assessing proper security design choices. The proposed methodology, along with the associated models and tools, aims at supporting developers, typically provided with few security skills, during the whole software life cycle. The results obtained in the evaluation of efficiency, usability and time spent in the assessment outlined that the proposed approach introduces a minimal increase in design cost, if compared with the number of person-months spent in security design and in "a-posteriori" assessment activities. According to the evaluators' opinion, the tool is very valuable, as it provides a lot of information on threats, security controls and metrics and helps even non-expert users to take more informed decisions related to security. Moreover, the satisfying level of usability of the tool enables to dramatically simplify the overall security assessment process. As previously outlined, however, a drawback is represented by the large amount of information that is generated and presented by the tool, which may be partly overlooked or neglected by both expert and non-expert users and that must be kept contained to maximize usability.

It is worth pointing out that the goal of our methodology is to help developers, especially those provided with limited security skills, to identify, as soon as possible, the main threats and vulnerabilities affecting the application under development, in compliance with the principles of Security-by-Design. The validation process showed that the methodology is able to correctly identify a set of applicable threats, but it does not provide any means to prove the completeness of the set of identified threats. The effectiveness of the methodology naturally depends on the completeness of the Threat Catalogue, and therefore it is fundamental that it is continually updated to include more threats and vulnerabilities and take into account a wider set of assets. In this regard, it is worth outlining that our methodology currently addresses only technical assets, while organizational and human-related assets have not been taken into account and are out of the scope of this paper, although we recognize that these may introduce additional un-predictable risks. In future developments of the SLA Generator, we are planning to extend the models in order to provide different rationales to identify applicable threats Ganin et al. (2017); Gisladottir et al. (2016), and also to extend the set of adopted tools, with more flexible solutions that are able to help developers to define and score the risks, based on multiple criteria and weights Ganin et al. (2017); Casola et al. (2009) and also based on specific application domains.

10. Conclusions

The wide adoption of modern software development methodologies, as Agile and DevOps, is dramatically reducing the time-to-market of many innovative services, mainly based on cloud technologies. In this context, security has never been considered as a primary requirement to take in consideration from the early stage of design and implementations. In this paper we proposed the adoption of a Security-by-Design methodology which relies on Security SLAs to model and assess security requirements. In particular, we have shown that the proposed methodology can be automated, thanks to the integration of some novel models and techniques. The proposed security models enabled us to reason over the security properties of a cloud application with a risk-based approach and, also, to take into account the complex dependencies among application components that may heavily affect the security of the application.

At this aim, we designed and implemented a comprehensive data model to collect security-related information (assets, threats, vulnerabilities,...) and a graph-based model to represent distributed cloud applications. Thanks to these, the proposed methodology has been almost completely automated and we developed a tool that supports (i) security requirement identification, performed by means of a risk analysis process, (ii) components (COTS) security assessments and (iii) cloud application security assessment.

The methodology was validated in the context of the MUSA European Project with two real world applications. The validation was performed by two independent teams, aiming at demonstrating the usability of the approach, its usefulness and, also, the time spent in the process to design and assess security in a cloud application.

The most relevant result was that, thanks to the introduced automation, teams without security skills were able to perform a security assessment in a very short time (few hours). The received feedback was very positive, as the developers were able to identify and formalize security requirements at a good level of detail (the security controls to be implemented) in the very early development stages, identifying possible security issues due to the distributed nature of the application.

In addition, the validation process outlined that the adoption of the proposed tool positively increased the awareness of the teams respect to security issues, and also motivated them to improve their knowledge and skills on the topic. As a conclusive remark, we wish to point out that security automation is needed but not fully addressable, it surely relies on a larger and larger diffusion of standard representations of security controls (security control frameworks), on a collection of open security data and catalogues (Threat Intelligence) and on the possibility to quantify security but the number of un-predictable threats and vulnerabilities always require flexible risk analysis techniques that take evaluators almost always in the loop. This could make it possible to fully negotiate and enforce security by means of Security SLAs, in a way similar to what is commonly done in different contexts for all service provisions.

We are already working to support, in the next future, multiple security standards (not only the NIST Control Framework SP-800-53) in order to simplify, and possibly automate, security certification processes like Common Criteria (ISO 15408 Common Criteria (2017)).

Moreover, as already said, in future development of the SLA Generator, we are planning to extend the models and the set of adopted tools, with flexible solutions that are able to help developers to define and score the risks, based on multiple criteria and wheights Ganin et al. (2017); Casola et al. (2009) and also based on specific application domains.

Acknowledgment

This research is partially supported by the project MUSA (Grant Agreement no. 644429) funded by the European Commission within call H2020-ICT-2014-1.

References

Aljawarneh, S.A., Alawneh, A., Jaradat, R., 2017. Cloud security engineering: Early stages of SDLC. Future Generation Comp. Syst. 74, 385–392.

Anderson, R., 2008. Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd.http://www.cl.cam.ac.uk/~rja14/book.html.

Azham, Z., Ghani, I., Ithnin, N., 2011. Security backlog in Scrum security practices. In: 2011 Malaysian Conference in Software Engineering. IEEE, pp. 414–417.

Benfenatki, H., Silva, C.F.D., Benharkat, A., Ghodous, P., Biennier, F., 2014. Methodology for semi-automatic development of cloud-based business applications. In: 2014 IEEE 7th International Conference on Cloud Computing, Anchorage, AK, USA, June 27 - July 2, 2014, pp. 954–955.

Berkeley DB Best practice. 2017. https://security.berkeley.edu/resources/best-practices-how-articles/system-application-security/database-hardening-best-practices.

Bousquet, A., Briffaut, J., Caron, E., Dominguez, E.M., Franco, J., Lefray, A., Lopez, O., Ros, S., Rouzaud-Cornabas, J., Toinard, C., Uriarte, M., 2015. Enforcing security and assurance properties in cloud environment. In: 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), pp. 271–280. doi:10.1109/UCC.2015.45.

Brangetto, P., et al., 2015. Economic aspects of national cyber security strategies, Project Report. https://ccdcoe.org/multimedia/economic-aspects-national-cyber-security-strategies.html.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J., 2004. Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems 8 (3), 203–236.

Casola, V., De Benedictis, A., Erascu, M., Modic, J., Rak, M., 2017. Automatically enforcing security SLAs in the cloud. IEEE Transactions on Services Computing 10 (5), 741–755.

Casola, V., De Benedictis, A., Modic, J., Rak, M., Villano, U., 2018. Per-service security SLAs for cloud security management: model and implementation. IJGUC 9 (2), 128–138.

Casola, V., De Benedictis, A., Rak, M., 2015. On the adoption of security SLAs in the cloud. Lecture Notes in Computer Science 8937, 45–62.

Casola, V., De Benedictis, A., Rak, M., Villano, U., 2017. A security metric catalogue for cloud applications. In: Barolli, L., Terzo, O. (Eds.), Complex, Intelligent, and Software Intensive Systems. Springer International Publishing, Cham, pp. 854–863.

Casola, V., De Benedictis, A., Rak, M., Villano, U., 2018. Security-by-design in multi-cloud applications: An optimization approach. Information Sciences 454-455, 344–362. doi:10.1016/j.ins.2018.04.081.

Casola, V., De Benedictis, A., Rak, M., Villano, U., 2018. Towards automated penetration testing for cloud applications. Proceedings - 27th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2018.

Casola, V., De Benedictis, A., Rak, M., Villano, U., Rios, E., Rego, A., Capone, G., 2019. Model-based deployment of secure multi-cloud applications. IJGUC 10 (6), 639–653. doi:10.1504/IJGUC.2019.102710.

Casola, V., Fasolino, A.R., Mazzocca, N., Tramontana, P., 2009. An AHP-based framework for quality and security evaluation. In: 2009 International Conference on Computational Science and Engineering, 3, pp. 405–411.

Catteddu, D., 2011. Security & Resilience in Governmental Clouds. Technical Report. CSA.

Cavoukian, A., Chanliau, M.D., 2013. Privacy and Security by Design: An Enterprise Architecture Approach. Ontario: Information and Privacy Commissioner Accessed January 27, 2020, https://www.ipc.on.ca/wp-content/uploads/Resources/pbd-privacy-and-security-by-design-oracle.pdf.

Chen, J., Zhu, Q., 2017. Security as a service for cloud-enabled internet of controlled things under advanced persistent threats: A contract design approach. IEEE Trans. Information Forensics and Security 12 (11), 2736–2750.

Cloud Security Alliance, 2011a. Consensus Assessment Initiative Questionnaire. https://cloudsecurityalliance.org/group/consensus-assessments/.

Cloud Security Alliance, 2011b. Security, Trust & Assurance Registry (STAR). https://cloudsecurityalliance.org/star/.

Cloud Security Alliance, 2019. Top Threats to Cloud Computing: Egregious Eleven. https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-egregious-eleven.

Collier, Z.A., DiMase, D., Walters, S., Tehranipoor, M.M., Lambert, J.H., Linkov, I., 2014. Cybersecurity standards: Managing risk and creating resilience. Computer 47 (9), 70–76. doi:10.1109/MC.2013.448.

Common Criteria, 2017. CCMB-2017-04-001: Common Criteria for Information Technology Security Evaluation v3.1 rev5. https://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R5.pdf.

CSA, 2015. Cloud controls matrix v3.0. https://cloudsecurityalliance.org/download/cloud-controls-matrix-v3/.

Cybersecurity and Digital Privacy (Unit H.1 Team), 2018. The Directive on security of network and information systems (NIS Directive). https://ec.europa.eu/digital-single-market/en/network-and-information-security-nis-directive.

Dekker, M., 2012. Critical Cloud Computing A CIIP perspective on cloud computing services. Technical Report. ENISA.

Dyck, A., Penners, R., Lichter, H., 2015. Towards Definitions for Release Engineering and DevOps. 2015 IEEE/ACM 3rd International Workshop on Release Engineering. 3–3.

D5.1 MUSA case studies work plan. 2018. https://www.musa-project.eu/documents2/d51-musa-case-studies-work-plan.

D5.5 Results of final evaluation of MUSA framework. 2018. https://www.musa-project.eu/documents2/d55-results-first-evaluation-musa-framework.

European Network and Information Security Agency (ENISA), 2012. Procure secure. a guide to monitoring of security service levels in cloud contracts.

Ganin, A.A., Quach, P., Panwar, M., Collier, Z.A., Jeffrey M. Keisler, D.M., Linkov, I., 2017. Multicriteria Decision Framework for Cybersecurity Risk Assessment and Management. Risk Analysis 1–17.

Geer, D., 2010. Are companies actually using secure development life cycles? Computer 43 (6), 12–16.

Ghani, I., Arbain, A.F.B., Oueslati, H., Rahman, M.M., ben Othmane, L., 2016. Evaluation of the challenges of developing secure software using the agile approach. Int. J. Secur. Softw. Eng. 7 (1), 17–37. doi:10.4018/IJSSE.2016010102.

Ghani, I., Yasin, I., 2013. Software Security Engineering In Extreme Programming Methodology: A Systematic Literature. Science International 25 (2), 215–221.

Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N., 2004. Requirements Engineering meets Trust Management: Model, Methodology, and Reasoning. In: Trust Management, pp. 176–190.

Gisladottir, V., Ganin, A.A., Keisler, J.M., Kepner, J., Linkov, I., 2016. Resilience of cyber systems with over- and underregulation. Risk Analysis 37 (9), 1644–1651. doi:10.1111/risa.12729.

Hong, J.B., Nhlabatsi, A., Kim, D.S., Hussein, A., Fetais, N., Khan, K.M., 2019. Systematic identification of threats in the cloud: A survey. Computer Networks 150, 46–69.

International Organization for Standardization, 2013. ISO/IEC 27002:2013: Information technology Security techniques Code of practice for information security controls.

International Standard Organization, 2018. ISO/IEC 27000:2018 Information technology Security techniques Information security management systems Overview and vocabulary.

Internet Engineering Task Force (IETF), 2013. RFC6819:OAuth 2.0 Threat Model and Security Considerations. https://tools.ietf.org/html/rfc6819.

Jayaram, K., Mathur, A.P., 2005. Software engineering for secure software-state of the art: A survey. Purdue University.

Joint Task Force Transformation Initiative I, 2013. NIST SP 800-53 Rev 4: Recommended Security and Privacy Controls for Federal Information Systems and Organizations. Technical Report.

Khan, N.F., Ikram, N., 2016. Security requirements engineering: A systematic mapping (2010-2015). In: 2016 International Conference on Software Security and Assurance (ICSSA), pp. 31–36. doi:10.1109/ICSSA.2016.13.

Koch, R., 2012. The Role of COTS Products for High Security Systems. 2012 4th International Conference on Cyber Conflicts 413–426.

Massacci, F., Prest, M., Zannone, N., 2005. Using a security requirements engineering methodology in practice : The compliance with the Italian data protection legislation 27, 445–455.

Mateski, M., Trevino, C., Veitch, C., Michalski, J., Harris, J., Maruoka, S., Frye, J., 2012. Cyber threat metrics. Technical report SAND2012-2427, Sandia National Laboratories. https://fas.org/irp/eprint/metrics.pdf.

Microsoft Corporation, 2016. The STRIDE Threat Model https://docs.microsoft.com/en-us/previous-versions/1680commerce-server/ee823878(v=cs.20).

Mitre Corporation, 2018a. Common Vulnerabilities and Exposures (CVE) System web site. https://cve.mitre.org/.

Mitre Corporation, 2018b. The Common Weakness Enumeration (CWE) web site. https://cwe.mitre.org.

Mohan, V., Othmane, L.B., 2016. SecDevOps: Is It a Marketing Buzzword? - Mapping Research on Security in DevOps. In: 2016 11th International Conference on Availability, Reliability and Security (ARES). IEEE, pp. 542–547.

Mougouei, D., Sani, N.F.M., Almasi, M.M., 2013. S-Scrum : a Secure Methodology for Agile Development of Web Services. World of Computer Science and Information Technology Journal (WSCIT) 3 (1), 15–19.

Mouratidis, H., Giorgini, P., 2007. Security Attack Testing (SAT) testing the security of information systems at design time $ 32, 1166–1183.

Mouratidis, H., Giorgini, P., Manson, G., 2005. When security meets software engineering: a case of modelling secure information systems $ 30, 609–629.

MUSA Consortium, 2016. MUSA project web site http://www.musa-project.eu.

MUSA project, 2017. D5.5 Results of Final evaluation of MUSA framework. https://musa-project.eu/documents2/d55-results-first-evaluation-musa-framework.

OASIS, 2013. Topology and Orchestration Specification for Cloud Applications Version 1.0. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.

OWASP, 2016a. The OWASP Risk Rating Methodology Wiki Page https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

OWASP, 2016b. The OWASP Top 10 2013. Project Home Page https://www.owasp.org/index.php/OWASP_Risk_Rating_1685Methodology.

OWASP, 2019. Application Security Verification Standard 4.0 https://github.com/OWASP/ASVS/raw/master/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0-en.pdf.

PaaSage Consortium, 2016. The CAMEL web site. http://camel-dsl.org/.

Pfleeger, S., Cunningham, R., 2010. Why measuring security is hard. IEEE Security Privacy 8 (4), 46–54. doi:10.1109/MSP.2010.60.

Pohl, C., Hof, H.-J., 2015. Secure Scrum: Development of Secure Software with Scrum (c), 15–20. arXiv:1507.02992.

Rak, M., 2017. Security assurance of (multi-)cloud application with security SLA composition. Lecture Notes in Computer Science 10232, 786–799.

Rak, M., Ficco, M., Battista, E., Casola, V., Mazzocca, N., 2014. Developing secure cloud applications. Scalable Computing: Practice and Experience 15 (1), 49–62.

Rak, M., Suri, N., Luna, J., Petcu, D., Casola, V., Villano, U., 2013. Security as a service using an SLA-based approach via SPECS. In: Proc. of CloudCom, 2013 IEEE 5th Int. Conf. on, 2, pp. 1–6. doi:10.1109/CloudCom.2013.165.

Rios, E., Iturbe, E., Orue-Echevarria, L., Rak, M., Casola, V., 2015. Towards self-protective multi-cloud applications: MUSA-a holistic framework to support the security-intelligent lifecycle management of multi-cloud applications. In: CLOSER 2015 - 5th International Conference on Cloud Computing and Services Science, Proceedings, pp. 551–558.

Rios, E., Rak, M., Iturbe, E., Mallouli, W., 2017. SLA-Based Continuous Security Assurance in Multi-Cloud DevOps. In: Proceedings of the International Workshop on Secure Software Engineering in DevOps and Agile Development co-located with the 22nd European Symposium on Research in Computer Security (ESORICS 2017), Oslo, Norway, September 14, 2017., pp. 50–68.

Ross, R., McEvilley, M., Oren, J. C., 2016. NIST SP 800-160: Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-160v1.pdf.

Singh, S., Jeong, Y., Park, J.H., 2016. A survey on cloud computing security: Issues, threats, and solutions. J. Network and Computer Applications 75, 200–222.

SPECS project web site 2016 http://www.specs-project.eu.

SINTEF, 2016. The CloudML Wiki web site. https://github.com/SINTEF-9012/cloudml/wiki.

SLA-Ready Consortium, 2019. The SLA-Ready project web site http://www.sla-ready.eu.

SLALOM Consortium, 2019. The SLALOM project web site http://www.slalom-project.eu.

SSL Labs, 2018. SSL Threat Model. https://www.ssllabs.com/projects/ssl-threat-model/.

The EU General Data Protection Regulation. 2018. https://www.eugdpr.org/.

The Software Assurance Forum for Excellence in Code (SAFECode), 2018. Fundamental Practices for Secure Software Development Essential Elements of a Secure Development Lifecycle Program - Third Edition.

Thillaiarasu, N., ChenthurPandian, S., 2016. Enforcing security and privacy over multi-cloud framework using assessment techniques. In: 2016 10th International Conference on Intelligent Systems and Control (ISCO), pp. 1–5. doi:10.1109/ISCO.2016.7727001.

Ur Rahman, A.A., Williams, L., 2016. Software security in DevOps. In: Proceedings of the International Workshop on Continuous Software Evolution and Delivery - CSED '16. ACM Press, New York, New York, USA, pp. 70–76.

Wäyrynen, J., Bodén, M., Boström, G., 2004. Security Engineering and eXtreme Programming: An Impossible Marriage? In: Extreme Programming and Agile Methods - XP/Agile Universe 2004, pp. 117–128.