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

Valentina Casola*, Alessandra De Benedictis*, Massimiliano Rakb, 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

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 life-cycle in an almost-completely automated way. It relies upon a completely automated security assessment process, which enables 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.

*Valentina Casola
Email address: casolav@unina.it (Valentina Casola)
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” [1]. The situation is going to change, particularly in Europe, due the introduction of the General Data Protection Regulation (GDPR [2]) and to the application of the Directive on security of network and information systems (NIS Directive) [3], 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. 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 [4].

Second, addressing security in clouds is an open issue by itself. The cloud computing paradigm 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. To cope with this issue, a recent report by
the European Union Agency for Network and Information Security (ENISA)
[5] 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, however, that Security SLAs are not yet widely adopted by
CSPs, especially in public clouds [6]. Furthermore, as outlined in [7, 8], CSPs
mainly offer SLAs that guarantee the same service terms 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.
A further factor is linked to the software development approaches most com-
monly used today, namely DevOps [9] and Agile. They dramatically reduce the
time-to-market of developed software but, at the same time, can be hardly in-
tegrated with security design and assessment processes [10, 11]. In fact, both
Agile methodologies and DevOps processes imply a high number of continu-
ous development iterations, typically carried out by personnel with low secu-
rity skills, and DevOps in particular requires high automation capabilities to
be effective. 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 that can be hardly automated. The final result is a
dramatically high number of hours of work spent monitoring the security pro-
vided 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 [12]. It refers to a holistic, anticipatory approach to security (in con-
trast to the traditional security-by-obscurity principle) based on the adoption
of the secure-by-default paradigm in the configuration of both software compo-
nents 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 and 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 [13], 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 COTS or even by insiders [14, 15]. 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 section 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 [13].

In light of the above considerations, this paper aims at proposing a novel security SLA-based Security-by-Design methodology for the development of secure cloud applications that:

- can be integrated within Agile and DevOps development processes,
- is able to support all the phases of the risk management life-cycle in an almost-completely automated way, and
• includes a completely automated security assessment process, which enables to assess the security properties granted by a cloud application and to report them in a security SLA.

The methodology includes a risk analysis activity, relying upon well-known tools and techniques, which is aimed at identifying the security controls and mechanisms that meet the application security requirements and that can be conducted even by developers provided with basic security skills, as discussed in Section 5.

All the phases of the proposed methodology strongly rely on a set of models and tools, which enable to automate most part of involved activities. In particular, as illustrated in Section 4, we have introduced a reference security model to cope with the threats that apply to cloud application assets, along with the countermeasures (i.e., the security controls) needed to thwart these threats and the information needed to verify and assess the countermeasures. We also adopted a security SLA model [16] to represent security controls and metrics and to quantify security by means of the adoption of security-oriented Service Level Objectives (SLOs). Furthermore, we used the MACM model [17] to represent application components and security features.

In order to validate the proposed methodology, we developed a tool that supports the security requirements identification, performed by means of a risk analysis process, and the cloud application security assessment, devoted to verifying the correct implementation and configuration of security controls given all the cloud deployment constraints. We integrated it in the DevOps cycle proposed by the MUSA project [18], recently closed and focused on the design of secure multi-cloud applications. In particular, we involved two evaluators’ teams, with minimum security skills, to prove the methodology and related tools, and collected their feedbacks on the efficiency and usability of the methodology. As illustrated in the paper, the results are very promising, since the tool provides the end user with a lot of information on threats, security controls and metrics, thus drawing the attention to many security aspects for application designers.
and developers. We also evaluated the time spent in the design and assessment phases. As shown in the Validation section, the results depend on the complexity of the application but the tools were very helpful to both teams.

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. In Section 3, the proposed Security-by-Design methodology is introduced, and its integration within common Agile software development methodologies is discussed. In Sections from 4 to 7, we show an innovative security modeling formalism based on Security SLAs, and 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 8, 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 9 some conclusions and future work are presented.

2. Related Work

In this section, we will discuss interesting works related to our contribution, which basically involves the approaches aimed at integrating security aspects within software development methodologies and processes, and the frameworks and solutions devoted to risk analysis and security assessment tasks within such processes.

2.1. Security in software development methodologies

The consideration of security aspects in the design of systems is the focus of so-called security engineering practices [19], 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 [20] 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 [21, 22]. 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. [23] proposed an extension to the Tropos agent-oriented software engineering methodology presented in [24]. 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 [25, 26, 27]. 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 process timeline. 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.

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 [12], which is an approach to security engineering conceived in contrast to the traditional security-by-obscurity principles. 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 pro-
cesses 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.

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. As a matter of fact, several challenges exist related to the development of secure software using agile methods, mainly 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 [28, 29, 30, 31]. Nevertheless, some proposals exist aimed at integrating security management into the most popular agile methodologies, such as Scrum. In this regard, Azham et al. proposed in [32] an extension to the Scrum methodology relying upon security backlogs, to be used during development, analysis and implementation in Scrum phases. Mougouei et al. introduced a different extension to Scrum, namely S-Scrum [33], 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 [31], 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 than 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 [9], 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 [11]. Recent studies outline that the DevOps automation tools, if correctly used, may help to correctly address security assessment procedures [10, 11], leading to the introduction of more security-oriented processes, named SecDevOps or DevSecOps [10]. However, as highlighted by cited papers, suitable methodologies able to address automated security assessment processes are still lacking.

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.

2.2. 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 [15, 34, 14], 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.
[35] 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 ap-
proaches 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. [14] 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 de-
cision process, useful to assess the overall utility of cybersecurity management
alternatives.

While several tools and techniques exist that focus on the security analy-
sis 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 Cun-
ningham [36] 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 vulnera-
bilities of interest for a system, use well-known processes for risk analysis and
assessment, and adopt a novel technique for automated security assessment.


In order to cope with the challenges outlined in the analysis of the state of the art, we propose an innovative Security SLA-based Security-by-Design (SSDE) process, able to meet 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. suggest the adoption of additional security mechanisms 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.

The SSDE process 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.

As shown in Figure 1, the process involves three main steps, namely Modeling, Per-Component Security Assessment and Per-Application Security assessment.

In the Modeling phase, a cloud application is represented as a collection of cooperating software components that can be mapped, one-to-one, to cloud resources (i.e., infrastructures, platforms or software services). The components may be either (custom or COTS) software artifacts that must be deployed onto a virtual machine to be executed, or software services offered by a cloud provider and whose functionalities can be simply invoked to accomplish a specific task. The model built in this phase describes the application architecture in terms
Figure 1: The SLA-based Security-by-Design Flow

of its components and their interconnections, also providing the deployment layout, namely the component-cloud resource mapping. As will be discussed in detail in Section 4, this phase relies upon a graph-based formalism, named MACM (Multi-cloud Application Composition Model [37]) that allows to easily specify not only the architectural information, but also the aspects related to security.

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 and countermeasures 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. It 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 4.2. This document is referred to as an **SLA Template - SLAT** (i.e., the *Requirement Component 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 5, in the proposed process these tasks are partially automated, thanks to the adoption of a comprehensive data model that 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 [37] and summarized in Section 7.2. 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. In the following subsection, we will illustrate how it can be implemented in an Agile development process, while in the last section, dedicated to Validation (Section 8), we will illustrate its integration into the MUSA DevOps workflow.

3.1. The SSDE process integrated in Scrum

In this section, we discuss the integration of the SSDE process in the Scrum agile software development methodology, which today is probably the most popular approach adopted to manage the development of software products in an iterative, incremental and flexible way.

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.

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 we aim at identifying a set of tasks and actions that
can be integrated in the traditional Scrum workflow without altering the Scrum principles, so to address security issues together with all the other functionalities and requirements of the system. It is worth noting that the proposed SSDE process is made of several specific tasks that can be isolated and considered as single actions to plan in the Sprint Planning phase. Since there are some logical dependencies among these tasks, the right assignment of the priorities, which drives the selection of the subsequent Sprints to execute, is a key factor. The Modeling must always be considered as a high-priority action, as well as the risk analysis sub-task of the Per-Component Security Assessment step, while the security assessment sub-task can be assigned a low priority and can be performed during the other development activities. The Per-Application Security Assessment, instead, should be considered as a high-priority action to be performed in the very early Sprints. In particular, we suggest to initially assume that the Requirement Component SLATs produced by the risk analysis task are correctly implemented by each component internally, in order to move up the application assessment and identify easily the possible criticalities related to deployment and to the application architecture. Moreover, we suggest that the Per-Application Security Assessment analysis takes place during the Sprint Review phase, in order to evaluate the model and organization of the software adopted and to influence the following Sprint Planning activities. After the first Sprint, Per-Application Security Assessment will be useful to assign priorities to the next Sprint Backlogs during planning events.

4. Modeling

The Modeling phase is devoted to building a 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 as
illustrated in the next subsections.

In particular, we developed a reference security model (described in Section 4.1) by 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 4.2) proposed by the SPECS project, it extends the Web Service Agreement standard (WS-Agreement[38]) and enables to describe, in a structured way, what 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 introduced a graph-based formalism named Multi-cloud Application Composition Model (MACM), described in Section 4.3. As discussed later in the paper, this formalism not only allows to describe the architecture of a cloud application, but it 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.

4.1. The reference Security Model

The proposed security model is reported in Figure 2, it puts together and extends different concepts introduced by several security standards and initiatives. 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 [39]. 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 weak-
ness, 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 [40] and the Common Weakness Enumeration (CWE) project [41]: 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 SSDE 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 data model, named Threat Catalogue and available as open source at the address http://bitbucket.org/cerict/sla-model,
which collects several information related to threats and other aspects. The catalogue is organized per-component type, i.e. for the specific software, protocol or technology to protect. In particular, threats in the catalogue:

- are mapped to the set of software component types (e.g., web applications, storage services, monitoring agents etc.) to which they may potentially apply (the assets);

- are classified based on the STRIDE threat categories [42] (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 derived from the CWE and other sources such as existing security guidelines and best practices;

In the current implementation it includes almost 100 well-known threats against different component types, mainly gathered from standards and open repositories (e.g., the OWASP top 10 Threats [43], the OAUTH Threat Model [44], the SSL Threat Model from SSLabs [45], the CSA Top Threats [46]) and from scientific papers.

In addition to threat-related information, as discussed in the next sections, the catalogue includes further information related to the enforcement and monitoring of countermeasures, which is used to automate the identification of security controls and construction of Security SLAs. In the practice threats will be mapped to the set of countermeasures, in terms of standard security controls (the NIST Security Control Framework is currently supported [13]), which should be adopted in order to prevent their exploitation.
4.2. The Security SLA model

The Security SLA model adopted in the proposed process has been introduced in the SPECS project and has been widely discussed in [47]. The model is based on the WS-Agreement standard, which has been extended to include cloud provider-specific information and security-related guarantees. In a Security SLA, the security policies adopted by a provider (possibly related to a specific service) are expressed in terms of standard security controls, defined 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 [13]. Security controls address 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 for example access control (AC), identification and authentication (IA), physical and environmental protection (PE) and awareness and training (AT).

Security guarantees are expressed by means of security Service Level Objectives (SLOs), which define the desired level of security in terms of conditions involving suitable security metrics. Each metric is associated with a specific security control and helps verify if the control has been correctly implemented and at which level (e.g., a control requiring the set-up of an authentication mechanism may be implemented at different levels, ranging from the simple user-password mechanism to much more complex two-factor authentication mechanisms).

The Threat Catalogue data model, previously introduced, also integrates a catalogue of security metrics, which has been introduced in the context of SPECS and enhanced by MUSA [48], and that collects security metrics taken from multiple sources including European projects, international standards and initiatives and scientific papers. Metrics in the catalogue are classified based on the type of component they apply to and are mapped to the respective security controls.
4.3. Modeling cloud applications with MACM

The MACM formalism allows 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. It is worth noting that several languages and tools exist for modeling cloud applications, such as CAMEL [49], CloudML [50] or Tosca [51], 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.

Application components are represented as graph nodes belonging to the IaaS, PaaS or SaaS cloud service 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. Moreover, any IaaS or PaaS node may host other PaaS or SaaS nodes, while any of the cloud service type nodes may use an SaaS node. MACM allows to represent security properties of components and providers in terms of both SLAs (SLA node type) and SLA Templates (SLAT node type), as defined in Section III. In particular, any CSP or cloud service type may support an SLAT and/or grant an SLA. 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 web applications, storage services, etc.) or to the protocol used for communication, to deployment configuration information, to generic constraints, etc.

Figure 3 shows the MACM of an example cloud application (referred to as the “messaging cloud application” hereafter) that allows users to upload and share personal documents through a web interface and to talk with one another thanks to a live chat service. The application is made of three components: (i) the application core \( \mathcal{W} \), which is the web application that offers the upload and visualization functionalities to the users, (ii) the application database \( \mathbb{DB} \), which is the database used by the application core to store the users’ data, and
(iii) the application chat service $S$, which is a live chat service, offered in SaaS mode, used by the application core to provide its users with instant messaging functionalities.

As shown in the figure, $W$ *uses* both $DB$ and $S$, and a single 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.

Besides application architecture, the MACM formalism allows also to represent a Security SLA document in compliance with the model discussed in the previous section. As an example, Figure 4 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 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.

The MACM model can be easily managed by using graph-based databases, which enable to easily maintain and query complex graphs using graph-oriented languages (like Cypher). In Section 7.1, 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 7.2.

5. Per-Component Security Assessment - Security Control Identification

As already outlined in Section 3, 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 the next section, 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 Figure 5, 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.

![Figure 5: Risk Analysis Process to build the Requirement SLA Template](image)

5.1. 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 an extended Threat Catalogue, where the threats are stored, together with their description, and associated to: (i) the asset 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 (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.
Such mappings, introduced in Section 4.1, are represented in Figure 6, which shows a simplified view of the Threat Catalogue data model used in this task.

![Figure 6: Threat selection data model](image)

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. 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 1, 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
<table>
<thead>
<tr>
<th>Threat</th>
<th>STRIDE cat.</th>
<th>Condition</th>
<th>Countermeasures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account Hijacking</td>
<td>SPOOFING</td>
<td>Does the component maintain account information?</td>
<td>AC-2, AC-2(12), AC-3, AC-7, AC-9, AC-17, AC-23, AC-9(2)</td>
</tr>
<tr>
<td>Cross-Site Request Forgery (CSRF)</td>
<td>TAMPERING</td>
<td>Does the component accept requests without checking their origin and trustworthiness?</td>
<td>SI-4, SI-10</td>
</tr>
<tr>
<td>Cross-Site Scripting (XSS)</td>
<td>TAMPERING</td>
<td>Are the inputs from users (forms, http requests) directly used without validation?</td>
<td>SI-10, RA-5</td>
</tr>
<tr>
<td>Man in the Middle attack</td>
<td>SPOOFING</td>
<td>Does the component communicate with other components without ensuring the authenticity and integrity of the communications?</td>
<td>SC-23, IA-2(13)</td>
</tr>
<tr>
<td>Buffer overflow exploitation</td>
<td>INFORMATION DISCLOSURE, DOS, TAMPERING</td>
<td>Does the component allow to write on memory without restrictions?</td>
<td>SI-16</td>
</tr>
<tr>
<td>Missing Function Level Access Control</td>
<td>ELEVATION OF PRIVILEGES</td>
<td>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?</td>
<td>AC-6, AC-6(1), AC-3</td>
</tr>
</tbody>
</table>

Table 1: Extract of the threats selected for component W of the example messaging application
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).

5.2. 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 [52], 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 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.

The automation process relies on a simple consideration: 12 of the 16 parameters are independent of the system owner, but strictly depend on the characteristics of the threat itself. As a consequence, each threat in the Catalogue is pre-labeled by security experts with default values for likelihood parameters and technical impact parameters. The last 4 values, namely Financial damage, Reputation damage, Non-compliance and Privacy violation, which measure the business impact of the threats, will be set by the evaluator, that is the only one able to correctly provide them.
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.

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”. Figure 7 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.

![Figure 7: The OWASP Risk Rating Methodology applied to the example messaging application](image)

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.
5.3. 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 4.1, the Threat Catalogue reports the association between threats and all possible countermeasures. This allows to automatically retrieve the full list of security controls 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 control 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).

Figure 8 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.

Figure 8: Security control selection data model

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.

5.4. 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 4.2, 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 2 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).

5.5. 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.
<table>
<thead>
<tr>
<th>Control</th>
<th>Metric</th>
<th>Description</th>
<th>SLO</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC-2</td>
<td>AUTHENTICATION_LEVEL</td>
<td>Level of authentication mechanism in place: 0: no authentication, 1: Login/Password, 2: 2 factor auth, 3: white-list of IPs + credentials, 4: combination of mechanisms (IP, port number, certificate or public keys, credentials etc.)</td>
<td>&gt; 0</td>
</tr>
<tr>
<td></td>
<td>SHARED_ACCOUNTS_PERC</td>
<td>Percentage of users with access to shared accounts</td>
<td>≤ 50</td>
</tr>
<tr>
<td>AC-3</td>
<td>ACC_CONTR_CORRECT</td>
<td>Total access control rules fulfillment in place</td>
<td>= yes</td>
</tr>
<tr>
<td>AC-6</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>AC-6(1)</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>AC-7</td>
<td>ACC_CONTR_LOGGING</td>
<td>Logging of all tentative of access in place</td>
<td>= yes</td>
</tr>
<tr>
<td></td>
<td>USR_AUTH_BEHAV_CHG</td>
<td>More than one station or browser used for connection by the user agent</td>
<td>= yes</td>
</tr>
<tr>
<td>AC-17</td>
<td>REMOTE_ACCESSSES_PERC</td>
<td>Percentage of remote access points used to gain unauthorized access</td>
<td>≤ 10</td>
</tr>
<tr>
<td></td>
<td>SRC_REQ_LOCATION</td>
<td>List of countries allowed to make requests</td>
<td>=</td>
</tr>
</tbody>
</table>

Table 2: Metrics and SLOs selected for component W of the example messaging application
6. Per-Component Security Assessment - Security Assessment Process

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 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 techniques, like penetration tests, which need specific expertise, are time-costly and can hardly be automated.

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 questionnaire 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 [53];
- Berkley DB Best Practices [54];
- CAIQ by Cloud Security Alliance (CSA) [55].
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.

As an example, let’s assume that an SSDE user aims at assessing the web application W for the security control

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. Per-Application Security Assessment

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 7.1) 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 7.2, 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.

### 7.1. 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:
  
  ```cypher
  $match (n:IaaS:service) return (n)
  $$
  ```

- list all the components hosted by the vm with id equal to '23':
  
  ```cypher
  $match (vm {id:'23'}), (n:service), (vm)-[:hosts]->(n) return (n)
  $$
  ```

- retrieve the SLAT supported by a node S:
  
  ```cypher
  $match (n:service {id:'S'}), (n)-[supports]->(slat) return (slat)
  $$
  ```

- retrieve the SLA granted by a node S:
  
  ```cypher
  $match (n:service {id:'S'}), (n)-[grants]->(sla) return (sla)
  $$
  ```

- list all the controls included in the SLAT associated with node 'S':
  
  ```cypher
  $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:

\[ SLA(c_j, \text{app}) = \bigwedge_{N_i \in N} SLA(c_j, N_i) \]  

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.

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 Figure 3, 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:

\[ SLA(AC - 3, DB) = SLAT(AC - 3, DB) \land SLA(AC - 3, VM) \]  

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:

\[
SLA(AC-3, W) = SLAT(AC-3, W) \land SLA(AC-3, VM) \land SLA(AC-3, DB) 
\] (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.

7.2. 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 3 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 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 \texttt{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 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 3, already presented. The starting template string is \(\text{slat}(\text{AC-3},<S>) \text{ AND } <X>\). In this example \(<S>\) is the node under analysis VM, and so the resulting initial string is \(\text{slat}(\text{AC-3},\text{VM}) \text{ AND } <X>\). As shown in Figure 3, the node VM is involved in the relationships \(\text{provides}\) with the role of \(\text{target}\) (having as \(\text{source}\) the CSP), and in three relationships \(\text{hosts}\) with the role of \(\text{source}\) (the corresponding targets are W, DB and S). Accordingly, the composition rule will be modified as follows:

- **start** the starting string is: \(\text{slat}(\text{AC-3},\text{VM}) \text{ AND } <X>\).

- **step 1** the \(\text{provides}\) relationship with role “target” modifies the rule into: \(\text{slat}(\text{AC-3},\text{VM}) \text{ AND } \text{sla}(\text{AC-3},\text{CSP}) \text{ AND } <X>\).

- **steps 2,3 and 4** the \(\text{hosts}\) relationship with role “sources” leaves the

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Role</th>
<th>function</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>-</td>
<td>slat(sc,&lt;S&gt;) AND &lt;X&gt;</td>
</tr>
<tr>
<td>provides</td>
<td>source</td>
<td>1</td>
</tr>
<tr>
<td>provides</td>
<td>target</td>
<td>sla(sc,&lt;S&gt;) AND &lt;X&gt;</td>
</tr>
<tr>
<td>hosts</td>
<td>source</td>
<td>&lt;X&gt;</td>
</tr>
<tr>
<td>hosts</td>
<td>target</td>
<td>sla(sc,&lt;S&gt;) AND &lt;X&gt;</td>
</tr>
<tr>
<td>uses</td>
<td>source</td>
<td>sla(sc,&lt;T&gt;) AND &lt;X&gt;</td>
</tr>
<tr>
<td>uses</td>
<td>target</td>
<td>&lt;X&gt;</td>
</tr>
</tbody>
</table>

Table 3: Composition rules for the AC control family
string <X> unmodified: \texttt{s-lat(AC-3,VM) AND s-la(AC-3,CSP) AND <X>}. 

- \textbf{end} all the relationships have been taken in consideration; at the end of the process, the final rule is: \texttt{s-lat(AC-3,VM) AND s-la(AC-3,CSP)}. 

The composition function obtained (\texttt{s-lat(AC-3,VM) AND s-la(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.

8. Validation

In order to validate the approach, we developed the SLA Generator Tool\footnote{Available at: https://musa-project.eu/content/service-level-agreement-support} to support all the phases of the process. The validation was carried out by testing the SLA Generator usage with two MUSA projects’ case studies (real-world applications), and we evaluated the economic benefits of the methodology in terms of its efficiency and usability and 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, and then we will illustrate the validation methodology and the obtained results. Finally, we will provide a conclusive discussion of the benefits introduced by our proposal and of its main limitations.

8.1. 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: \textit{Modeling}, \textit{Risk Assessment}, \textit{Cloud Service Selection and Decision Support}, \textit{SLA Generation}, \textit{SLA


Composition, Deployment Planning, Deployment and Execution. The details of the full solution are available on the project website and in [56]. Figure 9 shows the overall MUSA workflow and outlines the mapping with the SSDE phases.

![MUSA DevOps Workflow and mapping with the methodology](image)

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 trans-
lated into the Requirement Component SLATs and assessed according to the methodology presented in Section 6. The SSDE Per-Application Security Assessment phase is explicitly mapped onto the MUSA SLA Composition phase, by applying the techniques illustrated in Section 7.2.

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.

8.2. Validation Methodology

The SLA Generator was evaluated with respect to different quality metrics that include: Availability, Efficiency, Usability, Interoperability, Reusability, Testability and Time consumed. In the following discussion, we will focus on Efficiency, Usability and Time consumed, in order to highlight how the process can be easily integrated into existing development methodologies with a minimum effort, is compliance with business and economic requirements.

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.

8.2.1. Composition of the Evaluators Team

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.

Figure 10: Evaluation Teams composition

Two different evaluator teams were involved in the evaluation process. Figure 10 briefly summarizes the composition of the teams. 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.

8.2.2. 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 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:

$$Fulfillment\; degree = 100 \times \frac{(w_5 \times P_5 + w_4 \times P_4 + w_3 \times P_3 + w_2 \times P_2 + w_1 \times P_1)}{(n \times 5)}$$

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.

8.3. Validation Results

Figures 11 and 12 report the questions and answers we collected respect to the Efficiency and Usability metrics, respectively. Table 4 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 4), 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.

As shown in Figure 11, 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's 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.

![Figure 11: SSDE process and SLA Generator Tool Efficiency](image)

Usability-related questions, reported in Figure 12, 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 [14]. 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 4 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
<table>
<thead>
<tr>
<th>Time spent to:</th>
<th>Case Study</th>
<th>Avg</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a component SLA template</td>
<td>CS1</td>
<td>100 mins</td>
<td>30 mins</td>
</tr>
<tr>
<td>Create a component SLA template</td>
<td>CS2</td>
<td>32 mins</td>
<td>20 mins</td>
</tr>
<tr>
<td>Perform a self-assessment of a single component</td>
<td>CS1</td>
<td>78 mins</td>
<td>60 mins</td>
</tr>
<tr>
<td>Perform a self-assessment of a single component</td>
<td>CS2</td>
<td>24 mins</td>
<td>10 mins</td>
</tr>
<tr>
<td>Perform self-assessment of all components</td>
<td>CS1</td>
<td>326 mins</td>
<td>240 mins</td>
</tr>
<tr>
<td>Perform self-assessment of all components</td>
<td>CS2</td>
<td>154 mins</td>
<td>77 mins</td>
</tr>
<tr>
<td>Create a composite SLA</td>
<td>CS1</td>
<td>380 mins</td>
<td>60 mins</td>
</tr>
<tr>
<td>Create a component SLA</td>
<td>CS2</td>
<td>16 mins</td>
<td>10 mins</td>
</tr>
</tbody>
</table>

Table 4: Time spent on security assessment phases

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 [57].

8.4. 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 over-
all 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 devel-
opers, especially those provided with limited security skills, to identify, as soon
as possible, the main threats and vulnerabilities affecting the application un-
der 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 com-
pleteness 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 [14, 34], 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 [14, 58] and
also based on specific application domains.

9. 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 [22]).

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 weights [14, 58] 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


   URL http://www.musa-project.eu


URL http://dx.doi.org/10.4018/IJSSE.2016010102


[54] Berkeley DB Best practice, https://security.berkeley.edu/resources/best-practices-how-articles/


