"Automated Software Engineering Journal" manuscript No.
AUSE-D-12-00365

Adaptive, Model-driven Security Engineering for SaaS Cloud-based Applications

Mohamed Almorsy · John Grundy · Amani S. Ibrahim

Received: 9 Jul 2012 / Revised 16 Jan 2013 / Revised 12 Jul 2013 / Accepted: 12 Aug 2013

Abstract Software-as-a-service (SaaS) multi-tenancy in cloud-based applications helps service providers to save cost, improve resource utilization, and reduce service customization and maintenance time. This is achieved by sharing of resources and service instances among multiple "tenants" of the cloud-hosted application. However, supporting multi-tenancy adds more complexity to SaaS application's required capabilities. Security is one of these key requirements that must be addressed when engineering multi-tenant SaaS applications. The sharing of resources among tenants i.e. multi-tenancy increases tenants' concerns about the security of their cloud-hosted assets. Compounding this, existing traditional security engineering approaches do not fit well with the multi-tenancy application model where tenants and their security requirements often emerge after the applications and services were first developed. The resultant applications do not usually support diverse security capabilities based on different tenants' needs, some of which may change at run-time i.e. after cloud application deployment. We introduce a novel model-driven security engineering approach for multi-tenant, cloud-hosted SaaS applications. Our approach is based on externalizing security from the underlying SaaS application, allowing both application and security to evolve at runtime. Multiple security sets can be enforced on the same application instance based on different tenants' security requirements. We use abstract models to capture service provider and multiple tenants' security requirements and then generate security integration and configurations at runtime. We use dependency injection and dynamic weaving via Aspect-Oriented Programming (AOP) to integrate security within critical application entities at runtime. We explain our approach, architecture and implementation details, discuss a usage exam-

Mohamed Almorsy, John Grundy, and Amani S. Ibrahim
Centre for Computing & Engineering Software Systems,
Swinburne University of Technology, Melbourne, Australia
E-mail: malmorsy@swin.edu.au, jgrundy.swin.edu.au, aibrahim.swin.edu.au

ple, and present an evaluation of our approach on a set of open source web applications.

Keywords Software-as-a-Service · Model-driven Engineering · Adaptive-Security · Security Engineering · Tenant-Oriented Security

1 Introduction

Software-as-a-service (SaaS) is one of the key service delivery models introduced by the cloud computing model [3]. SaaS simplifies the software procurement process to renting services instead of buying them and their underlying infrastructure. Thus tenants can save infrastructure cost, software license, system administration, and development staff. Tenants can also switch to different service providers. The SaaS model helps service providers to target the small and medium enterprise markets by offering them a reasonable software adoption cost model.

Multi-tenancy is a new SaaS architecture pattern where service tenants share a single service instance. Multi-tenancy helps SaaS service providers to focus on operating, customizing, maintaining, and upgrading a single instance. On the other hand, multi-tenancy increases service tenants' concerns about the security of their outsourced cloud-hosted assets that are shared with other tenants and who may be either competitors or malicious users. Supporting multi-tenancy also adds more requirements on service providers as they have to develop or reengineer their systems to support multi-tenancy and to ensure tenants' data isolation. In addition, actual SaaS application tenants often become known only after applications have been delivered. Tenants usually have different security requirements that emerge at runtime based on their current business objectives and security risks.

Existing, traditional design-time security engineering approaches, such as KAOS [17], UMLsec [15], secureUML [18], focus on how to identify, capture, and model system security objectives and requirements that need to be enforced in the software under development. These approaches focus on mapping security requirements identified in the early-stage of security requirements engineering on system design entities (components, classes, methods, and interactions). Some of these efforts, such as UMLsec [15] support formal security analysis to verify the satisfaction of the specified security properties. Few of these [24] have toolsets that help, very limited, in generating security code or configurations with the system source code based on using model-driven engineering techniques. Most of these efforts [15], [18] do not address how these security requirements are designed and implemented in these systems. Thus, software developers will typically have to build these security requirements together with the system business function implementations. For example, a specified security property using UMLsec for example on a business function will be achieved by adding appropriate security code with the business

function code. By its very nature, this leads quickly to systems with built-in (hardcoded) security capabilities that are often hard to modify [19]. This approach also complicates the integration with third party security controls as it often requires manual effort by security engineers. Hardcoding such security within software systems limits the flexibility to adapt enforced security in order to meet the new security challenges. Integrating systems with built-in security with customers' operational environment security management systems requires reengineering such systems to inject new security integration code. When multi-tenant systems are considered, this hard-coded approach to security further complicates update of security requirements which often differ by tenant. While some techniques have been experimented to address this e.g. using dynamic Aspect-oriented Programming to inject updated security code [24], this is typically disconnected from the security requirements and design models.
In the area of SaaS applications security engineering, we have determined two key problems: maintaining isolation between different tenants' data; and enforcing different tenants' security requirements. The first problem can be easily addressed at design-time as one of the key security requirements. However, the later problem is hard to incorporate at design-time as software tenants and their security requirements emerge at runtime. Thus, we claim that the solution to this problem requires a different security model as we introduce in this paper.

Component-based (CBSE) and service-oriented (SOA) security engineering approaches [13][25] do support late security engineering (deployment time). However, most of these approaches focus on generating security code, using Aspect-oriented Programming (AOP). These approaches benefit from, but are limited by, the underlying application architecture (CBSE, SOA) to deliver flexible and adaptable security. Some adaptive security engineering approaches have been investigated [14, 33, 10]. However most focus on low-level details or limited to specific security attributes e.g. adaptive access control. These efforts require preparing applications at design time to support runtime adaptation. Thus, these efforts cannot be adapted to deliver multi-tenant SaaS application security engineering. New research efforts in securing multi-tenant SaaS applications have focused on: (i) (re)engineering multi-tenant SaaS applications to extend their security capabilities [8, 7]; (ii) maintaining isolation between different tenants' data at rest, at processing or at transmission [12, 29]; and (iii) developing security controls and architectures that deliver SaaS application security (e.g. access control) taking into account multi-tenancy dimension [40, 38]. Most of those efforts depend on or lead to built-in, or predefined, security architectures for SaaS applications. Thus tackling the loss of control concerns raised by cloud consumers – i.e. capturing and enforcing tenants' security requirements, and integration of SaaS applications with tenants' security infrastructure are not addressed before.

Furthermore, existing industrial security platforms such as the Java Security Model, Spring Security Framework (acegi), and Microsoft WIF, provide a

set of security mechanisms that help developers in securing their applications and reducing number of errors that arise from using custom security functions. Using these frameworks requires system engineers to write integration code exposing platforms supported APIs at every critical application entity. The resultant applications still have built-in security and are tightly coupled with the adopted platform. Integration with third-party security controls requires manual development and configuration changes.
A key gap in the multi-tenant application security area is that lack of adaptable security support as well as the lack of multi-tenant security engineering support i.e. to support capturing and enforcing different tenants' security requirements at runtime without a need to conduct application maintenance. We formulate this gap in the following research questions that we tackle in this paper:

- How can we capture different tenants' security requirements for different application or service entities?
- How can we enforce different tenants' security requirements on any arbitrary application or service entity?
- How can we verify that critical entities correctly enforce specified security needs?
- How can we carry out these tasks at runtime, as tenants emerge after application or service deployment?

In this paper, we introduce a novel approach called MDSE@R (Model-Driven Security Engineering at Runtime) for multi-tenant cloud-based applications. MDSE@R supports capturing, enforcing, and verifying different tenants' and service providers' security requirements at runtime without a need to modify/customize the underlying application implementation i.e. it works with both existing and new SaaS applications. Our approach is based on promoting security engineering efforts to be conducted at runtime instead of design time. This is facilitated by externalizing security from the target application. Thus both application and security can evolve at runtime. On the other hand, we automate the integration of whatever security requirements/controls within any application entity that was marked as critical/secure. The list of critical application entities emerge at runtime based on current risks.

Using MDSE@R, service providers deliver a service description model (SDM) as a part of the application delivery package, details are discussed in Section 4. The SDM is a mega-model (a mega-model [35] is a model that contains a set of models and a set of relations between these models) that contains details of the application features, architecture, classes, etc. Furthermore, they deliver a security specification model (SSM). The SSM is a mega-model that contains details of the security to be enforced on application entities (features, components, etc.). Rather than the mandatory security controls (i.e. security controls that cannot be replaced by tenants'), other security controls should not be built-in the delivered application, they rather implemented and deployed externally and weaved with the secured application entities at runtime.

Thus such controls can be updated, replaced, or disabled at runtime without modifying the target application. If this is not the case (i.e. the application is developed with all security controls built-in), we can use our preprocessing tool to disable such controls and deploy them externally [2]). Different sets of security controls can also be enforced at runtime.

During the provisioning of new tenants, the service provider creates an instance of the application or configures the shared instance to disable or enable certain features for the new tenant. This is reflected in the tenant copy of the application or the services description model. This SDM copy is called tenant service description model (TSDM). Tenants can specify their security details using their copy of the service SSM. This copy is called tenant security specification model (TSSM). Tenants can add new security controls or disable/replace/modify the existing security controls. At runtime, these models may be updated to reflect new security needs. These updates are automatically reflected on the target application using MDSE@R. The service provider can specify certain security controls as mandatory (or develop them within the application). This means that such controls cannot be disabled or modified by the application or service tenants. This is very important in enforcing security isolation controls, for example. Best practice patterns of common controls and configurations can be reused for common security needs by either service providers or service tenants.

Getting tenants involved in managing their asset security helps in reducing the lack-of-trust and mitigating the loss-of-control problems that arise from the adoption of cloud services. We have validated our approach on seven significant open source applications. We have conducted a security features evaluation, performance evaluation and user evaluation of our approach and prototype platform . Below we summarize the key contributions of in this paper:

- Model to capture detailed security requirements for different tenants in cloud application.
- Approach of linking security details to points of interest in target cloud application at differing levels of abstraction.
- Approach to take this system and security model and use to enforce requirements on running application.
- Demonstration of its ability to capture & enforce desired security on several third-party applications.

Section 2 begins with a motivating example for our research and identifies key challenges and requirements that must be satisfied by a multi-tenant security engineering approach. Section 3 reviews key related work. Section 4 provides an overview of our MDSE@R approach. Section 5 describes a usage example of our MDSE@R framework and toolset. Section 6 describes our framework architecture and implementation details. Section 7 presents our evaluation results of MDSE@R. Section 8 discusses implications of our approach, key strengths and weaknesses, and areas for further research.

2 Motivating Scenario

In this section, we introduce a simple and typical cloud scenario where a multi-tenant service has been procured by different tenants. Each of them is worried about the data security and has security requirements and controls that need to be applied. Such scenario cannot be satisfied by the existing security models i.e. supporting different security requirements on the same service instance.

Scenario. Consider "SwinSoft", a well-known software house in developing business applications. Swinsoft has recently developed a new cloud-based SaaS ERP solution called "Galactic. Galactic is designed to support both multi-tenant models including single-tenant, single-instance; and multi-tenant, single-instance. SwinSoft hosts Galactic on a cloud platform delivered by GreenCloud (GC). During the development of Galactic, SwinSoft used external services to speed up the application development. These services include: Currency-Now and build workflow services to get up-to-date currency exchange rates and flexible workflow engine (developed and deployed on GC); and Batch-MPRD to conduct transactions' posting using the map-reduce model that improves and parallelizes the batch posting operations (hosted in BlueCloud - another cloud platform).

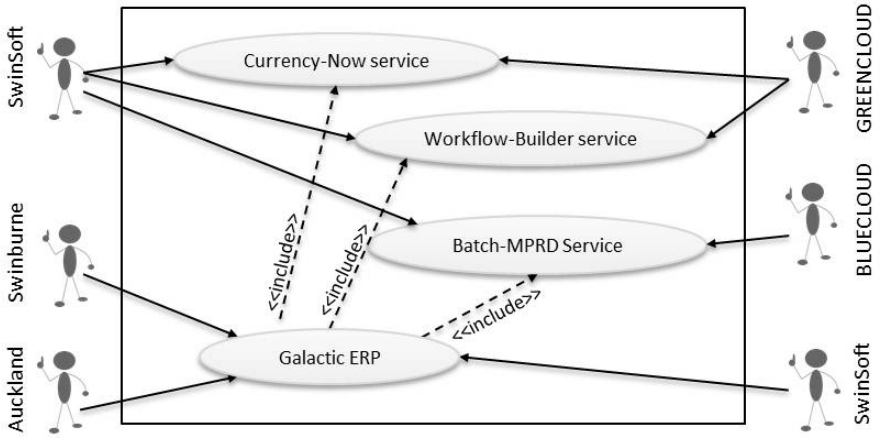


Fig. 1 Use case diagram for the motivating example

Swinburne University is going to purchase a new ERP solution in order to automate its internal process. After investigation of available solutions, Swinburne has decided to go for Galactic ERP solution, to save upfront investment required and keep infrastructure cost optimized. At the same time, "Auck-

land University has also decided to purchase the Galactic ERP application. However, each of these Galactic service tenants have their own, quite different, business functions and security objectives. Swinburne has special security requirements because it is ISO27000 certified. Swinburne security architects conduct periodic risk assessment. This may result in a requirement to reconfigure the enforced applications security to block newly discovered threats. It needs to maintain similar security policies on Galactic as those used in their local environment. This includes using active directory to support Single Sign-On (SSO), applying a role-based access control (RBAC) model on Galactic, their access control policies should consider end-user location and request time, integrity of data transmitted must be maintained, and confidentiality of Swinburne data must be enforced. Auckland assigns high risk to Galactic maintained assets because they are outsourced for hosting on external third-party cloud platform. Thus, they have strong security constraints that are different from their local systems. This includes applying an attribute-based access control (ABAC) model for access control [37], use of a two-factor authentication system, transaction accountability and audit-ability, and all data must be kept confidential. Both organizations thus would like to use the multi-tenant Galactic service while enforcing different security requirements and integrating with different security services.

Key challenges. The analysis of the above scenario identifies the following challenges: security requirements differ from one tenant to another; each tenant's security requirements may change over time based on current operational environment security and business objectives; Galactic security should support integration with each tenant's security controls in order to achieve coherent security solutions; and new security vulnerabilities may be discovered in Galactic application at any time. Using traditional security engineering techniques would require SwinSoft to conduct a lot of application maintenance iterations to deliver application patches that block vulnerabilities and adapt the application to every new customer needs. Multiple versions of the application, one for each tenant and with substantial differing security enforcement embedded in the application, would have to be maintained.

Key requirements. A new security engineering approach that addresses these challenges is needed. It should enable each tenant to specify and enforce their security requirements based on their current security needs. Security should be applied to any arbitrary application entity. No predefined security interception points specified at design time. It should support interception of any applicable application method. Security specification should be supported at different levels of abstraction based on the customers' experience, scale and engineers' capabilities. Integration of security with application entities should be supported at different levels of granularity, from the application as one unit to a specific application method. The security engineering approach should enable integration with third-party security controls. It should support

the application and security specifications to be reconfigured at both design time and runtime.

3 Related Work

Existing academic security engineering efforts have focused on capturing and enforcing security requirements at design time, supporting adaptive security, and multi-tenant security engineering. On the other hand, most industrial efforts have focused on delivering security platforms that can help software developers in implementing their security requirements using readymade standard security algorithms and mechanisms.

3.1 Design Time Security Engineering

Software security engineering aims to develop secure systems that remain dependable in the face of attacks [4]. Security engineering activities include: identifying security objectives that systems should satisfy; identifying security risks that threaten system operation; elicitation of security requirements that should be enforced on the system to achieve the expected security level; developing security architectures and designs that deliver the security requirements and integrates with the operational environment; and developing, deploying and enforcing the developed or purchased security controls. These efforts can be categorized as follows:

- Early-stage security engineering approaches focus only on security requirements elicitation and capturing at design time. KAOS [17] was extended to capture security requirements in terms of obstacles to stakeholders' goals. Obstacles are defined in terms of conditions that when satisfied will prevent certain goals from being achieved. Secure i* [18] focuses on identifying security requirements through analysing relationships between users, attackers, and agents of both parties. Secure Tropos [28] captures details about the security requirements and trust goals, introducing two categories of goals: hard goals that reflect system functional requirements and soft goals reflecting non-functional requirements (security).

- Later-stage security engineering approaches typically focus on security engineering during system design. Misuse cases [34] capture use cases that the system should not allow and may harm the system operation or security. UMLsec [15] extends UML with a profile that provides stereotypes to be used in annotating design elements with security intentions and requirements. UMLsec provides a comprehensive UML profile. However, it was originally developed for use during the design phase. UMLsec has stereotypes for predefined security requirements only (secrecy, secure dependency, critical,), , though it is possible to define extensions. Some extensions and applicatons of UMLsec enable it to be used to support later tasks

of software development e.g. security testing and verification [16**?**] and runtime verification of history-based properties [5]. A limitation we have found with UMLsec is that it mixes security with system entities at design time, which we have found complicates modifications of system security capabilities especially when they are applied at runtime. SecureUML [19] provides a meta-model to design RBAC policies of target systems. Both approaches are tightly coupled with system design models. Both early and later stage approaches lack a complete security model that captures security details and abstraction levels. Both do not support generating security code that realizes the specified security requirements.

– Security engineering processes include SQUARE [20], SREP [21] and Microsoft SDL. Such processes specify the steps to follow when capturing, modelling, and coding system security requirements. Such processes are aligned with system development processes. Most security approaches and processes focus on engineering security at design time. They often make assumptions about the security of the environments in which an application will operate. It is often difficult to integrate system's security with the operational environment security as software systems depend on their built-in security controls.

3.2 Adaptive Application Security

Several research efforts try to enable systems to adapt their security capabilities at runtime. Extensible Security Infrastructure [14] is a framework that enables systems to support adaptive authorization enforcement through updating in memory authorization policy objects with new low level C code policies. It requires developing wrappers for every system resource that catch calls to such resource and check authorization policies. Strata Security API [33] where systems are hosted on a strata virtual machine which enables interception of system execution at instruction level based on user security policies. The framework does not support securing distributed systems and it focuses on low level policies specified in C code.

Serenity [32] enables provisioning of appropriate security and dependability mechanisms for ambient/intelligence systems at runtime. Security attributes are specified on system components at design time. At runtime the framework links such Serenity-aware systems to the appropriate security and dependability patterns. Serenity does not support dynamic or runtime adaptation for new unanticipated security requirements.

Morin et al. [26] propose a security-driven and model-based dynamic adaptation approach enabling adapting applications to reflect defined context-aware access control (AC) policies. Engineers define security policies that take

into consideration context information. Whenever the system context changes, the proposed approach updates the system architecture to enforce the suitable security policies. Mouelhi et [27] introduce a model-driven security engineering approach to specify and enforce system access control policies at design time based on AOP-static weaving. These adaptive approaches require design time preparation (to manually write integration code or to use specific platform or architecture). They support limited security objectives such as AC. Unanticipated security requirements are not supported. No validation that the target system (after adaptation) correctly enforces security as specified.

3.3 Multi-tenancy security Engineering

The area of multi-tenant SaaS applications' security is relatively new. Possible solutions to multi-tenancy security are still under development by both industry and academia. Michael et al [6] discuss the limitations of security solutions proposed by different commercial cloud platforms. SalesForce [1] has introduced a simplified solution to support their CRM integration with tenants' security solutions. They focus on the Identity and Access Management (IAM) security content only. Tenants who are interested in integrating with SalesForce have to implement web services with a predefined signature.

Enabling applications to support multi-tenancy either during application development or by adapting existing web applications to support multi-tenancy has been investigated by [9, 23, 36, 39]. Cai et al [8, 7] propose an approach to transform existing web applications into multi-tenant SaaS applications. They focus on the isolation problem by analysing applications and identifying the required isolation points that should be handled by the application developers. Guo et al [12] developed a multi-tenancy enabling framework. The framework supports a set of common services that provide security isolation, performance isolation, etc. Their security isolation pattern considers the case of different security requirements of different tenants while still using a predefined, built-in, security controls .It depends on the tenants administration staff to manually configure security policies and map their users and roles to the application predefined roles.

Pervez et al [30] developed a SaaS architecture that supports multi-tenancy, security and load dissemination. The architecture is based on a set of services that provide routing, logging, security. Their proposed security service delivers predefined authentication and authorization mechanisms. No control by service consumers of the security mechanisms is supported and no isolation is provided between the authentication and authorization of data of different tenants. Xu et al [38] proposed a new hierarchical access control model for the SaaS model. Their model adds higher levels to the access control policy hierarchy to be able to capture new roles such as service providers' administrators (super and regional) and tenants' administrators. Service provider administra-

tors delegate the authorization to the tenants' administrators to grant access rights to their corresponding resources.

Zhong et al. [40] propose a framework that tackles the trust problem between service consumers, service providers and cloud providers on being able to inspect or modify data under processing in memory. Their framework delivers a trusted execution environment based on encrypting and decrypting data before and after processing inside the execution environment while protecting the computing module from being access from outside the execution environment. Menzel et al [22] propose a model-driven platform to compose services that represent the SaaS application. Their approach focuses on enabling cloud consumers to compose their system instances while defining their security requirements to be enforced on the composed web services. These security requirements are transformed into WS-policies-alike, applied on the resulting application, and deployed on a separate VM. The limitation of this approach is that it depends on building separate instances for each consumer. There is no means to update or reconfigure the defined security requirements or to extend the enforced security using third party security controls. These efforts we have surveyed deliver security using specific solutions and architectures.

3.4 Industrial Security Platforms

Existing industrial platforms including the Java Security Model, Spring Security Framework (acegi), and Microsoft Windows Identify Foundations (WIF), all help in securing systems by providing a set of security functions and mechanisms. However they usually require developers involvement in writing integration code with such platforms. Thus the resultant systems are tightly coupled with these platforms' capabilities and mechanisms. In addition, using third-party controls requires updating the system source code. Compared to existing efforts, MDSE@R does not assume specific system architecture or a security platform; no security code is required and no developers' involvement in integrating security controls; provides a comprehensive and extensible security model; and third-party security controls can be easily integrated with the target system.

4 MDSE@R

Our model-driven security engineering at runtime (MDSE@R) approach enables service tenants to be involved in securing their cloud hosted assets. As a consequence of tenants' involvement, a single service need to support capturing and enforcing different sets of security requirements of different tenants that become known at runtime. We name this as "Tenant-oriented Security" compared to the traditional security-oriented security where a service instance

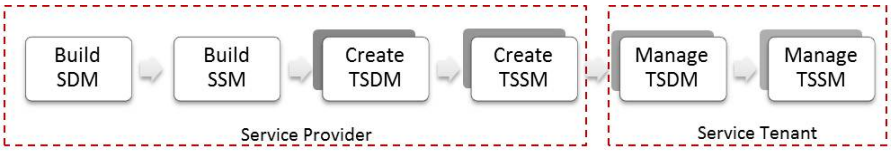


Fig. 2 Process flow of MDSE@R

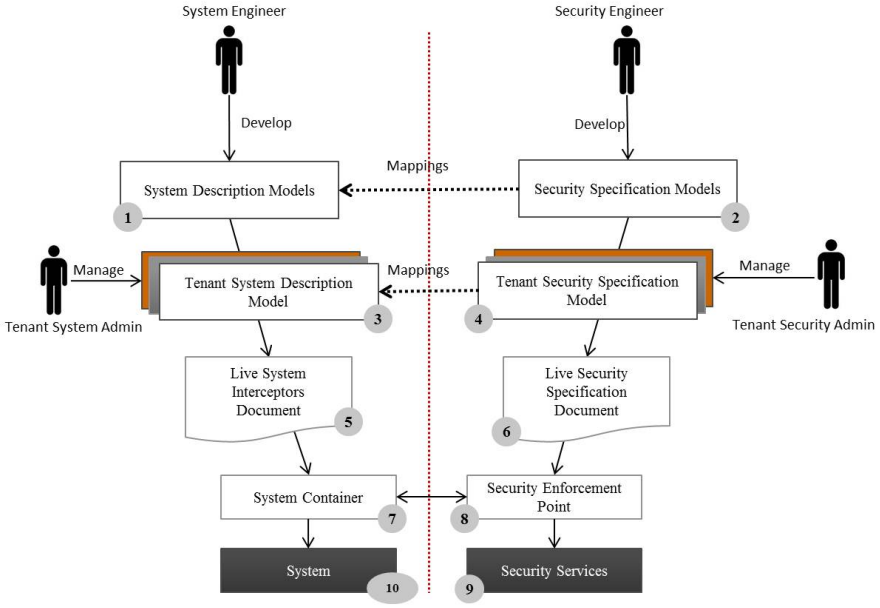


Fig. 3 Overview of MDSE@R approach

reflects only one set of security controls captured by the service provider at design time. Tenant-oriented security may require integrating cloud services with security controls selected by tenants and deployed on or out of the cloud platform. We base our MDSE@R on two key concepts: (i) externalizing security management and enforcement tasks from the application to be secured while being able to wrap the application and intercept calls to any arbitrary critical application entity at runtime using dynamic weaving AOP; and (ii) Model-Driven Engineering (MDE), using Domain-Specific Visual Language (DSVL) models to capture application and security attributes at different levels of abstraction. We automate the generation of security controls integration code rather than using hand-coding of bespoke solutions.

Fig. 2 shows the basic flow of MDSE@R to support multi-tenant security engineering. Service providers develop a detailed service description model (SDM). Then, they develop a security specification model (SSM) capturing all security details they deliver in their cloud service. Once a service tenant

registers to use the service, they will get a copy of the service SDM and SSM. Tenants can then use these models to manage (updated, delete, add) their instances and develop their security needs. Tenants can modify their security requirements and MDSE@R will then automatically update the enforced security on the running application to meet the tenant's new security requirements.

Fig. 3 gives an overview (covering main artifacts developed, key stakeholders, and key interactions) of the MDSE@R approach to support multi-tenant security engineering and tenants' security management at runtime. After capturing application and security models, the MDSE@R platform realizes such modeled changes using interceptors and AOP approach that injects security handlers into the target (secured) application entities (components, classes, and methods) as follows.

4.1 Modeling Service and Security Details

In this phase, stakeholders, from both the service providers and tenants, develop different models that capture details of the service, tenant instance, service security, and tenants' security details as follows:

4.1.1 Build Service Description Model (SDM)

A detailed service description model is delivered by the service provider (an example is shown in Fig. 7). This SDM captures various details of the target application including system features (using use case diagrams), system architecture (using component diagrams), system classes (using class diagrams), system behavior (using sequence diagrams), and system deployment (using deployment diagrams). These models cover most of the perspectives that may be required in securing a given system. Not all these models are mandatory. Tenant security engineers may need to specify security on system entities (using system components and/or classes models), on system status (using system behavior model), on hosting nodes (using system deployment model), or on external system interactions (using system context model). They may specify their security requirements on a coarse-grained level (using system features and components models), or on a fine-grained (using system class diagrams). The service SDMs can be synchronized with the running instance using models@runtime synchronization techniques [11], or manually by the service provider (models@runtime (reflection) research efforts enable management of consistency between system models and running software instances at runtime by reflecting any software changes to system models and vice versa). This also helps in supporting dynamic adaptation i.e. how to develop adaptive systems where system updates are applied on system models and then realized on system instances.

Some of the application or service description details, specifically the system class diagrams, can be reverse-engineered, if not available, from the target

application. We developed a new UML profile to extend UML models with attributes that help in: (i) capturing relations between different system entities in different models e.g. a feature entity in a feature model with its related components in the component model and a component entity with its related classes in the class diagram; and (ii) capturing security concepts/attributes (requirements, controls, etc.) mapped to the SDM entities e.g. security requirements specified on a given system feature or component. This helps in security enforcement and models weaving as we discuss later.

4.1.2 Build Service Security Specification Model (SSM)

A set of models developed and managed by the service provider security engineers to specify the security requirements/controls that the service providers enforce on their services (an example is shown in Fig. 8). It covers the details required during the security engineering process including: security goals and objectives, security risks and threats, security requirements, security architecture for the operational environment, and security controls to be enforced. These models capture different levels of abstractions. The key mandatory model in the security specification models set is the security controls model. It is required for generating the security integration required code (as described below).

4.1.3 Manage Tenant Service Description Model (TSDM)

This model describes system features, architecture and classes available for tenant T. It is usually different from one tenant to another. It depends on the multi-tenancy model adopted by the service provider in customizing tenant instance or configuring a single shared service instance. At tenant provisioning time, an initial TSDM is created as a copy from the system SDM. Then TSDM is updated to reflect current tenant's service instance details. Tenant system administrator can use this model later to turn features on/off at runtime. The TSDM helps in two scenarios: (i) to customize or configure the system based on tenant requirements e.g. tenant T is permitted to use certain features that he registered for. The service provider uses the tenant initial TSDM and delete other system features that are not required. The same approach can be used in both cases either the tenant has a separate instance or share the same instance with other tenants. Still the model@runtime synchronization techniques can be used to keep the TSDM synchronized with the running tenant instance; and (ii) to capture tenants' security requirements on their instance scope i.e. their features, components, methods, etc.

4.1.4 Manage Tenant Security Specification Model (TSSM)

This model is the tenant copy of the service SSM. It describes security objectives, requirements, architecture, design, and controls that the service tenants

have and want to enforce on their cloud-hosted assets. This may include authentication, authorization, auditing, encryption controls that tenants use in their internal sites or even from other security vendors. Tenants may decide to continue using the same security provided by the service provider or rather prefer to use their security controls. However, tenants will not be able to disable security controls that the service provider has marked as mandatory in the service SSM. This helps to avoid disabling critical security controls such as tenants' data isolation control provided by the service provider. This model can be used by tenants to manage security of multiple cloud-hosted applications i.e. single security model to manage all enterprise outsourced assets.

4.2 Weaving Service and Security Models into a Secure-Service Model

MDSE@R has two mapping levels. First, mapping SSM entities to service SDM entities. This mapping is developed and managed by the service providers at design time, deployment time, or even at runtime. Whenever the service provider discovers a security problem or has a new security requirement, they can directly apply it on the service security specification model and then map it to the service description model. Such a mapping is directly reflected on the tenants' models. Second, mapping TSSM entities to service TSDM entities. This mapping is developed and managed by the service tenant at runtime. Both mappings can be modified at runtime to reflect new needs.

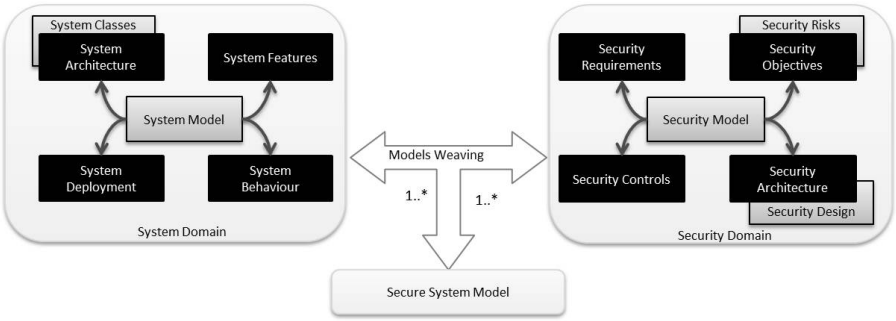


Fig. 4 Possible Weavings of service and security models

MDSE@R supports many-to-many mapping between the (tenant) service description model entities and (tenant) security specification model (SSM) entities, as shown in Fig.4. This is supported by our UML profile which extends every service description concept i.e. feature, component, class, method, host, connection, etc. with a set of security attributes i.e. security objectives, requirements, services and controls (see Fig. 11 for UML profile diagram). Using drag-and-drop between the SSM and SDM entities, SSM entity will be added as an attribute value, based on the dragged SSM entity, to the selected SDM

entity. One or more security entities (security objective, requirement and/or control) can be mapped to one or more service model entity (feature, component, class or method). Mapping a security concept on an abstract service entity e.g. a system feature - implies a delegation of the same security concept to the concrete entities e.g. the feature realization classes and methods. This is facilitated using our UML profile, which helps in managing traceability between application entities. Mapping an abstract security concept e.g. a security objective to a service entity - e.g. a class - implies mapping all security requirements, services, and controls that realize this security objective to that class. Any application entity that has a security mapping is called a critical application entity.

4.3 Enforcing Specified Security on Target Application Entities

In the previous steps, both security details and critical application entities emerge at runtime. MDSE@R automates the realization of the specified security on the critical application entities without the involvement of security or application engineers. This helps both parties to easily update their application and security capabilities to meet their needs. This helps in avoiding inconsistency problems that usually arise from the need to maintain both models and realizations, which force administrators and developers to update the security realizations directly.

Whenever the service provider or service tenant develops a new mapping or updates an existing mapping between an SSM entity and an SDM entity, the underlying MDSE@R platform propagates these changes as follows:

- Update Live Service Interceptors' Document (Fig.3-5). This document maintains the list of critical application entities (CP -an application entity that has security attributes mapped on it) where security controls should be weaved or integrated. Equation 1 states that the critical service entities - CP(s) - are the union of all tenants' critical points - CP (Ti) where To is the service provider.

$$CP(s) = \bigcup_{i=1}^{i=n} CP(T_i) \tag{1}$$

- Update Live Security Specification Document (Fig.3-6). This document maintains the list of security controls to be applied at every critical system entity. This may be defined by the service provider or by the service tenant. In this case we have to mark the service provider security controls as they should be enforced/applied first before any request to this critical system entity.
- Update the System Container (Fig.3-7). The container is responsible for intercepting system calls to critical system entities at runtime and delegating such requests to the default handler, the "Security Enforcement Point. Update Tenant Accessible Resources Document. This document maintains a list of system resources that should not be accessible for each tenant e.g.

if Swinburne did not buy the Customer Management module, they should not be able to access webpages or functionalities provided in this module. Equation 2 is used in specifying the tenant's inaccessible resources. The prohibited resources list for tenant Ti is the difference between the service SDM resources and the tenant Ti TSDM resources.

$$PR(T_i) = R(SDM) - R(TSDM(T_i)) \quad (2)$$

This list of tenant's prohibited resources is used by the MDSE@R to deny access to any of them for any given request submitted by one of the tenant's users. The application or service is now ready to enforce security specified by tenants and service providers based on the woven secure-service model. This update is conducted in parallel with the application or service operation. Thus it does not incur any further performance overhead.

4.4 Security Services

A key objective of MDSE@R is to avoid being tightly coupled with specific security controls, specific vendor, or specific security platform (Java security manager, spring acegi framework, Microsoft Windows Identity Foundations, etc.). in addition to keeping developers and administrators away from being deeply involved in integrating security controls with a target system which usually result in inconsistent security being enforced on different systems. We developed a common security interface for every security attribute (authentication, authorization). This interface, shown in Fig.5, specifies functions and signatures that each security control expects/requires in order to perform their tasks e.g. user identity, credentials, roles, permissions, claims, etc. A security control or service vendor must implement this interface in their connector or adapter to support integration with MDSE@R. This helps security vendors develop one connector that – using MDSE@R – can be integrated with all target applications/services.

4.5 Security Enforcement Point - SEP

So far we have prepared the system to intercept requests to critical methods via the system container and have prepared security controls to be communicated using the common security interface. The Security Enforcement Point (SEP) works as a bridge between the system container and the deployed security controls. SEP queries the security specification document for controls to enforce at every intercepted request. It then initiates calls (using the common security interface) to the designated security controls' clients or connectors. The SEP assigns results returned by such controls to the system context e.g. an authentication control returns userID of the requesting user after being authenticated. The SEP creates an Identity object from this userID and assigns it to the current thread' user identity attribute. Thus a secured application

```
1  public interface SecurityStandardInterfaceAuthentication
2  {
3      //Authentication
4      public string[] AuthenitcateUser();
5      public bool IsAuthenitcated(object subject);
6  }
7  public interface SecurityStandardInterfaceAuthorization
8  {
9      //Authorization
10     public string[] AuthorizeUser(object subject);
11     public bool IsAuthorized(object subject, object action,
12             object resource, string context);
13 }
14
15 public interface SecurityStandardInterfaceCryptography
16 {
17     //Cryptography
18     public string Hash(string plaintext);
19     public string Encrypt(string plaintext);
20     public string Decrypt(string ciphertext);
21     public string DigitalSign(string data);
22     public bool VerifySignature(string signature, string data);
23 }
24 public interface SecurityStandardInterfaceEncoding
25 {
26     //Encoding
27     public string Encode(string input);
28     public void AddCSRFToken();
29
30 public interface SecurityStandardInterfaceLogging
31 {
32     //Logging
33     public void Log(int type, string message);
34 }
35 public interface SecurityStandardInterfaceInputValidation
36 {
37     //Input Validation
38     public bool IsValid(string ruleName, string input)
39 }
```

Fig. 5 MDSE@R simplified common security interface

can work normally as if it has authenticated the user by itself. An application may use such information in its operations e.g. to insert a record in the DB, it uses the user identity to set the "enteredBy" DB field.

4.6 Testing the System-Security Integration

Before allowing the developed specifications and mappings to be applied to the live application or service, MDSE@R uses a security testing service to verify that the target application is now correctly enforcing the specified security

needs. We assume that security controls are already tested by the security vendors. Thus, our testing task should focus on verifying that security controls are correctly integrated within specified critical system entities. To automate this step, we use the live interceptors' document and the security specification document to generate a set of test cases (scripts) for each critical entity i.e. each critical entity will have a set of test cases according to the security specified on it (from the security specification document). Each test case verifies that a security control C is correctly integrated within the critical entity E. To test the correct integration, we simulate requests to E and check the resultant system security context (after calls - actual results) against the expected results e.g. user identity is correctly set, permissions are set as specified, etc. Finally, we generate a log of the test cases firing results to the tenant/service provider security engineers showing the failed test cases, the critical entities, and the failed to integrate security controls.

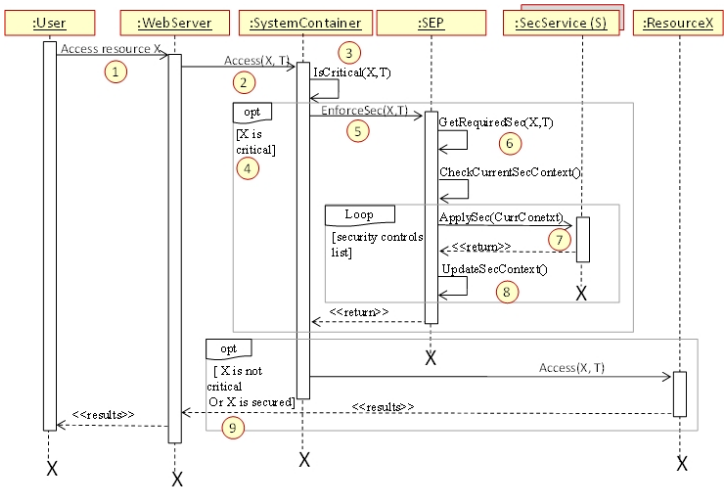


Fig. 6 Sequence diagram of a user request to critical application entity

Fig. 6 shows a sequence diagram describing a user requesting resource X, from a service operated by MDSE@R. In this scenario we have several interacting entities including user, webserver, MDSE@R system container, SEP, security services and the application resource. Once the web server receives a request submitted by the user to access resource X (1), it delegates the request to the system container along with the user's tenantID T (2). The system container queries the live service interceptors' document (3) to decide if the resource requested is marked as "critical" or not either by the tenant or by the service provider. If the resource is critical (4), the system container delegates the request to the security enforcement point (5). The SEP queries the security specification document for the required security controls to be enforced

on the requested resource by the user's tenant or by the service provider (6). This is an ordered list of security controls to be activated by the SEP. The SEP loops on the retrieved security controls list. Using the common security interface, the SEP generates requests to security controls required according to their type/family (7). After each security control call, the SEP updates the current threat security context (8). Finally, the SEP returns to the system container a recommendation to either proceed with the request or to deny it (9). If appropriate, the system container then forwards the request to the target resource or else returns an appropriate security exception to the caller.

5 Usage Example

The key objective of this usage example is to show how the service provider and tenants can collaborate together using MDSE@R and our platform toolsets to manage security of their services at runtime. We highlight the key stakeholders involved in the security engineering process along with their responsibilities and their expected outcomes of every step. We use the motivating example from Section 2, Galactic application developed by SwinSoft and procured by Swinburne and Auckland. SwinSoft wants to adapt its application security at runtime to block security holes and vulnerabilities that have been discovered at runtime. Two tenants using Galactic are worried about the security of their assets and have their own, different security requirements to be enforced on their Galactic ERP application instance(s). The examples of models in the Figures are taken from our prototype DSVL tool in use for this scenario.

5.1 Model Galactic System Description

This task is done during or after the system is developed. SwinSoft, the service provider, decides the level of application details to provide to their tenants in Galactic SDM. Fig. 7-A shows that SwinSoft SDM captures the description of system features including customer, employee and order management features (Fig. 7), system architecture details including Presentation Layer, Business Logic Layer (BLL) and Data Access Layer (DAL) (Fig. 7-B), system classes including CustomerBLL, OrderBLL, EmployeeBLL (Fig. 7-C), and system deployment including web server, application server, and data access server (Fig. 7-D). SwinSoft uses our UML profile (Fig. 13) to capture dependencies and relationships between system features and components, and system components and classes. This model is used as a reference by SwinSoft system and security engineers. No tenant is allowed to have write access to the Galactic SDM or its details.

5.2 Model SwinSoft Security Details

This task is conducted by SwinSoft (service provider) security engineers at the system deployment phase. This model is usually updated during their repetitive security management process to reflect new risks. In this scenario, SwinSoft security engineers document SwinSoft security objectives that must be satisfied by Galactic system (Fig. 8-A) including data integrity with medium importance, confidentiality with high importance, accountability with low importance. This model should be repeatedly revised to incorporate emerging changes in SwinSoft security objectives. Security engineers then refine these security objectives in terms of security requirements that must be implemented by the Galactic system, developing a security requirements model. A part of it is shown in Fig. 8-B including authentication requirements. This model keeps track of the security requirements and their links back to the high level security objectives. In this example, we show that the AuthenticateUser requirement is to be enforced on Galactic along with its detailed sub-requirements.

SwinSoft security engineers next develop a detailed security architecture including services and security mechanisms to be used in securing Galactic (Fig. 8-C). In this example, we show the different security zones (big boxes - Fig. 8-C) that cover SwinSoft network and the allocation of IT systems, including Galactic. The security architecture also shows the security services, security mechanisms and standards that should be deployed. SwinSoft security engineers finally specify the security controls (i.e. the real implementations) for the security services modelled in the security architecture model (Fig. 8-D). This includes SwinValidator, ESAPI.AccessController, and SecurityIsolator security controls. Each security control entity defined in the security controls model specifies its family (authentication, authorization, audit, etc.) and the deployment URL of its connector. Each security specification model maintains traceability information to parent model entities. In Fig. 8-d, we specify that SecurityIsolator realizes the "TenantsDataIsolation" requirement. Whenever MDSE@R finds a system entity with a mapped security requirement TenantsDataIsolation it adds SecurityIsolator as its realization control i.e. an SecurityIsolator check will run before the entity is accessed e.g. before a method is called or a module loaded. SwinSoft security engineers have to mark mandatory security controls that their tenants cannot modify or disable.

Fig. 7 Examples of the Galactic system description model

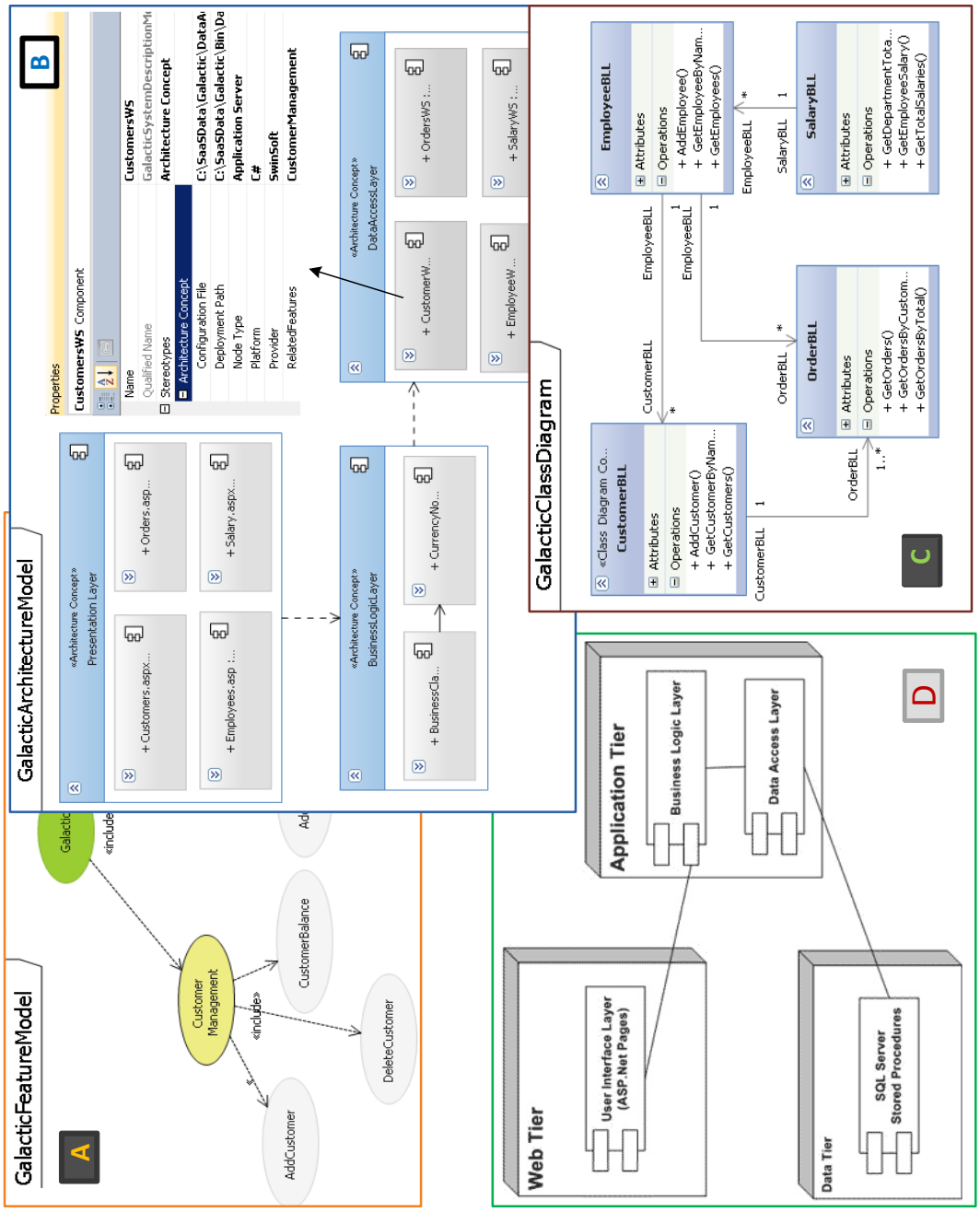
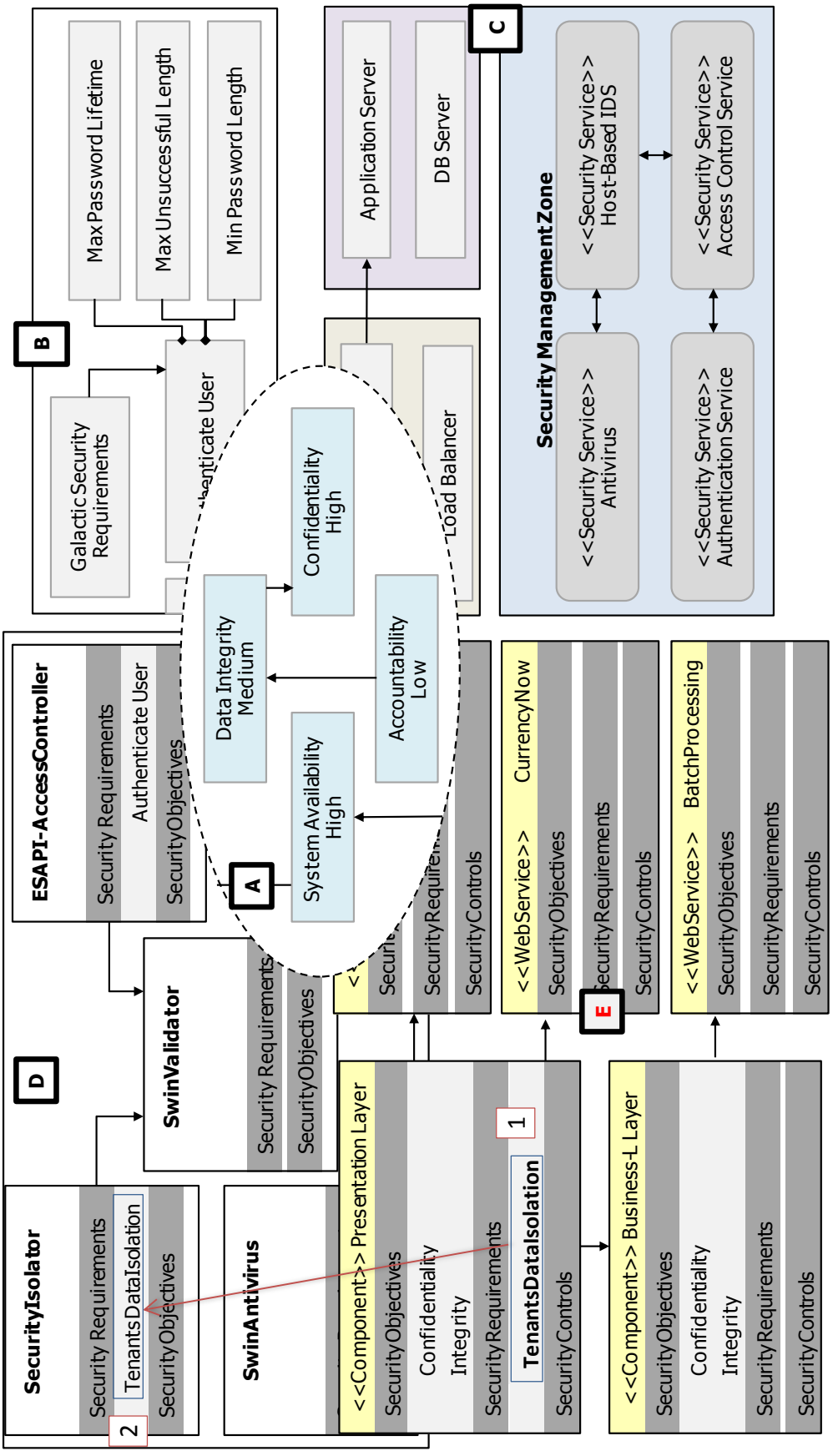


Fig. 8 Examples of SwinSoft Security Specification Models



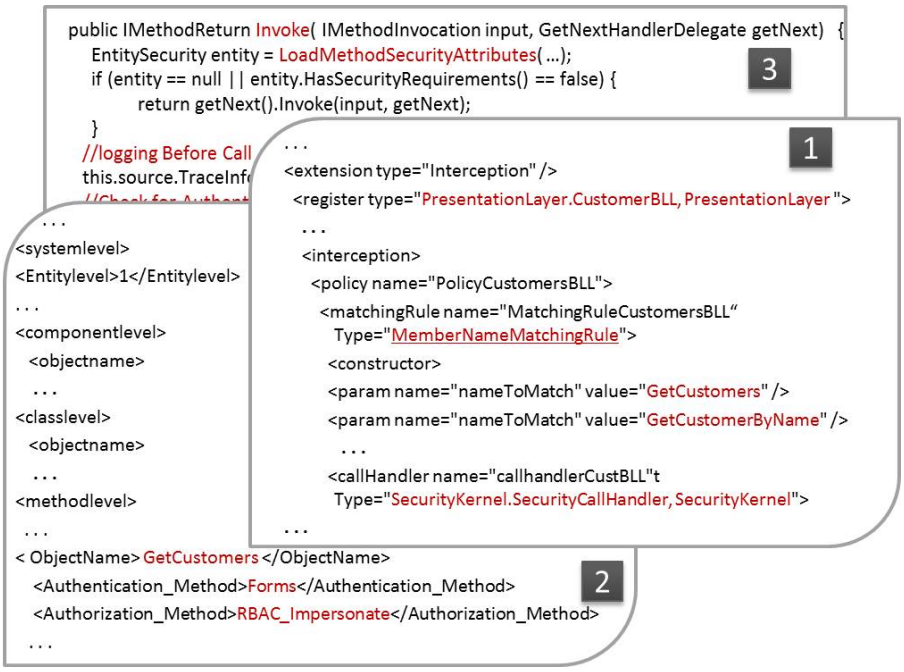


Fig. 9 Examples of the interceptors and security specification files

5.3 Weave System SDM and Security SSM

After the development of the Galactic SDMs and the security SSMs by SwinSoft security engineers, the SwinSoft security engineers map security attributes (in terms of objectives, requirements and controls) to Galactic system specification details (in terms of features, components, classes). This is achieved by drag and drop of security attributes to system entities. Thus, system feature, structure, or behaviour can dynamically and at runtime reflect different levels of security based on the currently mapped security attributes on it. Fig. 8-E shows a part of Galactic component diagram where PresentationLayer, a UML component entity, is extended with security objectives, requirements and controls compartments. In this example the security engineers have specified TenantsDataIsolation as one of the security requirement to be enforced on the PresentationLayer component (1). Such a requirement is achieved indirectly using SecurityIsolator control (2). MDSE@R uses the security attributes mapped to system entities to generate the full set of interceptors for system method calls , as in Fig. 9-1 (system interceptors document), and application entities' required security controls, as in Fig. 9.2 (security specification document).

```
[Test]
public void AuthorizeGetCustomersTesting()  {
   CustomerBLL  obj = new CustomerBLL();
   GenericPrincipal testIdentity =
                    new GenericPrincipal(new GenericIdentity("TestIdentity"),
                                                 new string[] { "TestRole" });
   try {
          obj.GetCustomers();
       }
   catch(Exception ex) {
          Assert.Fail("Firing of AuthorizeGetCustomersTesting test case failed");
   }
   Assert.AreNotEqual(testIdentity, System.Threading.Thread.CurrentPrincipal,
                 " LDAP Authorization Control is not correctly plugged-in", null);
}
```

Fig. 10 MDSE@R samples of the generated security integration test cases

5.4 Testing Galactic Security

Once security has been specified and interceptors and configurations generated, MDSE@R verifies that the system is correctly enforcing security as specified. MDSE@R generates and fires a set of required security integration test cases. Our test case generator uses the system interceptors and security specification documents to generate a set of unit test cases for each method listed in the interception document. The live systems interceptor document represents the source of system points we need to test for security integration. The live security specification document represents the source of security controls that we need to test their integration with the critical system points. The MDSE@R testing service has a set of predefined security control unit test templates for each security control attributes e.g. authentication, authorization, input validation, etc., including tests for successful and unsuccessful security enforcement. These unit test templates are used to generate test cases by replacing tags with specific security control names and critical point names. These generated tests are then run against the application and the results inspected to determine pass/failure. An example of a generated test case is shown in Fig10. This contains a set of security assertions (one for each security attribute specified on a given system entity). During the firing phase, the security enforcement point is instrumented with logging transactions to reflect the calling method, called security control, and the returned values. Security engineers should check the security test cases firing log to verify that no errors introduced during the security controls integration with Galactic entities. After SwinSoft security engineers have checked the MDSE@R Security testing service log files and make sure that no integration errors have been introduced, they can publish their updated Galactic security model for their tenants.

5.5 On-boarding Swinburne and Auckland Tenants

During tenants on-boarding process (preparing the service to be used by the new tenant), SwinSoft system engineers/admins start to customize/configure Galactic instance for tenant based on their requirements and purchased modules. Depending on the adopted multi-tenancy model, they may register new features or components as well. The final Swinburne or Auckland TSDM looks like the Galactic SDM in Fig.7. Swinburne and Auckland system administrators can update their own tenants TSDMs to reflect any further system customization, such as enabling or disabling sub-features such as calculate overtime, nightshifts, and vacations in the Employee Management module. This TSDM is used by the Swinburne security engineers to define required security on it. The updates done by SwinSoft or Swinburne on the TSDM are reflected on the prohibited resources list (Eq.2)

5.6 Swinburne and Auckland Manage their TSDMs and TSSM

Swinburne and Auckland security engineers go through the same process as SwinSoft did when specifying their security requirements and controls. Each tenant can customize their TSSM as far as they want and as frequent as required. For example, in Fig. 11 Swinburne engineers have specified that LDAP "realizes the AuthenticateUser requirement. Whenever MDSE@R finds a system entity with a mapped security requirement AuthenticateUser it adds LDAP as its realization control i.e. an LDAP authentication check will run before the entity is accessed - e.g. before a method is called or a module loaded. This applies to the CustomerBLL class methods, Fig11-(1). However, Swinburne security engineers have a different requirement for the GetCustomers method - the requester should be authenticated using Forms-based authentication as well, Fig11-(2). Auckland can specify their specific requirements, context, and security controls based on their specific needs. This results in quite different generated security enforcement controls. Both Swinburne and Auckland security engineers can modify the security specifications while their Galactic application is in use. MDSE@R framework updates interceptors in the target systems and enforces changes to the security specification for each system as required. For example, the Swinburne Galactic security model can be updated with a Shibboleth single sign-on [1]security authentication component and these updates applied to the running Galactic deployment.

6 MDSE@R Platform Architecture and Implementation

The architecture of MDSE@R platform is shown in Fig. 12. This is designed to support managing multiple SaaS applications hosted on a cloud platform. MDSE@R consists of system and security specification modellers, models and

[1] http://shibboleth.net/

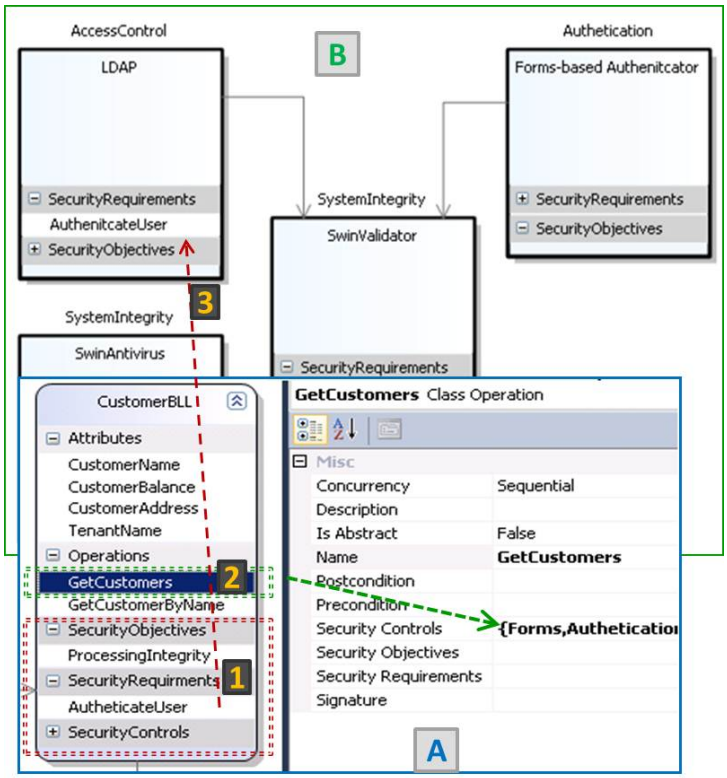


Fig. 11 Examples of Swinburne Security Specification Models

security controls specifications repositories, system container to intercept request, testing services and security enforcement point.

Our System Description Modeller (1) was developed as an extension of Microsoft Visual Studio 2010 modeller with an UML profile (Fig. 13) to enable system engineers modelling their systems' details with different perspectives including system features, components, deployment, and classes. The UML profile, as shown in Fig. 13, defines stereotypes and attributes to maintain the track back and forward relations between entities from different models. A set of security attributes to maintain the security concepts (objectives, requirements and controls) mapped to every system entity (Fig.7). The minimum level of details expected from the system provider is the system deployment model. MDSE@R can use this model to reverse engineer system classes and methods using .NET Reflection (in case of system binaries only available). We use .NET parsers to extract classes and methods from the system source code by analysing the generated Abstract Syntax Tree (AST) files by .NET parsers.

Our Security Specification Modeler (2) was developed as a Microsoft Visual Studio 2010 plug-in. It enables service providers and tenants, represented

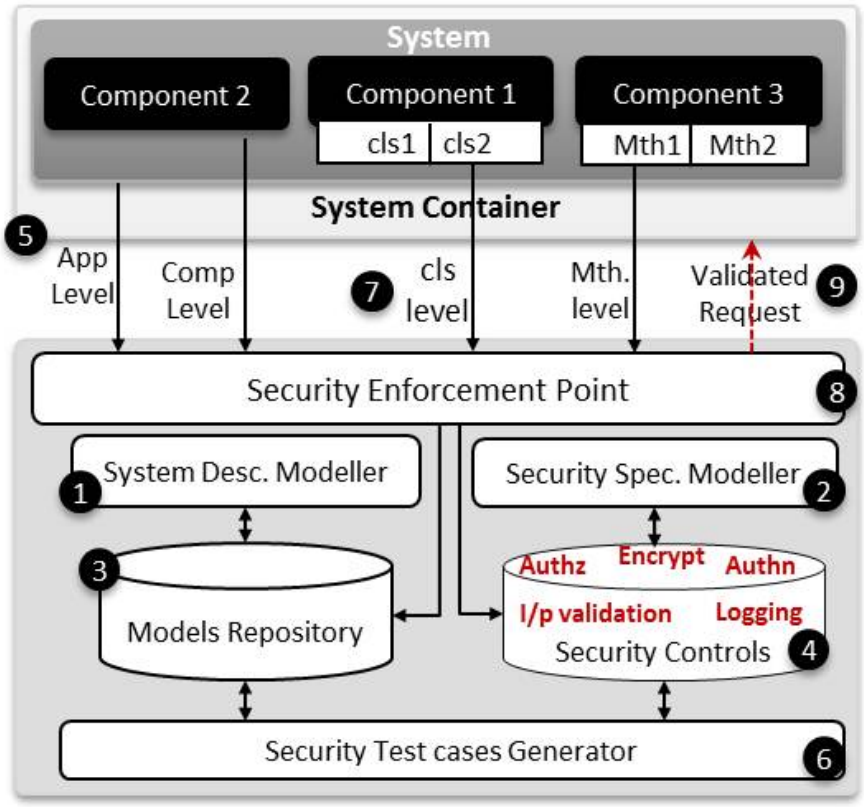


Fig. 12 MDSE@R platform architecture

by their security engineers, to specify the security attributes and capabilities that must be enforced on the service and/or its operational environment. The security modeler delivers a set of complete security DSVLs. Fig. 14 shows the meta-model of the MDSE@R security DSVL [2]. The security-objectives DSL captures customers security objectives and the relationships between them. Each objective has a criticality level and the defence strategy to be followed: preventive, detective or recovery. The Security requirements DSL captures customer's security requirements and relationships between requirements including composition and referencing relations. The Security Architecture DSL captures security architectures and designs of the customer operational environment in terms of security zones and security level for each zone; security objectives, requirements and controls to be enforced in each zone; components and systems to be hosted in each zone; security services, mechanisms and standards to be deployed in each zone or referenced from other zones. The Security Controls DSL captures details of security controls that are registered and de-

[2] http://www.ict.swin.edu.au/personal/malmorsy/Pubs/TR002.pdf

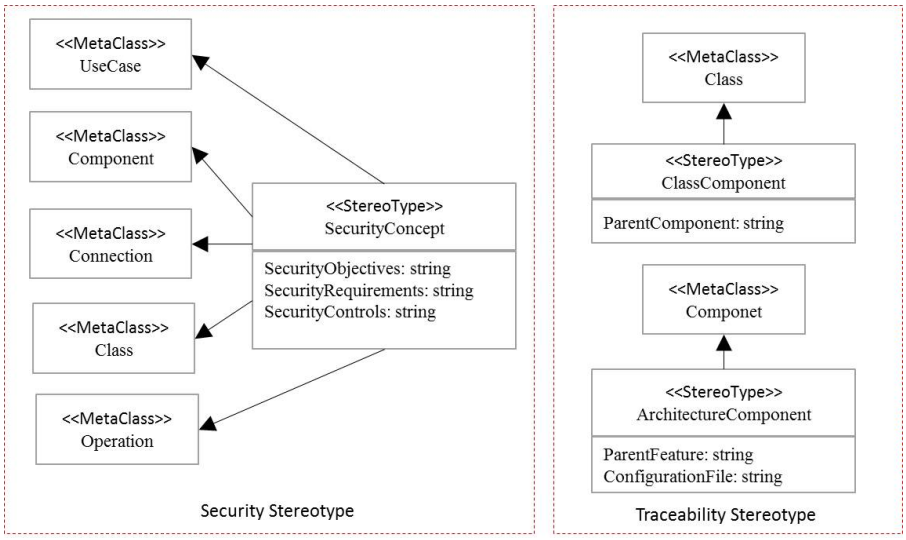


Fig. 13 MDSE@R system description model UML profile

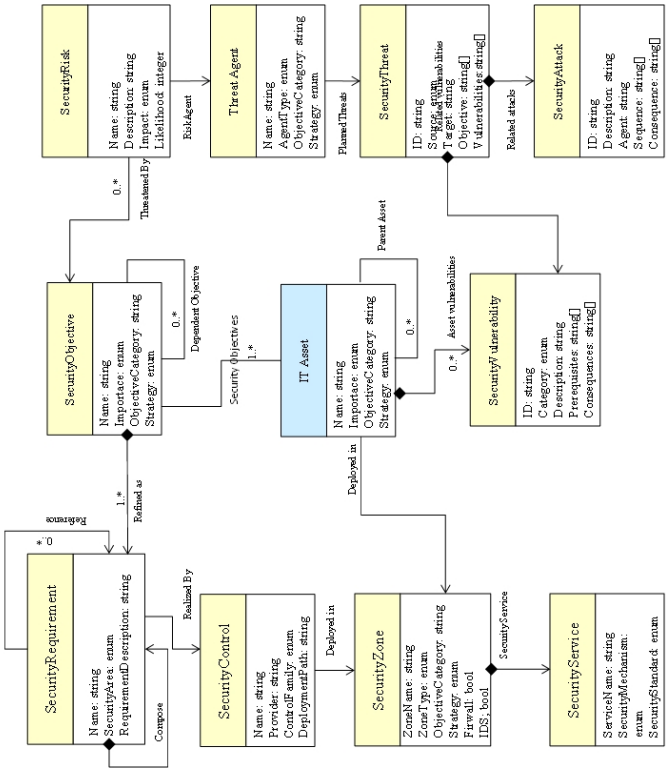


Fig. 14 MDSE@R security meta-model

ployed in the customer environment and relationships between these and the security requirements they cover.

Models Repository (3): Both modeling tools use a shared repository to maintain models developed either by the system engineers or the security engineers. This repository also maintains the live system interceptors' document and security specification document. An example of these documents is shown in Fig 9. This example shows a sample of the Galactic interceptors document generated from the specified security-system mapping. It informs the system container to intercept GetCstomerByName and GetCustomers methods (1); a sample of Swinburne security specification file defining the security controls to be enforced on every intercepted point (2); and a sample of the security enforcement point API that injects the necessary security control calls before and after application code is run (3).

Security Controls Database (4) is a database of the available and registered security patterns and controls. It can be extended by the service providers or by a third party security provider. A security control must implement certain APIs defined by the security enforcement point in order to be able to integrate with the target system security standard interface. Having a single enforcement point with a predefined security interface for each security controls family enables security providers to integrate with systems without having to redevelop adopters for every system. We adopted OWASP Enterprise Security API (ESAPI) library [3] as our security controls database. It provides a set of authentication, authorization, encryption, etc. controls that we used in testing our approach.

System Container (5): To support run-time security enforcement, MDSE@R uses a combined dependency injection and dynamic-weaving AOP approach. Whenever a client or application component sends a request to any critical system component method, this request is intercepted by the system container. The system container supports wrapping of both new developments and existing systems. For new development, Swinsoft system engineers should use the Unity application block delivered by Microsoft PnP team [4] to support intercepting any arbitrary class entity. Unity supports dynamic runtime injection of interceptors on methods, attributes and class constructors. For existing systems we adopted Yiihaw AOP [31] , where we can modify application binaries (dll and exe files) to add security aspects at any arbitrary system method (we add a call to our security enforcement point). For component level interception, we can use httpModules to add our interceptor on the component level.

Our Security Test Case Generator (6) uses the NUnit testing framework [5] to partially automate security controls and system integration testing. We developed a test case generator library that generates a set of security test cases for authentication, authorization, input validation, and cryptography for

[3] https://www.owasp.org/index.php
[4] http://pnp.azurewebsites.net/en-us/
[5] http://www.nunit.org/

every enforcement point defined in the interceptors document. MDSE@R uses NUnit library to fire the generated test cases and notifies security engineers via test case execution result logs.

At runtime, whenever a request for a system resource is received (7), the system container checks for the requested method in the live interceptors' document. If a matching found, the system container delegates this request with the given parameters to the security enforcement point (8). Security Enforcement Point (9) is a class library that is developed to act as the default interception handler and the mediator between the system and the security controls. Whenever a request for a target application operation is received, it checks the system security specification document to enforce the particular system security controls required. It then invokes such security controls through APIs published in the security control database (4). The security enforcement point validates a request via the appropriate security control(s) specified, e.g. imposes authentication, authorization, encryption or decryption of message contents. The validated request is then propagated to the target system method for execution (10).

7 Evaluation

In this section we summarize some of the experiments we have performed to assess the capabilities and scalability of our MDSE@R approach in:

- Capturing descriptions of different real systems and different security details for both service providers and tenants;
- Propagating security attributes on different system entities (features, components, classes, and methods);
- Enforcing unanticipated security requirements including authentication, authorization, auditing, etc. at runtime with an acceptable performance overhead;
- Validating that security controls are correctly integrated with the target entities.

7.1 Bechmark applications setup

We have selected a set of seven web-based, real-world, large, widely-used, and commercial open source web applications developed using ASP.Net (currently we have a .NET parser only) as our benchmark to evaluate MDSE@R. These applications cover a wide business spectrum. We divided the evaluation set into two groups: Group-1 (G-1) has two applications including Galactic (an ERP system developed internally in our group for testing purposes) and PetShop (a well-known reference e-Commerce application). Both applications have been modified to adopt the Unity application block as the system container. Group-2 (G-2) has five third-party web applications. SplendidCRM is an open source CRM that is developed to with the same capabilities of the well-known open

source SugarCRM system. It has been downloaded more than 400 times. SugarCRM has a commercial and community version. KOOBOO is an open source enterprise CMS used in developing websites. It has been downloaded more than 2000 times. BlogEngine is an open source ASP.NET 4.0 blogging engine. It has been downloaded more than 46000 times. BugTracer is an open-source, web-based bug tracking and general purpose issue tracking application. It has been downloaded more than 500 times. NopCommerce is an open-source eCommerce solution. It has more than 10 releases. For this group we use Yiihaw framework as the system container to inject interceptors into system binaries. Except for Galactic, we do not have any previous experience with these applications. Table 1 shows some statistics about the selected set of applications including Lines of codes in (KLOC), No. of files, No. of components, No. of classes, and No. of methods included. It is clear that benchmark applications vary from large-scale systems such as SplendidCRM, KOOBOO, and NopCommerce to small scale such as PetShop.

Table 1 Benchmark applications statistics

Benchmark	KLOC	Files	Components	Classes	Methods
Galactic	16.2	99	7	101	473
PetShop	7.8	15	5	25	256
SplendidCRM	245	816	28	6177	6107
KOOBOO	112	1178	34	7851	5083
NopCommerce	442	3781	45	5127	9110
BlogEngine	25.7	151	4	258	616
BugTracer	10	19	2	298	223

Table 2 Security controls used by service provider, Swinburne, Auckland

Sec. Attribute	SwinSoft Ctls	Swinburne Ctls	Auckland Ctls
Authn.	ESAPI	Forms-based	LDAP
Authz.	ESAPI	Forms-based	LDAP
I/P santization	ESAPI	-	-
Audit	ESAPI	PrivateAuditor	PrivateAuditor
Cryptography	ESAPI	DES	AES
Sec. Isolation	ESAPI	-	-

7.2 Experimental setup

Using MDSE@R, we developed three security specification models (SSM) with security objectives, requirements, and controls as in Fig. 7. One model for ser-

vice provider and two other models were copied from it and modified to reflect two security requirements sets. We specified security requirements and controls for authentication, authorization, input validation, logging and cryptography as shown in Table 2. We used MDSE@R to model the system description (SDMs) for applications in Group-1, as we know the details of these systems. For Group-2, we used system deployment diagram for these applications and used MDSE@R to reverse engineer systems' class diagram from there binaries. Thus for applications in Group-1 we should be able to map security to system features, components, classes, methods. However, in Group-2 we should be able to specify security on component, class, and method levels only.

7.3 Evaluation results

Table 3 shows security attributes that MDSE@R succeeds in capturing and enforcing at runtime, including authentication, authorization, input sanitization, auditing and cryptography. This represents most common security attributes. Table 3 also shows that MSDE@R succeeded in mapping and enforcing these security attributes on all systems in both Group-1 and Group-2 with different levels of system abstractions (F: feature, C: component, S: class, and M: method). Note that for Group-2 applications we do not have a system feature model to map and enforce security on this level. The enforcement of cryptography has a limitation with Group-2 applications especially when securing methods. This is because it requires that the caller and callee expect parameters of type String. To address this problem, we used format-preserving encryption (FPE) techniques. The output of these techniques is in the same format (type) of the input - i.e. if the input to encrypt is of type integer then the output is of the same type.

Table 3 Results of validating MDSE@R against Group-1 and Group-2 applications

Benchmark App.		Security Attributes					
		Sec. Isolation	Authn.	Authz.	Input Sanitization	Audit	Cryptography
Group-1	Galactic	F, C, S, M	F, C, S, M	F, C, S, M	F, C, S, M	F, C, S, M	F, C, S, M
	PetShop	F, C, S, M	F, C, S, M	F, C, S, M	F, C, S, M	F, C, S, M	F, C, S, M
Group-2	Splendid	C, S, M	C, S, M	C, S, M	C, S, M	C, S, M	(C, S, M)*
	KOOBOO	C, S, M	C, S, M	C, S, M	C, S, M	C, S, M	(C, S, M)*
	NopCommerce	C, S, M	C, S, M	C, S, M	C, S, M	C, S, M	(C, S, M)*
	BlogEngine	C, S, M	C, S, M	C, S, M	C, S, M	C, S, M	(C, S, M)*
	BugTracer	C, S, M	C, S, M	C, S, M	C, S, M	C, S, M	(C, S, M)*

7.4 User evaluation

We carried out a preliminary user evaluation of our tools and platform to assess MDSE@R approach and platform usability. We had seven post-graduate researchers, not involved in the development of the approach, use our developed tools and platform after receiving an hour training session on the tool

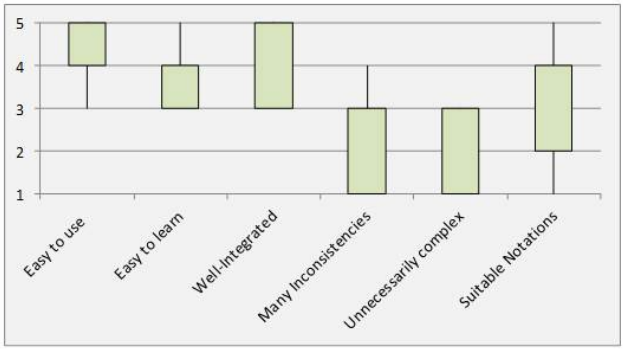


Fig. 15 Level of agreement of usability factors (1: Strongly disagree ... 5: Strongly agree)

and platform features. We asked them to explore several MDSE@R system and security DSVL specifications of the PetShop and Galactic applications. Then we asked them to perform updates on these models and to modify the security specification models at run-time. We conducted a basic usability survey to gain their feedback on our DSVL, modelling tools, and the security enforcement platform. The results show that they successfully understood and updated security models for the target systems. They gave positive feedback about the overall approach and the tool usability, and the capabilities in managing system security, as shown in Fig. 15 (1: Strongly disagree to 5: Strongly agree). A key recommendation was to use more expressive icons in the security DSVL rather than just boxes.

7.5 Performance evaluation

In this section, we discuss the performance evaluation of our MDSE@R platform in both runtime performance overhead ad offline adaptation overhead.

7.5.1 Runtime performance overhead

The runtime performance overhead of MDSE@R equals time incurred intercepting requests, plus time spent by the security enforcement module in querying the security requirements repository to be enforced on the intercepted point, plus time spent in calling the security controls specified. Time spent by the security controls themselves we do not factor in, as this needs to be spent whether using our approach or traditional hard-coded security solutions. Arguably, traditional approaches may incur some of these other time penalties as well e.g. checking authenticated user access controls or generating audit checkpoint information to log. Fig. 16 shows the time required (in msec) by MDSE@R to process a request for systems with different numbers of concurrent users and different number of system entities that have been marked as

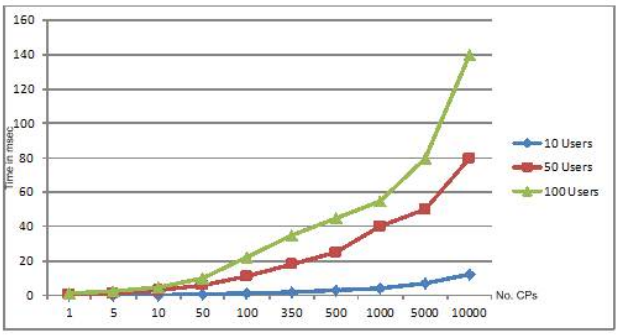


Fig. 16 Average performance overhead of MDSE@R platform

critical. Experiments were conducted on a Core2Duo desktop PC with 4GB Memory. The max performance overhead we got for a system with 10000 CPs defined and having 100 users concurrently sending requests equals 140msec. This performance considers efficient memory utilization as interceptors and security specification documents are loaded as needed. Significantly better performance could probably be achieved by caching these MDSE@R models in memory and using a hash table data structure to enable faster search. Using replicas of the MDSE@R platform on different servers and for different applications will result in further improvement of its performance overhead.

7.5.2 Security adaptation overhead

We have measured the adaptation delay incurred by MDSE@R in order to realize a single simple mapping between a security entity and a system entity e.g. system methods. This overhead equals on average 3 seconds. This represents the time taken to update the security interceptors and security specification documents and time to generate and fire the required integration test case(s). This is an offline task and so does not impact the performance of the system running instance.

8 Discussion

Our MDSE@R approach promotes multi-tenancy security engineering from design time to runtime. This is based on externalizing security engineering activities including capturing objectives, requirements controls, and realization from the target system implementation. This permits both security to be enforced and critical system entities to be secured to evolve at runtime (supporting adaptive security at runtime). It enables enforcing different security requirements sets for different tenants who are not known at design time. We name this as "Tenant-oriented security" compared to the traditional service-oriented security where a service can reflect only one set of security

requirements usually captured by service provider at design time. Finally, a key benefit reaped from MDSE@R approach is to the support model-based security management. Tenant security requirements, architecture and controls are maintained and enforced through a set of centralized TSSMs instead of low level scattered configurations and code that lack consistency and are difficult to modify. A tenant can have a single TSSM for all of their IT systems that captures all of their security specifications and can be updated anytime to reflect his new configurations. Thus any update to their TSSM will be reflected on all IT systems that use MDSE@R platform.

In our evaluation we developed one security model and used it with different systems. Each system enforces the security mapped to its entities. Any update to the security model results in updating all systems linked to it. This is a key issue in environments where multiple applications must enforce the same security requirements. Having one place to manage security reduces the probability of errors, delays, and inconsistencies. Automating the propagation of security changes to underlying systems simplifies the enterprise security management process. The multi-tenancy security engineering of existing services (extending system security capabilities) has three possible scenarios: (i) for systems that already have their SDMs, we can use MDSE@R directly to specify and enforce multi-tenant security at runtime; (ii) for systems without SDMs, we can reverse engineer parts of the required system models (specifically the class diagram) using MDSE@R (if these binaries can be read and not obfuscated). Then we can use MDSE@R to engineer required system security; (iii) for systems with built-in security, we can use MDSE@R to add new security capabilities only. MDSE@R cannot itself help modifying or disabling existing security. However, we have been working on extending our approach to support deletion of existing security methods and partial code using modified AOP techniques [2]

The selection of the level of details to apply security on depends on the criticality of the system. In some situations, we may intercept calls to the presentation layer only (webserver) while considering the other layers secured by default (not publicly accessible). In other cases, such as integration with a certain web service or using third party component, we may need to have security enforced at the method level (for certain methods only). Security and performance trade-off is another dilemma to consider. The more security validations and checks the more resources required. This impacts application performance. This should be included as a part of the Service Level Agreement (SLA) with the tenants. We plan to extend our generated test cases to include performance tests in the near future, allowing MDSE@R provider to assess the overhead of new security configurations in terms of cost and to help both providers and tenants to optimize the security level enforced. MDSE@R helps in engineering security into systems at runtime, while the security controls configuration and administration should be managed by security administrators. MDSE@R does not support defining business rules at runtime e.g. an employee should not be able to retrieve a customer's records if customer is of type VIP. The target

system should have this rule while MDSE@R will provide the current user roles/permissions as returned by the tenant security control.

Security isolation between different tenants' data is a very critical requirement in engineering security of a multi-tenant SaaS application. In MDSE@R, we consider security isolation as one of the security controls that simply performs authorization of the tenants supplied inputs before proceeding with the requests. Thus no tenant can access other tenants data by providing malicious inputs. However, the service providers have to perform the data filtration when loading/storing data from/to the application database. One may argue that our approach may lead to a more open and vulnerable system as we did not consider security engineering during design time. Our argument is that at design time security engineering is often done by security non-experts and this is a key reason why we still discover a lot of vulnerabilities in deployed systems. However, service providers can still perform security engineering during design time using MDSE@R. The service provider delivers both the SDM and SSM to their tenants for further customization. This also helps small tenants or tenants who are satisfied with the delivered security.

9 Summary

MDSE@R is a new model-driven approach to dynamically engineering security for multi-tenant SaaS applications at runtime. Our approach is based on using a set of multi-level service description models (SDM), developed by service providers, to describe different perspectives of their applications; a set of security specification models (SSM), developed by the service provider, to capture security objectives, requirements and environment security controls using Domain-Specific Visual Languages. Then, tenants can customize their copies of the SDM and SSM to reflect their application and security configurations. MDSE@R then bridges the gap between these two specifications through merging of the service and security models for both service provider and service tenants into a joint service security model.

MDSE@R uses dynamic injection of security enforcement interceptors and code into the target application to enforce the security specified. Security specifications are thus externalized and loosely coupled with application specifications, enabling both the application and security specification to evolve. It also allows sharing of security specification models among different applications "model-based security management". Security controls can be integrated with MDSE@R (which was implicitly integrated with the tenant service) by implementing a common security interface that we have introduced.

We have developed a set of modeling tools and a prototype of MDSE@R. We have successfully validated our approach by applying it to seven web-based applications, most of them open source, successfully modeling and enforcing a range of security needs on these applications. We performed a preliminary user evaluation of our toolset that demonstrates that it is readily usable by a technical audience but with little security engineering background. We assessed

the performance overheads of using our current prototype of MDSE@R. It has a performance overhead ranging from 0.13msec up to 140msec per request for each critical application entity. MDSE@R has adaptation delay of 3sec for each simple mapping between SSM and SDM. This represents time to update interceptors and security specification documents as well as generating and firing security test cases.

Acknowledgements Funding provided for this research by Swinburne University of Technology and FRST SPPI project is gratefully acknowledged. We also thank Swinburne University of Technology for their scholarship support for the first and third authors.

References

1. Akai, S., and Chiba, S. Extending aspectj for separating regions. ACM, 2009.
2. Almorsy, M., Grundy, J., and Ibrahim, A. S. Supporting automated software re-engineering using re-aspects. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering* (New York, NY, USA, 2012), ASE 2012, ACM, pp. 230–233.
3. Almorsy, M., Grundy, J., and Mueller, I. An analysis of the cloud computing security problem. In *Prof. of 2010 Asia Pacific Cloud Workshop, Colocated with APSEC* (Sydney, Australia, 2010).
4. Anderson, R. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley and Sons, 2001.
5. Bauer, A., and Juerjens, J. Security protocols, properties, and their monitoring. In *Proceedings of the fourth international workshop on Software engineering for secure systems* (New York, NY, USA, 2008), SESS '08, ACM, pp. 33–40.
6. Brock, M., and Goscinski, A. Toward a framework for cloud security algorithms and architectures for parallel processing. vol. 6082 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2010, pp. 254–263.
7. Cai, H., Wang, N., and Zhou, M. J. A transparent approach of enabling saas multi-tenancy in the cloud. In *Services (SERVICES-1), 2010 6th World Congress on* (5-10 July 2010 2010), pp. 40–47.
8. Cai, H., Zhang, K., Zhou, M. J., Gong, W., Cai, J. J., and Mao, X. S. An end-to-end methodology and toolkit for fine granularity saasization. In *Cloud Computing, 2009. CLOUD '09. IEEE International Conference on* (21-25 Sept. 2009 2009), pp. 101–108.
9. Chinchani, R., Iyer, A., Ngo, H., and Upadhyaya, S. A target-centric formal model for insider threat and more. Tech. rep., Technical Report 2004-16, University of Buffalo, US, 2004.
10. Elkhodary, A., and Whittle, J. A survey of approaches to adaptive application security. In *Int. Workshop on Software Engineering for Adaptive and Self-Managing Systems* (2007), pp. 1–16.

11. Gordon Blair, Nelly Bencomo, R. B. F. Models@run.time. In *IEEE Computer* (2009), pp. 22–27.
12. Guo, C. J., Sun, W., Huang, Y., Wang, Z. H., and Gao, B. A framework for native multi-tenancy application development and management. In *E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on* (23-26 July 2007 2007), pp. 551–558.
13. Hafner, M., Memon, M., and Breu, R. Seaas - a reference architecture for security services in soa. *Journal of Universal Computer Science vol. 15* (2009), 2916–2936.
14. Hashii, B., Malabarba, S., Pandey, R., and al, e. Supporting reconfigurable security policies for mobile programs. North-Holland Publishing Co., 2000.
15. Jurjens, J. Towards development of secure systems using umlsec. In *Fundamental Approaches to Software Engineering*, vol. 2029. Springer Berlin Heidelberg, 2001, ch. Lecture Notes in Computer Science, pp. 187–200.
16. Jurjens, J., and Wimmel, G. Formally testing fail-safety of electronic purse protocols. In *Automated Software Engineering, 2001. (ASE 2001). Proceedings. 16th Annual International Conference on* (nov. 2001), pp. 408 – 411.
17. Lamsweerde, A., Brohez, S., and al, e. System goals to intruder anti-goals: Attack generation and resolution for security requirements engineering. In *Proc. of the 3rd Workshop on Requirements for High Assurance Systems* (Monterey, 2003), ACM, pp. 49–56.
18. Liu, L., Yu, E., and Mylopoulos, J. Secure i* : Engineering secure software systems through social analysis. *International Journal of Software and Informatics Vol.3, pp. 89-120* (2009).
19. Lodderstedt, T., B. D., and Doser, J. Secureuml: A uml-based modeling language for model-driven security. In *The 5th International Conference on The Unified Modeling Language* (Dresden, Germany, 2002), vol. 2460, Springer-Verlag, pp. 426–441.
20. Mead, N., and Stehney, T. Security quality requirements engineering (square) methodology. ACM, 2005.
21. Mellado, D., Fernndez-Medina, E., and Piattini, M. Applying a security requirements engineering process. In *Computer Security ESORICS 2006*, D. Gollmann, J. Meier, and A. Sabelfeld, Eds., vol. 4189 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2006, pp. 192–206.
22. Menzel, M., Warschofsky, R., Thomas, I., and Willems, C. Meinel, C. The service security lab: A model-driven platform to compose and explore service security in the cloud. In *Services (SERVICES-1), 2010 6th World Congress on* (5-10 July 2010 2010), pp. 115–122.
23. Mietzner R., Leymann F., P. M. P. Defining composite configurable saas application packages using sca, variability descriptors and multi-tenancy patterns. In *Internet and Web Applications and Services,*

2008. ICIW '08. Third International Conference on (8-13 June 2008 2008), pp. 156–161.

24. Montrieux, L., Jürjens, J., Haley, C. B., Yu, Y., Schobbens, P.-Y., and Toussaint, H. Tool support for code generation from a umlsec property. In *Proceedings of the IEEE/ACM international conference on Automated software engineering* (New York, NY, USA, 2010), ASE '10, ACM, pp. 357–358.
25. Morin, B., Barais, O., Nain, G., and al, e. Taming dynamically adaptive systems using models and aspects. In *IEEE 31st Int. Conf. on Software Engineering* (Vancouver, BC, 2009), IEEE Computer Society, pp. 122–132.
26. Morin, B., Mouelhi, T., Fleurey, F., and al, e. Security-driven model-based dynamic adaptation. ACM, 2010.
27. Mouelhi, T., Fleurey, F., Baudry, B., and et al. A model-based framework for security policy specification, deployment and testing. In *Proceedings of the 11th Int. Conf. on Model Driven Engineering Languages and Systems* (France, 2008), Springer-Verlag.
28. Mouratidis, H., and Giorgini, P. Secure tropos: A security-oriented extension of the tropos methodology. *International Journal of Software Engineering and knowledge Engineering* (2007).
29. Pervez, Z., Lee, S., and Lee, Y.-K. Multi-tenant, secure, load disseminated saas architecture. In *Advanced Communication Technology (ICACT), 2010 The 12th International Conference on* (7-10 Feb. 2010 2010), vol. 1, pp. 214–219.
30. Pervez, Z., Lee, S., and Lee, Y.-K. Multi-tenant, secure, load disseminated saas architecture. In *Proceedings of the 12th international conference on Advanced communication technology* (Gangwon-Do, South Korea, 2010), IEEE Press, pp. 214–219.
31. Rasmus Johansen, Stephan Spangenberg, P. S. Yiihaw .net aspect weaver usage guide.
32. Sanchez-Cid, F., and Mana, A. Serenity pattern-based software development life-cycle. In *19th International Workshop on Database and Expert Systems Application* (2008), pp. 305–309.
33. Scott, K., Kumar, N., Velusamy, S., and al, e. Retargetable and reconfigurable software dynamic translation. IEEE Computer Society, 2003.
34. Sindre, G., and Opdahl, A. Eliciting security requirements with misuse cases. *Requir. Eng. 10*, 1 (2005), 34–44.
35. Thomas Vogel, Andreas Seibel, H. G. The role of models and megamodels at runtime. In *Proceedings of the 2010 international conference on Models in software engineering* (2010), pp. 224–238.
36. Wang, D., Zhang, Y., Zhang, B., and Liu, Y. Research and implementation of a new saas service execution mechanism with multi-tenancy support. In *Proceedings of the 2009 First IEEE International Conference on Information Science and Engineering* (2009), IEEE Computer Society, pp. 336–339.
37. Xin Jin, Ram Krishnan, R. S. A unified attribute-based access control

model covering dac, mac and rbac. In *Proceedings of the 26th Annual IFIP WG 11.3 conference on Data and Applications Security and Privacy* (2012), pp. 41–55.

38. Xu, J., Jinglei, T., Dongjian, H., Linsen, Z., Lin, C., and Fang, N. Research and implementation on access control of management-type saas. In *2010 The 2nd IEEE International Conference on Information Management and Engineering (ICIME)* (16-18 April 2010 2010), pp. 388–392.
39. Zhang, X., Shen, B., Tang, X., and Chen, W. From isolated tenancy hosted application to multi-tenancy: Toward a systematic migration method for web application. In *Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on* (16-18 July 2010 2010), pp. 209–212.
40. Zhong, C., Zhang, J., Xia, Y., and Yu, H. Construction of a trusted saas platform. In *2010 Fifth IEEE International Symposium on Service Oriented System Engineering (SOSE)* (4-5 June 2010 2010), pp. 244–251.

Mohamed Almorsy received his Bachelor and Masters degrees in computer science, Ainshams University, Cairo, Egypt. Mohamed has more than seven years of experience in the software development industry. He began his PhD at Swinburne University of Technology in 2010. Mohameds main focus is cloud computing security management.

John Grundy is Professor of Software Engineering. He is the Head of Computer Science & Software Engineering and Director of Centre for Complex Software Systems and Services Centre Director, Centre for Computing and Engineering Software Systems, Faculty of Information & Communication Technologies, Swinburne University of Technology.

Amani S. Ibrahim received his Bachelor and Masters degrees in computer science, Ainshams University, Cairo, Egypt. Amani began her PhD at Swinburne University of Technology in 2010. Amani focuses mainly on securing the cloud computing infrastructure using vitualization-aware security solutions.