2012 IEEE Fifth International Conference on Cloud Computing

TOSSMA: A Tenant-Oriented SaaS Security Management Architecture

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 - **Multi-tenancy helps service providers to save costs, improve resource utilization, and reduce service customization and maintenance time by sharing of resources and services. On the other hand, supporting multi-tenancy adds more complexity to the shared application's required capabilities. Security is a key requirement that must be addressed when engineering new SaaS applications or when re-engineering existing applications to support multi-tenancy. Traditional security (re)engineering approaches do not fit with the multi-tenancy application model where tenants and their security requirements emerge after the system was first developed. Enabling, runtime, adaptable and tenant-oriented application security customization on single service instance is a key challenging security goal in multi-tenant application engineering. In this paper we introduce TOSSMA, a Tenant-Oriented SaaS Security Management Architecture. TOSSMA allows service providers to enable their tenants in defining, customizing and enforcing their security requirements without having to go back to application developers for maintenance or security customizations. TOSSMA supports security management for both new and existing systems. Service providers are not required to write security integration code to use a specific security platform or mechanism. In this paper, we describe details of our approach and architecture, our prototype implementation of TOSSMA, give a usage example of securing a multi-tenant SaaS, and discuss our evaluation experiments of TOSSMA.**

Keywords: Cloud computing, cloud computing security, multi-tenancy security, SaaS application security

I. INTRODUCTION

Cloud computing is a new paradigm shift in computing platforms that delivers a new generation of internet-based, highly scalable distributed computing platforms [1]. Software-as-a-Service (SaaS) is one of the three key service delivery models delivered by the cloud computing model [2]. The cloud model, including SaaS, is based on two key characteristics: *multi-tenancy*, where multiple tenants share the same service instance, and *elasticity*, where tenants can scale the amount of their allocated resources based on current demands. Although both characteristics target improving resource utilization, cost and service availability, these gains are threatened by multi-tenancy security implications. Sharing applications that process critical information with different tenants without sufficient proven security isolation, security SLAs or tenant control, results in "loss-of-control" and "lack-of-trust" problems.

Existing multi-tenant SaaS engineering approaches focus on how to deliver configurable rather than customizable business features [3]. This simplifies the capture of new tenants' requirements in terms of system configurations that are loaded based on the current user. However, such software engineering approaches depend on traditional security engineering techniques that focus on design-time capture, design and implementation of security. The resultant SaaS applications provide security with built- in, hard-coded security controls. The delivered security is thus limited and may turn out to be far from eventual cloud consumers' needs that often emerge after application deployment. Moreover, Consumers have no control on the security of their assets. This exacerbates the "loss of control" problem from the customer perspective.

Current research efforts in securing multi-tenant SaaS applications focus mainly on: (re)engineering of multi-tenant SaaS applications to extend their security isolation capabilities [4, 5]; maintaining isolation between different tenants' data at rest, at processing and/or at transmission [6-8]; and developing security controls and architectures to deliver SaaS application security functions e.g. access control, taking into account the multi-tenancy dimension [9, 10]. Such efforts lead to built-in security. Tackling loss-of-control problem, enforcing tenants' security requirements rather than the service provider security requirements, and integrating SaaS applications with tenants' security infrastructure are not addressed.

In this paper we introduce TOSSMA, tenant-oriented security management architecture for multi-tenant SaaS applications. TOSSMA is based on our new SaaS security model "Tenant-Oriented Security" where a given service can capture and enforce different sets of security requirements at runtime based on its tenants. This overcomes the existing classic model "Service-Oriented Security" where a given service enforces one set of security requirements usually captured and developed by the service provider. TOSSMA enables every tenant of a given SaaS application to specify, enforce, and monitor the security of their cloud hosted assets. Moreover, it enables SaaS application providers to manage security isolation between their service tenants. TOSSMA is based on instrumenting the application with a general security wrapper at design time, using the inversion of control design pattern, or after development time, using dynamic-weaving Aspect-Oriented Programming (AOP).

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.146

This enables TOSSMA to intercept calls to any arbitrary application resource. Based on every tenant's defined security requirements, TOSSMA generates a set of critical system entry points that should be intercepted, at runtime using the system security wrapper, to enforce the tenant specified security. The security requirements details vary from high-level security objectives (system should authenticate, authorize, encrypt, digital sign, etc.) to security controls to be used (system should use CA identity manager, McAfee antivirus, Microsoft WIF, etc.). Whenever a request is received for a critical system resource (critical resources differ from tenant to another), TOSSMA enforces security controls specified by the current requesting tenant. This may be achieved by utilizing external security solutions provided by the cloud platform provider, the service provider or the service consumer (tenant). TOSSMA can easily integrate with third-party security controls using a predefined standard security interface that is used to communicate with security controls using predefined APIs signatures implemented by security controls' vendors.

Our approach has three main contributions in the area of the security management of multi-tenant SaaS applications. TOSSMA enables SaaS applications to satisfy new security requirements at runtime defined by the SaaS application tenants. TOSSMA enables SaaS applications to easily integrate with the available security controls deployed on the hosting cloud computing platform without modifying the target application. This enables SaaS providers to focus on application functionality and not on application security engineering. TOSSMA mitigates the loss-of-control security problem raised by cloud consumers when adopting the SaaS model by our proposed tenant-oriented security model. TOSSMA provides a security management console that enables tenants to specify and revise their own security requirements based on their internal security management process. TOSSMA can addresses the security isolation problem by enabling service providers to inject authorization security controls that validate and authorize users' inputs at system critical entry points - *e.g.* users of tenant T1 cannot send requests to a system resource with malicious inputs to read tenant T2 data.

Section II presents a motivating example of this research and overview on multi-tenancy and its impact on SaaS applications. Section III discusses our approach. Section IV gives details of TOSSMA architecture. Section V goes through a usage example of the developed architecture. Section VI discusses TOSSMA implementation details. Section VII discusses our experimental evaluation results, key implications and further research. Section VIII reviews related work in SaaS application security engineering.

II. BACKGROUND

A. Motivating 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". SwinSoft hosts Galactic on a cloud platform delivered by "GreenCloud" (GC). GC delivers a PaaS service delivery model with a set of business functions. SwinSoft depends on third party services, delivered by GC, to deliver better functionality to its customers. SwinSoft uses the following services: *Currency-Now service* to get up-to-date currency exchange rates; and *Batch-MPRD* to conduct transactions' posting using the map-reduce model that improves and paralyzes the batch posting operations.

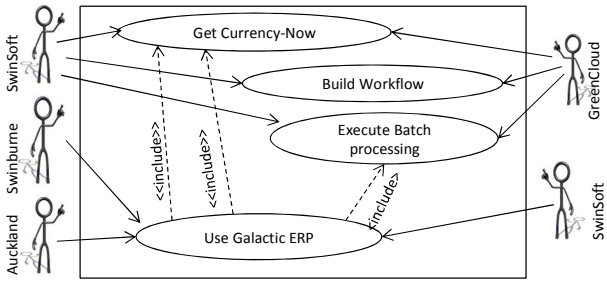

Figure 1: A use case diagram for our motivating example

"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 the Galactic ERP solution. This is to save upfront investment required and keep infrastructure costs optimized. At the same time, "Auckland University" has also decided to purchase the Galactic ERP system.

However, each of these Galactic service consumers has their own quite different security objectives. Swinburne 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 so 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, use of a two-factor authentication system, transaction accountability and auditability, and all data must be kept confidential. Both organizations thus would like to use the multi-tenant Galactic service while modeling and enforcing different security requirements and integrating with different security services.

B. SaaS Applications and Multi-tenancy

A SaaS application may be hosted on top of PaaS, IaaS or directly hosted on cloud platform infrastructure. Although this gives flexibility in applications deployment, it complicates the development of a complete security and threat model [1]. Moreover, the SaaS threat model differs from one cloud platform to another based on the cloud platform architecture and security solutions employed [2].

Multi-tenancy implies sharing of computational resources, storage, services, and applications between the

cloud platform tenants. Adopting multi-tenancy with SaaS results in a set of requirements that must be addressed by the SaaS application. We have identified two key requirements in the area of SaaS applications' security engineering. The first one is the security isolation among tenants' assets at rest (storage), during processing (in memory), and during transient (among application components or between the application and the tenant site). *Second*, it is required to support enforcement of different security requirements on the same service instance at runtime. Application customization approaches do not fit well with runtime and multi-tenant specification and security enforcement. These security requirements may change over time as new risks emerge.

A multi-tenant SaaS security architecture that addresses these challenges should allow tenants to define their security requirements and change them over time based on their risk management process and new security objectives. It should allow each tenant to enforce their security requirements independent from other tenants' requirements. The enforcement of these security requirements at runtime should not require redeveloping or customizing application instance(s) for existing or new tenants. It should support the integration of the target SaaS application with third party security controls. It should support weaving security controls at any application entry point.

III. OUR APPROACH

TOSSMA is based on externalizing security realization code and management activities from the target application. This includes defining security, integrating and enforcing security controls, and monitoring security of the target SaaS application. At the same time, we update the thread security context with security controls returned results – e.g. user identity information. Thus SaaS applications avoid being overwhelmed with security implementation details. Moreover, avoid built-in, hard-coded security controls and thus enforced security can be changed at runtime without reengineering the SaaS application.

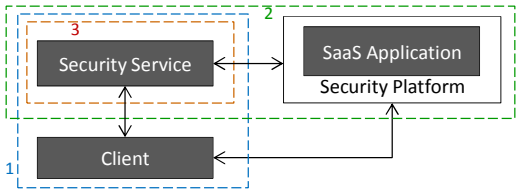

Figure 2: Client, SaaS, Security Platform, and Security services possible interactions, deployments and ownerships

Externalizing security requires being able to inject security functions/controls into system functions at runtime and at any arbitrary system entry point specified by the SaaS application provider or consumers. This should be supported for existing and new applications. Using web server httpModules is limited to application URLs. Moreover, they will not help with back end components (application tier and database tier). Thus, we adopt aspect-oriented programming model and dependency injection design pattern to inject security dynamically into system functions at runtime.

To support seamless integration of third-party security controls (specified by the service providers or consumers) with the target system without modifying the target application, we developed a standard security interface that defines a set of standard security operations along with the expected parameters for each operation – e.g. AutheticateUser, IsAuthenitcated, AuthorizeUser, Encrypt, decrypt, etc. A security vendor has to develop an adaptor (interface) that we use when integrating the application with the target security control, at runtime. The standard interface helps security vendors to develop one adaptor for all applications.

Figure 2 shows that a security service can be delivered by the SaaS service provider (2); the cloud platform provider (3); a security vendor selected by the service client (1). The interactions between a client, the SaaS application, and the security services should proceed as follows: the client makes a request to a resource, and then the security platform intercepts the request and enforces the defined security including authentication, authorization, logging, input validation, etc. The security services then interact with the client and our security platform to perform the defined security operations. Finally, the security platform either proceeds with the request (after setting the security context of the thread) or deny the request. *Thus the SaaS does not need to know how the request has been authenticated, authorized, etc.* Below we discuss a set of key possible security attributes to show how our approach can satisfy these requirements taking into consideration the challenges discussed in Section II.

Identity Management: user identity is a set of information that discriminates between different users. Identity could be managed by every IT system individually or centrally using an identity management system. The later scenario requires developing a connector for every IT system to be integrated with the identity provider. Adopting our approach helps standardizing the connector between both entities, as we have one standard interface defined by the security platform.

Authentication: authentication modes may be claim-based or classic authentication. In classic authentication, applications expect their clients to enter their identity information – e.g. username and password. This means that the authentication security controls are built-in. In claim-based authentication, applications expect security tokens issued by a trusted party that authenticated the user at early stage. This token contains a set of claims about the user identity, roles and other information. Our security platform can be integrated with classic authentication controls or claim-based authentication service. The security platform passes requests to the tenant authentication service. Once the authentication service returns, we use the returned information – e.g. user identity and claims - to set the thread security information user identity and roles.

Authorization: authorization requires details about the requested resource, the requesting user and his assigned permissions, etc. The authorization control checks if the user is authorized to access the requested resource. The security platform intercepts requests and generates authorization request with user identity, requested resource and action to the tenant selected authorization control.

Logging: logging has different levels of verbosity. Moreover, the details used in a logging transaction may differ as well including current user, timestamp, resource, action,

parameters, etc. The security platform sends log requests to tenant logging control including with required information.

Cryptography: confidentiality of data at rest, transmission and processing can be achieved using cryptography techniques including cryptographic algorithm and key management. The responsibility of the security platform is to intercept requests and initiate requests to the tenant encryptor or decryptor based on the defined security requirements.

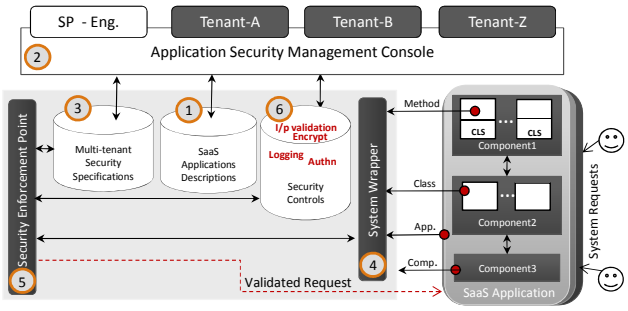

Figure 3: TOSSMA architecture

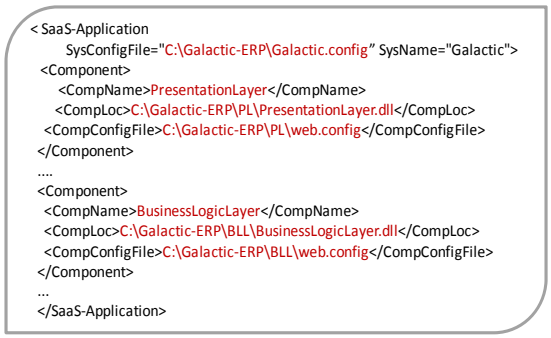

Figure 4: Part of the Galactic architecture description file

IV. TOSSMA ARCHITECTURE

TOSSMA is based on externalizing security from the target applications. Thus applications do not need to perform or know how security is enforced, but still can use security information in their normal operation – e.g. to filter data based on current requesting tenant. This is achieved by wrapping the SaaS application with a system container that can intercept any arbitrary application resource (component, class, or method) at runtime. Tenants' security requirements are captured using a multi-tenant security management console. Such requirements are queried and enforced by the security platform based on the requested resource and the requesting tenant.

The TOSSMA architecture, as shown in Figure 3, consists of a SaaS application description database, system wrapper, application security management console, multi-tenant security requirements database, security controls register, and security enforcement point.

- ***Application architecture description database***: TOSSMA is designed as a platform that can handle multiple SaaS applications hosted on the same cloud platform. Each service provider interested in integrating their SaaS application with TOSSMA has to provide their application's architecture description file, as shown in Figure4, that specifies application main components, deployment packages, and components' configuration files. These architecture description files are maintained in the application architecture description database (Figure 3-1) and used by TOSSMA to reverse engineer system details (classes and methods) and configure (inject/de-inject) system interceptors.
- ***Application security management console*** (Fig 3-3): to simplify the process of specifying SaaS applications security requirements at the supported details level, each tenant is introduced with an application description that is reverse engineered from the registered application architecture. Tenants select from the system description (components, classes, methods, etc.) the critical points they are worried about and specify their security requirements on them, as shown in Figure 5.
- ***A multi-tenant security requirements database***: the security requirements specified by the service provider or by a service tenant are captured and maintained in the multi-tenant security requirements database (Fig 3-4). Tenants and providers can view, maintain and enforce their own security requirements without impacting others' requirements. Each entry in the database contains a system entry point name along with the security requirements to be enforced on it. The priority of the specified security requirements goes to the service provider first then its tenants. For example, if the service provider specified a certain authorization security requirement on a system entry E and other tenants specified their own authorization security requirements/controls on E as well, then TOSSMA enforces the service provider authorization requirements and then tenants' authorization requirements.
- ***The system wrapper*** (Fig 3-2) is a module responsible for injecting interceptors into the running SaaS application at critical system entry points specified by tenants. The system wrapper is based on dynamic-weaving AOP where both point-cuts and aspects are specified at runtime. For example, if tenant T1 specified certain security requirements on Component C, this means that all methods in this component should be intercepted to enforce security requirements specified by T1 on C. Whenever the tenant or the SaaS provider discover a threat or have a new security objective for a given method M, they can extend their enforced security at runtime with the new required security on M. The system wrapper then intercepts requests directed to any registered method and delegates it to the nominated handler (security enforcement point) that enforces the latest security requirements specified on the given method.

To support security integration with new and existing applications, TOSSMA provides two system wrappers: (a) design time dependency injection wrapper used by service providers during application development; (b) static, after development AOP-based wrapper is used to modify existing application binaries to inject security aspects based on tenants' and service providers' needs.

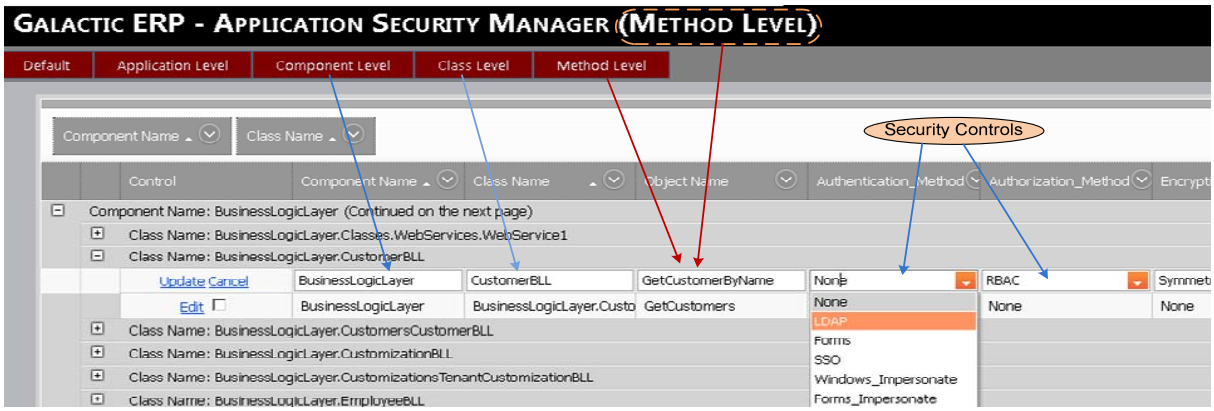

Figure 5: TOSSMA application security management console

- ***Security controls database****:* tenants, service provider, and cloud platform providers register security controls that can be used in securing the SaaS tenants' data while being stored, processed and transmitted (Fig 3-5). Each registered security control should have its APIs, or web service URL, and the expected parameters, so TOSSMA can communicate with it whenever required using the standard security interface. Each security control is mapped to a predefined security category including authentication, authorization, encoder, input validation, and auditing. For each security category, TOSSMA has a predefined set data items to be communicated with target security controls.
- ***Security enforcement point****:* requests to a critical system resource are intercepted by the system wrapper and delegated to the security enforcement point (Fig 3-6). This loads the security requirements specified for the intercepted method/resource, according to the current requester's tenant using from the multi-tenant security requirements database. This means that we can enforce different tenants' security requirements on the same method based on the requesting party. The security enforcement point checks the retrieved requirements and issues requests to the corresponding security controls with the necessary information (parameters) required for each security control. For example, if we have an access control requirement to use CA Identity Manager, the security enforcement point calls the corresponding CA identity manager client API and passes the requester identity and the requested resource URI. Based on the returned results, the security enforcement point decides either to continue with the request processing by the SaaS application or to reject the request, as shown in Figure 3.

TOSSMA supports specifying and enforcing tenants' security at four levels, as shown in Figures 3 and 5: *Application level* where each call to any public method will be intercepted. All methods enforce the same security requirements; *Component level* where methods/services inside a specific component are secured; *Class level* where methods/services inside a specific class are secured; and *Method level* where specific methods are intercepted.

V. USAGE EXAMPLE

In this usage example we focus on multi-tenant SaaS applications where tenants share a single instance of the service. This is the most complex scenario for any multi-tenant SaaS application from security specification, enforcement and management perspectives. To demonstrate the capabilities of our new TOSSMA architecture, we revisit the motivating example from Section II. Each tenant has their own distinct security requirements to be enforced on Galactic. We developed a prototype of our architecture to help in practically evaluating the architecture. We also developed a prototype for Galactic application as a sample multi-tenant application.

The first step in enabling TOSSMA platform to manage security of Galactic application is to register Galactic in TOSSMA. To host Galactic, SwinSoft should deliver a high level architecture of Galactic illustrating its main components, locations and corresponding configuration files, as shown in Fig4. Upon registering Galactic application architecture, TOSSMA reverse engineer Galactic to retrieve the application public classes and methods. The results are organized and displayed in the security management console UI, Figure5.

Once a service tenant, e.g. Swinburne or Auckland, has registered to use Galactic, they get access to TOSSMA security management console. Then, they can manage their assets security at the level they would like to work on (system/component/class/method), Fig5. TOSSMA is responsible for (i) delegating security requirements to the lower levels, (ii) raising and resolving conflicts of security requirements specified at different levels – e.g. we assume that if a tenant specified two different security requirements one on a component C and another on one of its methods M, this means that he is really interested in applying different (higher/lower) security requirements on this specific method rather than the other component' methods. Fig6-2 shows example of the security requirements XML file for Swinburne. It specifies that whenever the intercepted method is "GetCustomers" then TOSSMA should enforce authentication using Forms-based authentication and authorization using RBAC impersonate

control. This file is generated and maintained by the security management console so that tenants can revise their enforced security requirements as needed.

SwinSoft, GreenCloud, Swinburne and Auckland register security controls that can be used in securing Galactic and tenants' data maintained by Galactic. Each security control should reflect the control URL and its category (authentication, authorization, input validation…). Examples of registered security controls for authentication (LDAP-based, forms-based, SSO…) and authorization (RBAC, ABAC…) are shown in Figure5.

Any update to a tenant's security requirements or a registration of new tenant with new security requirements triggers the security enforcement point to modify the system wrapper and add new interceptors into Galactic methods – e.g. add interceptor to GetCustomers method as shown in Figure6-1 - as specified in the tenants security requirements. Thus only methods specified by tenants as critical will be intercepted to weave security controls required by Galactic tenants.

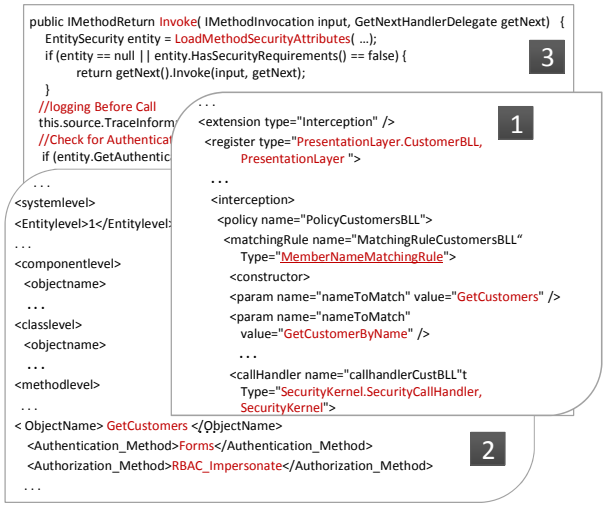

Figure 6: Examples of security specification file, system wrapper configuration, and security enforcement point

VI. Implementation

Our prototype implementation for TOSSMA uses dynamic AOP to intercept the system execution at runtime based on system interceptors' document. TOSSMA prototype and sample application were developed using C#. Microsoft .Net does not implement AOP. We use "Unity - application block" developed by the Patterns and Practices team at Microsoft. Unity allows us to define interceptors on certain components, classes, and methods through application configuration files. Unity application requires a method "Handler" to be called whenever a registered component, class or method is requested. We developed the security enforcement point as a class library, and used it as the interception handler. Figure 6-3 shows a sample of the security enforcement point. We adopt Yiihaw[1] as a system wrapper for existing applications. Yiihaw enables modifying application binaries (dll and exe files) to inject the security aspects wherever specified by the application tenants. A default aspect is weaved with critical points. It simply calls the security enforcement point handler before and after the method body. Figure 6-3 shows a sample of the security enforcement point. Interception pointcuts' signature is defined based on the tenant selected method or component signatures captured by the security management console. This console is developed using ASP.Net so it can be deployed as a web application for the SaaS tenants interested in securing their SaaS data, shown in Fig5. We adopted the OWASP Enterprise Security APIs (ESAPI) library as our security controls database[2].

VII. Discussion

A. Experimental Evaluation

In this section we summarize our experimental evaluation we have performed to assess the capabilities of TOSSMA in capturing a range of SaaS application security requirements for different tenants, generating interceptors for the tenants specified critical application entry points; and enforcing such security requirements/controls at runtime based on the intercepted request' tenant.

Benchmark Applications: We have tested our architecture with two newly developed applications (GalacticERP and PetShop), where we use the Unity application block as the system wrapper. We also tested TOSSMA on two existing, third-party web applications (SplendidCRM, KOOBOO). Table1 summarizes there statistics (lines of code, no. files, classes, methods).

Table1: Benchmark applications statistics

Benchmark	KLOC	Files	Classes	Methods
Galactic	16.2	99	101	473
PetShop	7.8	15	25	256
SplendidCRM	245	816	6177	6107
KOOBOO	112	1178	7851	5083

Table2: Security controls used by Tenant1, Tenant2

Sec. Attribute	Tenant(1) Control	Tenant(2) Control
Authn. & ID Mgmt	Forms-based	LDAP
Authz.	Forms-based	LDAP
I/P santization	ESAPI Validator	ESAPI Validator
Audit	ESAPI Auditor	Private Auditor
Cryptography	DES	AES

Experimental Results: We validated our architecture capabilities in enforcing authentication, authorization, input validation, logging and cryptography on both new and existing applications. Table2 shows two sets of security controls we used to enforce different security attributes on target systems defined by two different tenants. Results of our experiments are shown in Table3. TOSSMA succeeded in capturing and enforcing different security attributes (identity management, authentication, authorization, cryptography, digital signature, and input validation) for multiple tenants at runtime on the same service instance. However, it suffers from two key limitations. Supporting cryptography is limited, as it currently requires the caller

[1] www.itu.dk/~sestoft/papers/yiihaw-usage-guide.pdf

[2] https://www.owasp.org/index.php/ESAPI

and callee to use parameters of only type "string". Thus we could only apply it on methods with signatures that fit with these requirements. This can be handled using web server httpModules extensions (component level). *Second*, applications with existing, built-in, security need to be modified to disable existing security before using TOSSMA. Otherwise they will keep enforcing the old as well as the new security requirements.

TABLE3: VALIDATION RESULTS OF TOSSMA ON NEW AND EXISTING SAAS

Security Requirements	New Development		Existing Application	
	Galactic	PetShop	Splendid	Kooboo
Identity Mgmt	√	√	√	√
Authentication	√	√	√	√
Authorization	√	√	√	√
Input Validation	√	√	√	√
Audit	√	√	√	√
Cryptography	√	√	o	o

Performance Evaluation: The Performance overhead of adopting TOSSMA architecture, to support multi-tenant adaptable security, depends on the number of critical system entries and the number of concurrent users currently requesting critical system resources, as shown in Figure7 (time in msec). The performance overhead is measured on a desktop PC with core2 duo processor and 4GB memory. This performance overhead will impact only the tenant secured resources. Thus, if the tenant does not enforce security on resource X, then he will not suffer from any performance overhead when using this resource, although other tenants may be enforcing certain security on X. This is crucial in managing tenants' SLAs.

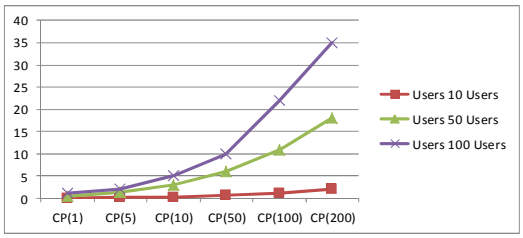

Figure 7: Performance evaluation of TOSSMA

B. *Threats to validity*

Integrating any given SaaS application with TOSSMA does not require any further code by the SaaS service provider. Once the system architecture is available, TOSSMA is able to integrate and secure the target system. Efforts required by the system tenants to secure their data are limited to security configuration activities including selecting the critical system entities where security should be enforced, and specifying security controls to be applied at the selected system entities. These activities are done visually through the security management console. Security controls configurations are managed by security admins.

TOSSMA supports four different levels of security specification (system, component, class/object, method). The selection of the level of detail to apply interception on depends on the criticality, architecture, environment, etc. of the target system. In some situations like web applications, we may need to intercept calls to the presentation layer only while considering the other layers secured by default (not accessible except from the presentation layer). In other cases such as offering certain web services or using third party services to deliver certain functions we may need to have security enforced at the method level for certain methods only. There is also a security and performance trade-off. The more security validations and checks the more resources required that impacts its performance.

TOSSMA enables service providers to implement security controls required to support tenants' data isolation at any required system entry point. TOSSMA enforces SaaS application provider security requirements/controls before enforcing the tenants' security requirements. Thus tenants will not be able to read data of other tenants where they are not authorized to read. SaaS application providers can use TOSSMA as a plug-in of their applications. Moreover, cloud providers can use it as a PaaS to manage all the cloud hosted SaaS applications.

Overall, our approach provides a tenant-oriented SaaS security management architecture, a toolset that supports capturing application architecture and tenants security requirements, and enforcing such security specifications on a SaaS application at runtime without the need for bespoke system customizations. It promotes security engineering from application provider at design time to the end-user at runtime. This allows tenants' security engineers to consider new security issues that arise during system operation and have not been seen during the design phase. Our approach works for both new systems and existing systems.

We are working on extending TOSSMA to support capturing high-level security requirements (risks and threats) and automatically generate security requirements, mechanisms and the required security configurations accordingly. We plan to automate the testing of security controls integration into the specified system entry points.

VIII. RELATED WORK

The area of multi-tenant SaaS applications' security engineering is relatively new. Existing multi-tenancy security solutions from industry and academia are under development.

Michael et al [11] discuss the limitations of security solutions proposed by different commercial cloud platforms. SalesForce.Com has introduced a simplified solution to support their CRM integration with tenants' security solutions. They focus on the Identity and Access Management (IAM) area only. Tenants who are interested in integrating with SalesForce have to implement web services with a predefined signature. Microsoft has introduced more advance extensible security model - Windows Identity Foundation (WIF) to enable the service providers to deliver applications with extensible security. It requires service providers to use and implement certain interfaces in system development. The java Spring framework has an extension framework – Acegi. It implements a set of security controls for identity management, authentication, and authorization. It requires manual configuration of the application to adopt these controls. Moreover, it does not support for multi-tenant security.

Enabling applications to support multi-tenancy either during application development or by adapting existing web

applications to support multi-tenancy has been investigated by [12-15]. Cai *et al.* [4, 5] propose an approach to transform existing web applications into multi-tenant SaaS applications. They focus on the isolation problem by analyzing applications to identify the possible isolation points that should be handled by the application developers. Guo *et al.* [6] 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 while still using a predefined, built-in, security controls. It depends on the tenant's administration staff to manually configure security policies and map their users and roles to the application predefined roles. Pervez *et al.* [7] 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 on the security mechanisms used. Moreover, no isolation is provided between the authentication and authorization data of different tenants.

Xu *et al.* [9] propose 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 administrators delegate the authorization to the tenants' administrators to grant access rights to their corresponding resources. Zhong *et al.* [8] 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.* [16] propose a model-driven platform to compose SaaS applications as a set of services. Their approach focuses on enabling cloud consumers to compose their system instances and define their security requirements to be enforced on the composed web services. However, tenants' instances must be deployed on separate VMs. Moreover, there is no means to update or reconfigure the defined security.

These efforts deliver security using specific solutions and architectures. However, they do not give tenants control on their assets security, do not support multi-tenant security, and do not support runtime enforcement.

IX. SUMMARY

TOSSMA is a new tenant-oriented, SaaS application security management architecture. It promotes security engineering from system-oriented security to tenants-oriented security. This enables multi-tenant SaaS applications to easily capture different tenants' security requirements and enforce such requirements using the security controls selected by the SaaS tenants. Security controls are weaved with the application at runtime without a need for re-engineering or developers' involvement. TOSSMA mitigates four main problems in multi-tenant cloud applications: the loss of security control over cloud hosted assets by letting each tenant secure their data based on the importance and the risks they consider; the integration of the SaaS application security with the tenant's already existing and enforced security mechanisms; the customization of SaaS applications security to mitigate new vulnerabilities; and providing isolation between tenants' data by extending applications to enforce authorization at critical methods. We have developed a prototype for TOSSMA using .Net. We have evaluated our approach on four applications (Galactic, PetShop, SplendidCRM, and KOOBOO). We conducted performance evaluation of TOSSMA with different SaaS applications' sizes and number of concurrent users.

ACKNOWLEDGEMENT

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] M. Almorsy, J. Grundy, and I. Mueller, "An analysis of the cloud computing security problem," presented at the Asia Pacific Cloud Workshop, APSEC2010, Sydney, Australia, 2010.

[2] Cloud Security Alliance, "Domain 10: Guidance for Application Security V2.1," July 2010.

[3] W. Sun, X. Zhang, et al, "Software as a Service: Configuration and Customization Perspectives," in Proc.*2008 4th IEEE World Congress on Services Part II, 2008*, pp. 18-25.

[4] H. Cai, N. Wang, et al, "A Transparent Approach of Enabling SaaS Multi-tenancy in the Cloud," in Proc.*2010 6th IEEE World Congress on Services,* 2010, pp. 40-47.

[5] H. Cai, K. Zhang, et al, "An End-to-End Methodology and Toolkit for Fine Granularity SaaS-ization," in Proc. 2009 *IEEE Int. Conf. on Cloud Computing, 2009*, pp. 101-108.

[6] C. Guo, W. Sun, et al, "A Framework for Native Multi-Tenancy Application Development and Management," in Proc. *9th IEEE Int. Conf. on E-Commerce Technology*, pp. 551-558.

[7] Z. Pervez, S. Lee, et al, "Multi-tenant, secure, load disseminated SaaS architecture," in Proc. *12th Int. Conf. on Advanced communication technology*, South Korea, 2010, pp. 214-219.

[8] C. Zhong, Y. Xia, H. Yu, "Construction of a Trusted SaaS Platform," in Proc. *2010 5th IEEE Int. Symposium on Service Oriented System Engineering,* 2010, pp. 244-251.

[9] T. J. Jing Xu, H. Dongjian, et al, "Research and implementation on access control of management-type SaaS," in Proc. *IEEE Int. Conf. on Information Management and Engineering,* 2010, pp. 388-392.

[10] B. Wang, H. He, et al, "Open Identity Management Framework for SaaS Ecosystem," in Proc. *IEEE Int. Conf. on e-Business Engineering*, 2009, pp. 512-517.

[11] M. Brock and A. Goscinski, "Toward a Framework for Cloud Security," Algorithms and Architectures for Parallel Processing." vol. 6082, C.-H. Hsu, L. Yang, J. Park, and S.-S. Yeo, Eds., ed: Springer Berlin / Heidelberg, 2010, pp. 254-263.

[12] R. Mietzner, F. Leymann, et al, "Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-tenancy," in Proc. *3rd Int. Conf. Internet and WebApplications and Services*, 2008, pp. 156-161.

[13] D. Wang, Y. Zhang, et al, "Research and Implementation of a New SaaS Service Execution Mechanism with Multi-Tenancy Support," in Proc. *IEEE Int. Conf. on Information Science and Engineering*, 2009, pp. 336-339.

[14] B. S. Zhang, X. Tang, et al, "From isolated tenancy hosted application to multi-tenancy: Toward a systematic migration method for web application," in Proc. *IEEE Int. Conf. on Software Engineering and Service Sciences,* 2010, pp. 209-212.

[15] R. Chinchani, A. Iyer, et al, "A target-centric formal model for insider threat and more," TR2004-16, Buffalo University, US2004.

[16] M. Menzel, R. Warschofsky, et al, "The Service Security Lab: A Model-Driven Platform to Compose and Explore Service Security in the Cloud," in Proc. *World Congress on Services* , 2010, pp. 115-122.