In Proceedings of the 4th International Symposium on Cyberspace Safety and Security (CSS 2012), Melbourne, Australia, Dec 12-13 2012.

MDSE@R: Model-Driven Security Engineering at Runtime

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

Abstract. New security threats arise frequently and impact on enterprise software security requirements. However, most existing security engineering approaches focus on capturing and enforcing security requirements at design time. Many do not address how a system should be adapted to cope with new unanticipated security requirements that arise at runtime. We describe a new approach - Model Driven Security Engineering at Runtime (MDSE@R) - enabling security engineers to dynamically specify and enforce system security requirements based on current needs. We introduce a new domain-specific visual language to model customer security requirements in a given application. Moreover, we introduce a new UML profile to help capturing system architectural characteristics along with security specifications mapped to system entities. Our MDSE@R toolset supports refinement and merger of these visual models and uses model-driven engineering to take the merged model and specify security controls to be enforced on the target system components. A combination of interceptors (via generated configurations) and injected code (using aspect-oriented programming) are used to integrate the specified security controls within the target system. We describe MDSE@R, give an example of using it in securing an ERP system, describe its implementation, and discuss an evaluation of applying MDSE@R on a set of open source applications.

Keywords: Security engineering, model-driven engineering, domain-specific visual languages, aspect-oriented programming

1 Introduction

Security engineering [1] focuses on delivering secure applications that maintain their operations and achieve desired goals even if under attack. Unfortunately both security goals and attacks frequently change over time [16]. Thus security engineering cannot be a one-time process. Software enforced security needs to be revisited and updated whenever new security requirements or challenges arise.

On the other hand, most current security engineering processes are conducted side by side with the system engineering process [2]. This requires having system engineers deeply involved in engineering security of their systems. However, system engineers often lack experience in identifying, and sometimes in protecting against, possible security issues. They may also lack knowledge about customers' security

needs as some potential customers may not even be known at design time. Thus the final product will often be incomplete from a security perspective. Moreover, such systems usually have security hardcoded in their source code either as security realization functions [3, 4] or as security annotation attributes that are translated at runtime into security controls delivered by an underlying security platform [5]. In either case, unanticipated security requirements are not usually considered. Software maintenance is usually required to address such emerging security vulnerabilities and new security requirements raised by customers. Maintenance may take much more time than acceptable where discovered vulnerabilities can be exploited [6, 17]. Moreover, sometimes the system vendor may no longer even exist. Thus post-deployment discovery of security issues or changed application environment security needs are hard to address using existing software security engineering approaches.

Existing security engineering efforts focus on how to identify, capture, and model security objectives and requirements and then how to map such requirements to system entities at design time, for example KAOS, UMLSec, secureUML [3, 11, 12, 13, 15]. These security engineering approaches typically result in systems with fixed and built-in security, limited integration with third party security controls, and very limited flexibility in terms of adaptation and integration with the software operational environment security management systems. Component-based (CBSE) and service-oriented (SOA) security engineering approaches generate security code, using Aspect-oriented Programming (AOP) or WS-Security [7, 9], mostly based on design-time or deployment-time captured security requirements. These approaches benefit from, and are limited to, the underlying system architecture (CBSE, SOA) to deliver flexible and adaptable security. Some adaptive security engineering approaches have been investigated. However most focus on low-level details or is limited to specific security properties – e.g. delivering adaptive access control [10]. These efforts require preparing systems at design time to support runtime adaptation.

We introduce a novel approach, Model Driven Security Engineering at Runtime (MDSE@R), which promotes security engineering from design time to runtime to eliminate problems that arise from the unanticipated requirements. MDSE@R is based on externalizing security from the target system. Thus security to be enforced and critical system entities to be secured can change at runtime to reflect current risks. Moreover, the system does not need to know how security is defined or enforced. At the same time, it still performs normally – e.g. system can use current user-identity to filter data specific to current user without knowing how identity was set.

The software vendor, at design or deployment time, develops a system description model – SDM – using our new UML profile. This model captures system features, components, classes, behaviour, and deployment details. The system SDM is delivered as part of the system delivery package. System customers or the software vendor's security engineers develop, at runtime, a security specification model – SSM – using our new security DSVL. The SSM captures current security goals and objectives, requirements, architecture, and controls that should be used in securing the target software system. Using this set of system and security models (SDMs and SSMs); MDSE@R derives a detailed, merged system security model for the target software systems. This merged system-security model is then used to automatically, dynamically, and at runtime, inject security extensions into target system to achieve the required security capabilities.

Security SSMs are managed and updated at runtime by the software customers, to reflect changes in their operational environment and newly discovered security threats. To support integrating a third party security control with the target system, we introduce a standard security interface with a set of operations e.g. AuthenticateUser, AuthorizeUser, IsAuthenticated, Encrypt, etc. The security control vendor has to develop an adapter that realizes our security interface. Thus MDSE@R platform can easily consume such controls at runtime. We have validated our approach on eight significant open source applications. We conducted a performance evaluation and preliminary user evaluation.

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

2 Motivation

Consider SwinSoft, a software company that is building a large web-based ERP system - “Galactic”. Galactic provides customer management, order management, and employee management modules. SwinSoft targets different markets in different countries for Galactic. However, such markets, domains and likely customers have different regulations and information security standards that must be satisfied. Galactic must integrate with diverse customers’ existing security solutions and other application security. Moreover, SwinSoft has found that customers' security requirements that Galactic must meet may change dramatically over time.

Swinburne University, a new prospect from the education sector, wants to purchase a new ERP solution in order to improve its internal enterprise management processes. Swinburne has special security requirements because it is ISO27000 certified. Its enterprise security architects conduct periodic risk assessment. This may result in a requirement to reconfigure the deployed applications’ security to block newly discovered threats. Swinburne also wants to have its ERP system security flexible enough as it is planning to integrate it with the existing and future partners. This implies that Galactic security will change over time after the deployment.

At the same time, SwinMarket, a big brand supermarket chain, has decided to purchase Galactic. SwinMarket also has a need for highly customizable security options on different system features that Galactic must satisfy. SwinMarket expects security solutions deployed in its application operational environment to change over time. Galactic must support quick realization of security updates. Any delay in patching newly discovered vulnerabilities means a loss of money. SwinSoft tried both existing traditional security engineering and Software Product Lines approaches to deliver an efficient customized security. However, these approaches could not help to deliver adaptable security within an acceptable cost and time.

3. Our Approach

The MDSE@R approach is based on two key concepts: (i) externalizing security management and enforcement from the system to be secured while being able to intercept calls to any arbitrary critical system entity at runtime. Thus the system does not need to be overwhelmed with how security is defined, enforced, or modified; and (ii) Model-Driven Engineering (MDE), using DSVL models to describe system and security properties at different levels of abstraction. Figure 1 shows an overview, and steps, of how to apply MDSE@R in runtime system security engineering:

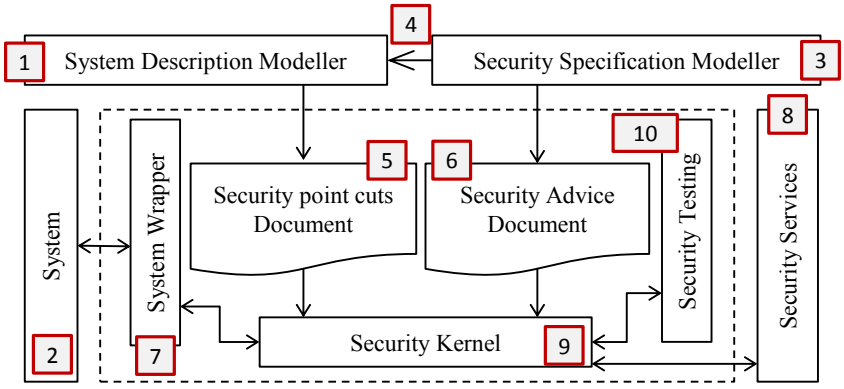

Figure 1. Overview of MDSE@R dynamic security engineering approach

1. Build System Description Model (SDM): A detailed system description model (Fig.1-1) is delivered by the system provider. The SDM, an example is shown in Figure 2, describes various details of the target system. Our system description model covers: system features (using use case diagrams), system architecture (using component diagrams), system classes (using class diagrams), system behaviour (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. It depends on the system vendor and the customer agreements. Customer security engineers may need to specify security on system entities (using system components and/or classes models), on system status (using system behaviour model), on hosting nodes (using system deployment model), or on external system interactions (using system context model). Moreover, they may specify their security requirements on coarse-grained level (using system features and components models), or on fine-grained ones (using system class diagrams). The system SDMs can be synchronized with the running instance using models@runtime synchronization techniques, or manually by the system vendor. Some of the system description detail, specifically the system class diagram, can be reverse-engineered, if not available, from the target system (Fig.1-2). We developed a new UML profile (Fig 2-A) to extend UML models with architectural and security stereotypes 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 entities (requirements, controls, etc.) mapped to a system entity, see step 3.

2. Build Security Specification Model (SSM): A set of models developed and managed by customer security engineers (Fig.1-3) to specify the current security needs that must be satisfied in the target system, an example is shown in Figure 3. 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 security controls model is mandatory. It is used in generating security realization and integration code.

3. Weave System and Security Models: A many-to-many mapping between the system description model (SDM) entities and security specification model (SSM) entitles is managed by the customer security engineers (Fig.1-4). One or more security entities (security objective, requirement and/or control) are mapped to one or more system model entity (feature, component, class and/or method). Mapping a security concept on an abstract system entity – e.g. system feature - implies a delegation of the same security concept to the concrete entities – e.g. the feature realization classes and methods. This is achieved using our UML profile (Fig2-A). Moreover, mapping an abstract security concept – e.g. security objective – to a system entity – e.g. system class – implies mapping all security requirements, services, and controls that realize this security objective to that class.

***4. Enforce specified security on target system ent*ities:** In the previous steps, both security details and critical system entities emerge at runtime. MDSE@R automates the realization of the specified security on the critical system entities without any involvement from the security or system engineers. Whenever a mapping is defined\updated between an SSM entity and an SDM entity, the underlying MDSE@R platform propagates these changes as follows: (i) Update the Live System Interceptors' Document (Fig.1-5) which maintains a list of *critical entities* (system entities that have mapped security entities) where security controls should be weaved or integrated; (ii) Update the Live Security Specification Document (Fig.1-6) – which maintains the list of security controls to be applied at every critical entity; (iii) Update the System Container (Fig. 1-7) - the container is responsible for intercepting system calls to critical system entities at runtime and redirecting them to a default handler.

5. Test System-Security Integration: Before putting modifications online (activating the specified security adaptations), MDSE@R verifies that the target system is correctly enforcing the specified security level. We do not have to test the security control itself, however we need to make sure that the right security control is correctly integrated within the right system entity as specified. The MDSE@R security testing component (Fig.1-8) generates and fires a set of security integration scripts (test cases). These test cases simulate requests to system entities that have security specifications and compare the security context after calls (actual results) with the expected results – e.g. user identity is correctly set, permissions are set as specified, etc. A log of the test cases' firing results is generated to the security engineers showing the test cases and their status pass/fail.

6. Security Services (Fig.1-10) - In MDSE@R, we need to be independent of any security platform (java, spring, Microsft WIF, etc.) or security mechanism and support easy integration of third-party security controls selected or specified by the customer. We define a *standard security interface* for every security attribute (authentication, authorization…). This interface specifies parameters expected by

each security control, based on security function, in order to perform its task – 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 integrate with MDSE@R. This helps security vendors develop one adaptor for all systems.

***7. Security Enforcement Point -* SEP** - (Fig.1-9) – this 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 security interface) to the designated security controls' clients. Moreover, 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 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. Usage Example

Here we demonstrate how the system vendor and customers can use our MDSE@R, and the provided platform toolsets, in engineering security of their system at runtime. Moreover, we highlight the involved stakeholders and their responsibilities, and the expected outcomes of every step. We use the motivating example from Section 2, Galactic system developed by SwinSoft and procured by Swinburne and SwinMarket. The two customers have their own security requirements to be enforced on their Galactic ERP application instances. We illustrate our usage example using screen dumps from our toolset.

1. Model Galactic System Description

This task is done during or after the system is developed. SwinSoft, the service vendor, decides the level of application details to provide to its customers in the Galactic SDM. Figure 2 shows that SwinSoft provides its customers with description of system features including customer, employee and order management features (Fig. 2-b), system architecture including presentation, business and data access layers (Fig. 2-c), system classes including CustomerBLL, OrderBLL, EmployeeBLL (Fig.2-d), and system deployment including web server, application server, and data access server (Fig. 2-e). SwinSoft uses our UML profile (Fig. 2-a) to capture dependences and relations between system features and components, and components and classes.

2. Model Swinburne Security Needs

This task is conducted by Swinburne security engineers during their security management process to define and refine organizational security needs starting from security objectives down to realization security controls. This model should be repeatedly revisited to incorporate any emerging changes in Swinburne security objectives. In this scenario, Swinburne engineers document Swinburne security objectives that must be satisfied by Galactic (Fig. 3-a). 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 (Fig. 3-b).

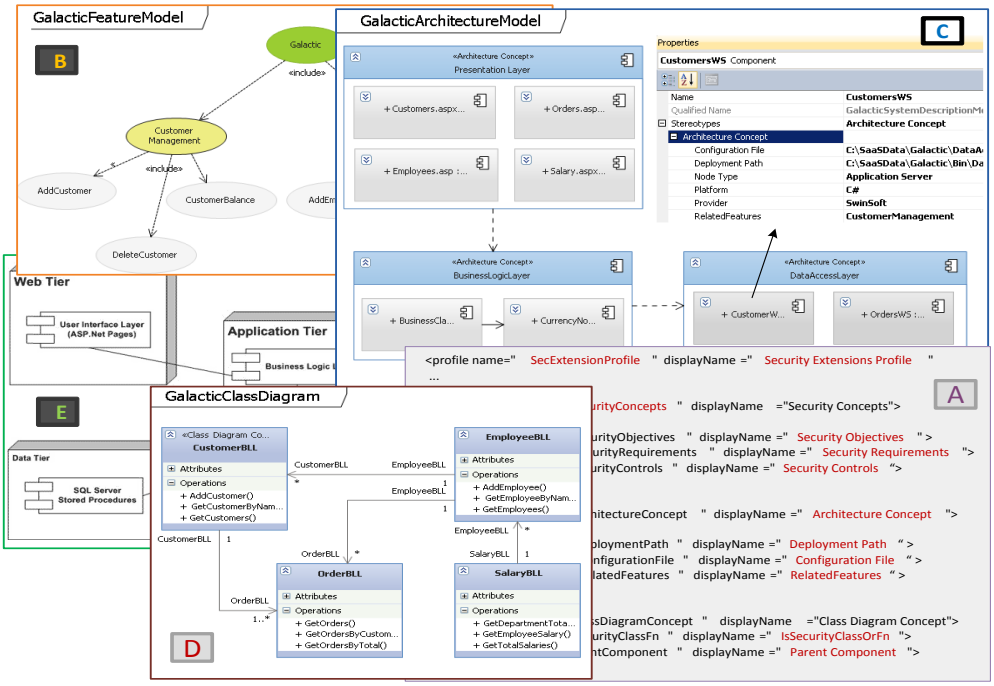

Figure 2. Examples of the Galactic system description model

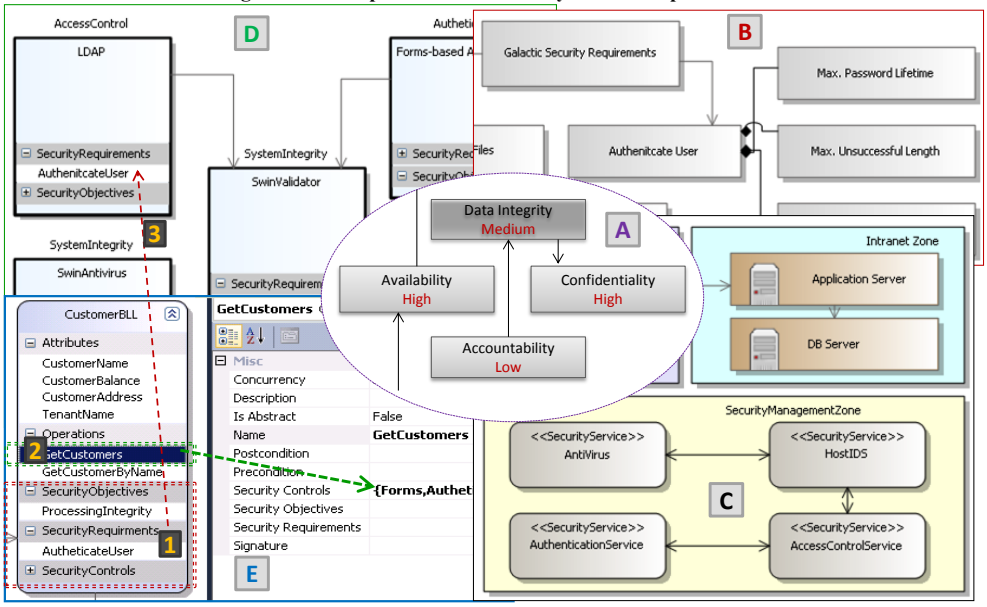

Figure 3. Examples of Swinburne security specification model

This model keeps track of the security requirements and their link 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. Swinburne security engineers next develop a detailed security architecture including services and security mechanisms to be used in securing Galactic (Fig. 3-c). In this example we show the different security zones (big boxes) that cover Swinburne 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. Swinburne security engineers finally specify the security controls (i.e. the real implementations) for the security services modelled in the security architecture model (Fig. 3-d). This includes SwinIPS Host Intrusion Prevention System, LDAP access control and SwinAntivirus. Each security control entity defined in the security controls model specifies its family (authentication, authorization, audit, etc.) and the deployment path of its adaptor. Each security specification model maintains traceability information to parent model entities. In figure 3-d, we specify 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.

3. Weave System SDM and Security SSM

After developing the system SDMs – done by SwinSoft, and the security SSMs – done by Swinburne security engineers, the Swinburne security engineers can 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 in our toolset. Any system feature, structure or behaviour can dynamically and at runtime reflect different levels of security based on the currently mapped security attributes on it. Figure 3-e shows a part of Galactic class diagram where CustomerBLL, a UML class entity, is extended with security objectives, requirements and controls compartments. In this example the security engineers have specified AuthenticateUser as one of the security requirement to be enforced on the CustomerBLL class (1). Such a requirement is achieved indirectly using LDAP control (3). Moreover, they have specified Forms-based authentication on the GetCustomers method (2). This means that a request to a method in the CustomerBLL class will be authenticated by the caller's Windows identity (LDAP), but a request to the GetCustomers method will be authenticated with a Forms-based identity. MDSE@R uses the security attributes mapped to system entities to generate the full set of methods' call interceptors and entities' required security controls, as in Fig. 4-1, 2.

4. Galactic Security Testing

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 test cases for each method listed in the interception document. The generated test case contains a set of security assertions (one for each security property specified on a given system entity). Security engineers should check the security test cases firing log to verify that no errors introduced during the security controls integration with Galactic entities.

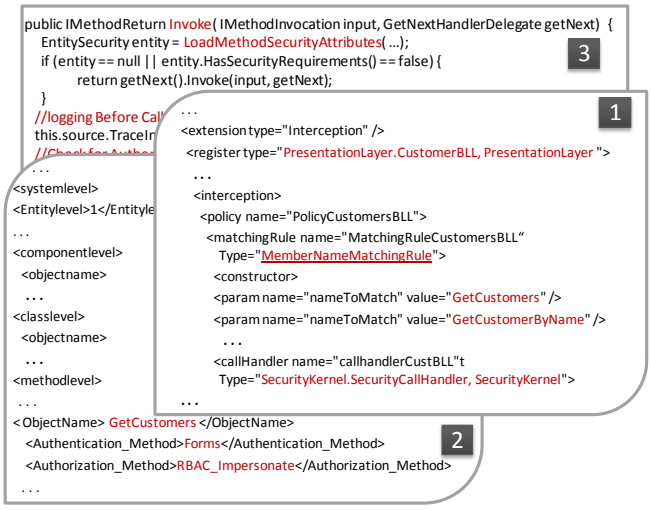

Figure 4. Examples of the interceptors and security specification files

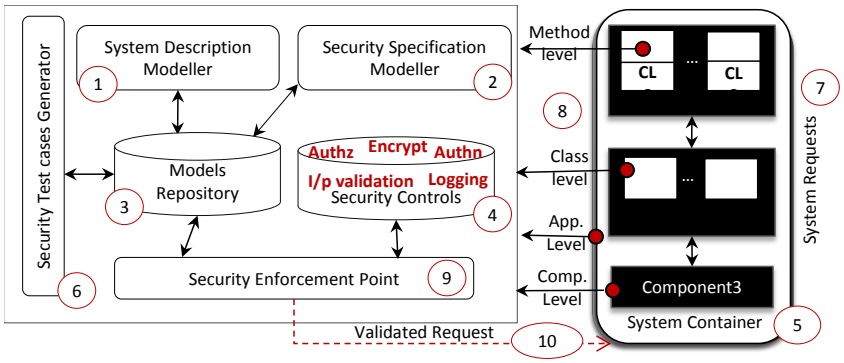

Figure 5. MDSE@R architecture, tools and sequence of operation

5. ARCHITECTURE AND IMPLEMENTATION

The architecture of the MDSE@R platform, shown in Figure 5, consists of:

- System description modeller (1) is developed as an extension of Microsoft VS2010 modeller with an UML profile (Fig. 2-A) to enable system engineers modelling their systems' details with different perspectives including system features, components, deployment, and classes. The UML profile defines stereotypes and attributes to maintain the track back and foreword relations between entities from different models. Moreover, a set of security attributes to maintain the security concepts (objectives, requirements and controls) mapped to every system entity.

- Security specification modeller (2) is developed as a DSVL using Microsoft VS2010 plug-in. It enables application customers, represented by their security engineers, to specify the security attributes and capabilities that must be enforced on the system and/or its operational environment. The security modeller delivers a set of

complete security DSVLs. This DSVL captures customer's security objectives and relationships between them, security requirements and relationships (including composition and referencing relations), security architecture and design (including security zones, security services, mechanisms to be deployed in each zone), security controls details and relationships to corresponding security requirements.

***- Models Repository*:** Both modelling tools use a repository (3) 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 4. 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 & 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: is a library of available and registered security patterns and controls. It can be extended by the system 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. 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 as our security controls database. It provides a set of authentication, authorization, encryption, etc. controls that we used in our testing.

- System Container: 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 (Fig. 5-5). 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 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, 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).

- Security Test Case Generator (6) uses the NUnit testing framework 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 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.

- Security Enforcement Point (8) 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 (9).

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

Benchmark Applications		Statistics			Security Attributes				
		KLOC	Files	Classes	Authn	Authz.	I/P Valid.	Audit	Crypto.
G1	Galactic	16.2	99	101	F, C, S, M				
	PetShop	7.8	15	25	F, C, S, M				
G2	Splendid	245	816	6177	C, S, M				(C, S, M)*
	KOOBOO	112	1178	7851	C, S, M				(C, S, M)*
	NopComm	442	3781	5127	C, S, M				(C, S, M)*
	BlogEngine	25.7	151	258	C, S, M				(C, S, M)*
	BugTracer	10	19	298	C, S, M				(C, S, M)*
	TinyERP	6	20	22	C, S, M				(C, S, M)*

F: Security attribute successfully applied on feature level & propagated to lower entities
C:Security attribute successfully applied on component level & propagated to lower entities
S:Security attribute succesfully applied on classes **M**:Security attribute can be applied on method level

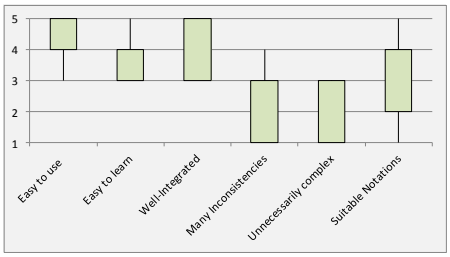

Figure 6. MDSE@R usability evaluation

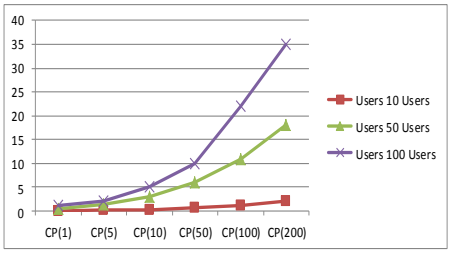

Figure 7. Average performance overhead of MDSE@R

6. Evaluation

In our evaluation of MDSE@R we target assessing the approach capabilities and scalability in (1) capturing descriptions of different real systems; (2) capturing different security details; (3) propagating security attributes on different system entities (features, components, etc); and (4) enforcing unanticipated security including authentication, authorization, auditing, etc. at runtime within a reasonable time.

Benchmark Applications: We used a set of eight real, different domains, and open source .NET applications in validating our approach capabilities. We divided the evaluation set into two groups: Group-1 has two applications: Galactic-ERP that we developed from scratch, and PetShop e-Commerce application, an open source application. Both applications have been modified to adopt the Unity application block as the system container. Group-2 has six third-party web applications: SplendidCRM, KOOBOO, BlogEngine, BugTracer, NopCommerce, and TinyERP. For this group we use Yiihaw framework as the system container. Table 1 shows some statistics about the selected applications including KLOC, files, and classes.

Experimental Setup: Using MDSE@R, we developed one security specification model (SSM) including security objectives, requirements, and controls as in Fig. 3. We specified security requirements and controls for authentication, authorization, input validation, logging and cryptography. 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.

Evaluation Results: We now have the required models to manage the various applications' security. Table 1 shows security attributes that MDSE@R succeeds to *capture,* at runtime, including authentication, authorization, input sanitization, audit and cryptography. This represents most of known security attributes. Table1 also shows that MSDE@R succeeds 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). However, Group-2 applications 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) as it requires both the caller and callee to use parameters of type *string*.

User Evaluation: We have carried out a user evaluation of our tools and platform to assess MDSE@R usability. We had seven post-graduate researchers, not involved in the development of the approach, to use our developed tools and platform. We had them explore several MDSE@R system and security DSVL specifications of the PetShop and Galactic applications. They then modified the security requirements and updated systems security specification at run-time. We conducted a usability survey to gain there feedback on our DSVL models, modelling tools, and security enforcement platform. The results, Figure 6 (1: strongly disagree, 5: strongly agree), show that they successfully understood and updated security models of the target systems. They recommend using expressive icons in security DSVL instead of boxes.

Performance Evaluation: We have tested the performance overhead of MDSE@R on a desktop PC with core2 Duo processor and 4GB memory. The results, as shown in Figure 7, shows that the performance overhead range from 0.13 up to 35 msec for 200 critical point and 100 concurrent users. This overhead equals time spent by SEP to query the security requirements to enforce on the given point, initiate and call the security controls specified. Moreover, we have measured the adaptation delay incurred by MDSE@R to realize a single simple mapping between a security entity and a system entity. This overhead equals *avg. 2sec*. This represents time to update the interceptors and security specification documents and time to generate and fire the required integration test case(s).

7. Discussion

Our MDSE@R approach promotes 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 systems to support adapting security at runtime. Moreover, a key benefit reaped from our approach is the support of model-based security management. Customer security requirements, architecture and controls are maintained and enforced through set of centralized SSMs instead of low level scattered configurations and code that lack consistency and is difficult to modify. In our evaluation we developed one security model and used it with different systems.

Each system enforces the security mapped to its entities. Moreover, 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.

The security engineering of *existing* systems (extending system security capabilities) has three possible scenarios: (i) systems that already have their SDMs, we can use MDSE@R directly to specify and enforce security at runtime; (ii) systems without SDMs, we reverse engineer parts of system models using MDSE@R. Then we can use MDSE@R to engineer required system security. Finally, systems with built-in security, in this case we can use MDSE@R to add new security capabilities only. MDSE@R cannot help with modifying or disabling existing security. We are currently working on an extension of MDSE@R to support deletion of existing security methods and partial code using modified AOP techniques.

The selection of the level of details to apply security on depends on the criticality of the system. In some situations like web applications, we may intercept calls to the presentation layer only 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. We plan to extend our generated test cases to include performance tests in the near future, allowing MDSE@R to help optimizing the level of security enforced.

MDSE@R helps in engineering security into systems at runtime, while the controls' configuration and administration should be managed by security administrators. Moreover MDSE@R does not support defining business rules at runtime – e.g. employee should not be able to retrieve customers' records of type VIP. The target system should have this rule while MDSE@R will provide the current user roles/permissions as returned by the customer security control.

Customers who do not want extra security control or do not have enough staff can use security delivered by software vendors using MDSE@R. Then, they deliver both SDM and SSM to customers. This helps in further customizations whenever needed.

8. Related Work

1. Design time Security Engineering: Software security engineering aims to develop secure systems that remain dependable in face of attacks [1]. Security engineering activities include: identifying security objectives; identifying security risks that threaten system operation; elicitation of security requirements that should be enforced; developing security architectures and designs that deliver the security requirements; and developing, deploying security controls.

- ***Early-stage security engineering*** approaches focus only on security requirements elicitation and capturing at design time. KAOS [11] was extended to capture security requirements in terms of obstacles to stakeholder's goals. Obstacles are defined in

terms of conditions that when satisfied will prevent certain goals from being achieved. Secure i* [12] focuses on identifying security requirements through analysing relationships between users, attackers, and agents of both parties. Secure Tropos [13] 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. 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. Misuse cases [14] capture use cases that the system should not allow and may harm the system operation or security. UMLSec [3] introduce a UML profile that provides stereotypes to be used in annotating design elements with security intentions and requirements. UMLSec provides a comprehensive UML profile but it was developed mainly for use during the design phase. UMLSec has stereotypes for predefined security requirements only (secrecy, secure dependency, critical…). SecureUML [15] provides a meta-model to capture RBAC policies of target systems. Both approaches are tightly coupled with system models. Moreover, they focus on mapping security requirements on system design rather than developing security architecture, design and implementation details.

2. Adaptive Application Security: Several research efforts try to deliver adaptable security capabilities. Serenity [5] enables provisioning of appropriate security and dependability mechanisms for Ambient Intelligence systems at runtime. Security attributes are specified at design time. At runtime, the framework links Serenity-aware systems to appropriate security patterns. Serenity does not support capturing new unanticipated security requirements. Morin et al [10] propose a security-driven and model-based dynamic adaptation approach to help applications reflecting defined context-aware access control policies. Engineers define security policies that take into consideration context information. Whenever the system context changes, the proposed approach updates system architecture to enforce matched security policies. Mouelhi et al [7] 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 write integration code or to use specific platform or architecture). Moreover, they support limited security objectives such as access control. Unanticipated security requirements are not supported. No validation that target systems correctly enforce specified security.

9. Summary

We introduce MDSE@R as a new model-driven approach to dynamically engineer security for software applications at runtime. MDSE@R is based on using a set of multi-level system description models, developed by system providers, to describe different application perspectives; a set of security specification models, developed by

system customers, to capture security details using DSVL. MDSE@R then bridges the gap between these two specifications through merger of both system and security models into a joint system-security model. MDSE@R uses dynamic injection of security enforcement interceptors and code into the target system to enforce the security specified. Security specifications are thus externalized and loosely coupled with system specifications, enabling both the system and security specification to evolve. We have developed a modelling toolset and a prototype of MDSE@R. We evaluated MDSE@R on a set of eight applications. We conducted user evaluation and performance evaluation of the MDSE@R platform, and adaptation overhead.

10 References

[1] R. Anderson, "What is Security Engineering?," in *Security Engineering: A Guide to Building Dependable Distributed Systems*, 2nd ed. Indianapolis, Indiana: Wiley and Sons, 2001, pp. 3-12.

[2] T. Phan, J. Han, et al, "SOABSE: An approach to realizing business-oriented security requirements with Web Service security policies," in *Proc. Int. Conf. on Service-Oriented Computing and Applications*, Taipei, Taiwan, 2009, pp. 1-10.

[3] J. Jürjens, "Towards Development of Secure Systems Using UMLsec," in *Proc. 4th Int. Conf. on Fundamental Approaches to Software Engineering*, London, UK, 2001, pp. 187-200.

[4] H. Mouratidis and J. Jurjens, "From goal-driven security requirements engineering to secure design," *Int. Journal of Intelligent Systems,* vol. 25, 2010, pp. 813-840.

[5] F. Sanchez-Cid and A. Mana, "SERENITY Pattern-Based Software Development Life-Cycle," in *Proc. 19th Int. Workshop on Database and Expert Systems Application*, Italy, 2008, pp. 305-309.

[6] M. Almorsy, J. Grundy, A.S. Ibrahim, "VAM-aaS: Online Cloud Services Security Vulnerability Analysis and Mitigation-as-a-Service", in *Proc. 13th Int. Conf. on Web Information System Engineering*, Nov 28-30 2012, Paphos, Cyprus, Springer.

[7] B. Morin, O. Barais, et al, "Taming Dynamically Adaptive Systems using models and aspects," in *Proc. 31st IEEE Int. Conf. on Software Engineering*, Vancouver, BC, 2009, pp. 122-132.

[8] T. Mouelhi, F. Fleurey, et al, "A Model-Based Framework for Security Policy Specification, Deployment and Testing," in *Proc. 11th Int. Conf. on Model Driven Engineering Languages and Systems*, France, 2008, pp. 537-552.

[9] M. Hafner, M. Memon, et al, "SeAAS - A Reference Architecture for Security Services in SOA," *Journal of Universal Computer Science,* vol.15, 2009, pp. 2916-2936.

[10] B. Morin, T. Mouelhi, et al, "Security-driven model-based dynamic adaptation," in *Proc. 25th Int. Conf. on Automated software engineering*, Belgium, 2010, pp. 205-214.

[11] A. Lamsweerde, S. Brohez, et al, "System Goals to Intruder Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering," in *Proc. RE'03 Workshop on Requirements for High Assurance Systems*, Monterey, 2003, pp. 49-56.

[12] L. Liu, S. Eric, et al, "Secure ¡*: Engineering Secure Software Systems through Social Analysis," Int. Journal of Software and Informatics, vol. Vol.3, pp. 89-120, 2009.

[13] H. Mouratidis and P. Giorgini, "Secure Tropos: A security-oriented Extension of the Tropos Methodology," *Int. Journal of SW Eng. and knowledge Engineering,* vol. 17, 2007, pp 285-309.

[14] G. Sindre and A. Opdahl, "Eliciting security requirements with misuse cases," *Journal RE,* vol. 10, 2005, pp. 34-44.

[15] T. Lodderstedt, D. Basin, et al, "SecureUML: A UML-Based Modeling Language for Model-Driven Security," in *Proc. 5th Int. Conf. of Unified Modeling Language*, Dresden, 2002, pp. 426-441.

[16] M. Almorsy, J. Grundy, A.S. Ibrahim, "Supporting Automated Vulnerability Analysis using Formalized Vulnerability Signatures", in *Proc. 27th Int. Conf. on Automated Software Engineering*, 2012, Essen, Germany.

[17] M. Almorsy, J. Grundy, A.S. Ibrahim, "Supporting Automated Software Re-Engineering Using "Re-Aspects", in *Proc. 27th Int. Conf. on Automated Software Engineering*, 2012, Essen, Germany.