Adaptive Security on Service-based SCM Control System

Gabriel Serme
Eurecom
serme@eurecom.fr

Muhammad Sabir Idrees
Eurecom
idrees@eurecom.fr

***Abstract*—On a large-scale application subject to dynamic interactions, the description and enforcement of security rules are complex tasks to handle, as they involve heterogeneous entities that do not have the same capabilities. In the context of SCM-application for example, we have different goods that are being transported across different systems. At one point, items and systems communicate together to signal presence, report issues during transport, certify validity of previous checks, etc. Security capabilities of the involved parties are heterogeneous and one might want to specify security policies on an abstract level and let the involved systems enforce them according to their contexts and the specific capabilities of each party. In this paper, we propose a framework for security mechanisms adaptation when services are involved by using Aspect-Oriented-Programming (AOP) concepts that can be applied to SCM applications. The novelty is the expressivity of security policy at a global level and the enforcement at a local level, through a specific and distributed aspect model that has a larger semantic to catch up events relevant for business usage and dedicated to security concerns.**

Keywords-SOA, Security, AOP.

I. INTRODUCTION

An SCM application can be viewed as a long chain process along which goods have to pass through mandatory gates. It involves various devices, from embedded systems like sensors to large-scale servers in backend systems. Sensors usages are dedicated to data collection and signal triggering. They try to capture real-world status and measure it. Backend systems allow for data processing but need to adapt to all devices communicating with them, as each can have a different communication protocol and data format.

The heterogeneity of platforms and software used in devices makes it difficult to manage simple security rules, especially across a supply chain. In order to deal with the multiple possibilities and not to interfere with the business part of software, one might want to describe security behavior for one system that adapts to security capabilities of systems communicating with it. To do so, we propose an architecture that allows correct modularization of security concerns to quickly intervene in applications and make them adapt to the conditions they can face up to.

The application uses the SOA architectural style to provide a loosely-coupled platform where entities can integrate with each other. In the following sections, we start by explaining the different concepts we are using in our proposed architecture. Namely, Web Services and SOA concepts, security properties we aim to express in an adaptive manner and also AOP (Aspect-oriented programming) paradigm. Then, we describe the proposed architecture and process to handle service adaptation with two examples highlighting difficulties to adapt security for systems accordingly.

II. SERVICES

Service Oriented Architectures (SOA) enable a world of loosely-coupled and interoperable software components towards reusability. Nowadays, the main entity used to represent a software service is a Web Service. Web-Services represent a paradigm defined by W3C as ”a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, conveyed using HTTP with an XML serialization in conjunction with other Web-related standards” [1]. Web Services can also be addressed through other transport mechanisms such as JMS or ESBs.

The Web Service standards stack goes beyond the atomic service, and proposes different approaches depending on the level of abstraction. Service behavior can be defined when linking different services together, *e.g.,* with BPEL4WS or BPMN 2.0 [2]. It allows definition of service composition to realize a so-called business process.

III. SECURITY

As services are advancing fast and are being extensively deployed in applications spanning different organizations, it becomes crucial to ensure security and trust for these applications to hold their promise. It was only recognized in recent years that services are themselves susceptible to various attacks at different levels of system conceptualization [3].

Since SOA has a flexible set of design principles used during the phases of system development, integration, and evolution, one obvious and common challenge is to secure SOA. This often involves invasive modifications, in particular to enable new security functionalities that require modifications to applications. Furthermore, enforcing crosscutting security functionality in service-based systems is difficult to specify and implement because the security of services and

 ISBN: 978-1-61208-144-1

their compositions is not modular. Modifications made to one part of an application may interact strongly with the security properties of other parts of the same application. Security properties generally pervade software systems, that is, security properties crosscut service-oriented architectures. Enforcing service level security needs specialization based on the implementation.

We propose in the following an aspect-based service model that proposes an original method to introduce several security properties. These security properties are specified in a security policy language that is then interpreted to generate crosscutting concerns. It includes *Integrity* which relates to communication, storage (resources), and execution (process - infrastructure) integrity. The execution environment integrity is an important security objective together with data integrity measures. *Confidentiality* relates to message exchange between entities such as sensors and services, or need-to-know principles limitation applied to specific resources. *Authentication and Authorization* crosscut applications to decide at several points if a given subject is allowed to perform an action on a given resource. Whereas authorization decisions are mainly on server-side, authentication mechanism needs to adapt all peers to agree on the scheme, including sensors authentication. Applying *non-repudiation* requires the implementation of an asymmetric encryption scheme in the execution environment supporting the computation.

The aforementioned properties represent security goals we want to apply on applications by adapting them with our framework.

IV. Aspect-Oriented Programming

The term Aspect-Oriented-Programming [4] (AOP) has been coined around 1995 by a group led by Gregor Kiczales, with the goal to bring proper separation of concerns for cross cutting functionalities. Roots for foundations can be traced back to adaptive programming, or composition filters [5]. O. Selfridge introduced a notion that can be related to AOP as ”demons that record events as they occur, recognize patterns in those events, and can trigger subsequent events according to patterns that they care about” [6]. But the approach has then derived to become a discipline apart.

The aspect concept is composed of several advice/pointcut couple. Pointcuts allow to define where (points in the source code of an application) or when (events during the execution of an application) aspects should apply modifications. Pointcuts are expressed in pointcut languages and often contain a large number of aspect-specific constructs that match specific structures of the language in which base applications are expressed, such a pattern language based on language syntax. Advices are used to define modifications an aspect may perform on the base application. Advices are often expressed in terms of some general-purpose language with a small number of aspect-specific extensions, such as the *proceed* construct that allows the execution of the behavior of the base application that triggered the aspect application in the first place. The main advantage using this technology is the ability to intervene in the execution without interfering with the base program code, thus facilitating maintainability.

V. Architecture proposal

In this paper, we shape the solution we are currently implementing at the service layer. The root of the problem is to instrument several services at the same time, potentially not under the same execution environment to realize a specific security property. Illustration examples are available in next section. The architecture is presented in Figure 1. It contains two parts, involving design and runtime part. The design part involves business stakeholders to define aforementioned security policies (also denoted rules), and security experts to provide concrete security mechanisms as pre-defined aspects. The runtime part of the architecture leverages the aspect model to modify the different execution environments and make them satisfy security policies specified by business and security stakeholders. The main piece is the runtime engine whose goal is to detect a certain *state* across platforms. The state is described by *rules* composed of *predicates*. Upon matching between rules and a state, platforms coordinate to realize a new behavior. Locally, systems implement *mechanisms* to realize a behavior specified in the knowledge base and make usage of *context information* available at execution.

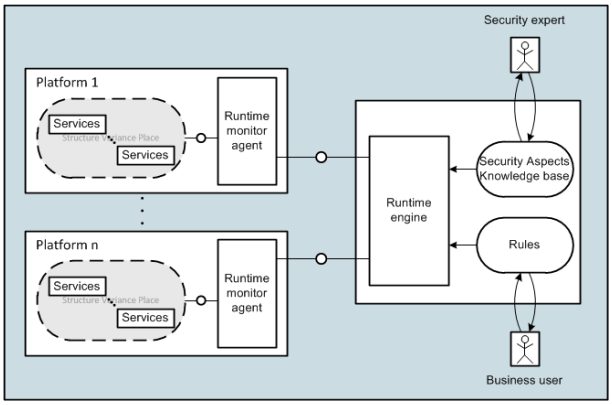

Figure 1. Proposed architecture

Runtime monitor agents and runtime engine work together to realize a distributed aspect model, introduced in [7]. The advantage of such an architecture is to intervene in one specific organization where we separate security concerns across different platforms. The security code is no longer tied to the business code. Rather, it is decoupled from business code intentionally and bound with the distributed pointcut language. The runtime engine uses rules to gather state of various services at the same time, and security aspects are used in advice to dispatch security behavior

 ISBN: 978-1-61208-144-1

across several services. For example, in the integrity and origination scenario, the distributed aspect model tracks all places where messages come in the system to taint them. Then, the system tracks these taint messages to weave security aspects when behavior is needed.

Security policies are specified by business users, security experts and architects then derived in rules to realize a specific security property. To provide an enriched semantic addressing all concerns of services and security at the same time, we are developing a policy which contains high level description of wanted behavior, through predicates. The predicates allow matching with a particular state. The language semantic relates to events and actions a platform can generate, *i.e.,* services, messages and resources. *Services* can be atomic or composite, *e.g.,* orchestration of services with BPEL4WS or BPMN2.0, or simply a service that consumes other services. Examples of predicates that can be used are "receive" or "reply" to match a service call. There are also predicates for *messages* and *resources*. Predicates have different level of abstraction relating to the service stack we discussed above.

The policies also contain behavior that stakeholders want to introduce across systems. In our framework, we address only security concerns such as the ones described in previous sections: confidentiality, integrity, authorization, etc. In Listing 1, the policy describes integrity and non-repudiation presence in messages when they are issued by sensors. The message origination is verified when we are able to verify signature. The behavior is described in an abstract way to indicate which parties are concerned and what shall be executed. Platforms receive this behavior and are in charge of translating it according to mechanisms available for the given platform. The Listing 2 in next section is an exemple of java code to verify a given signature for a message. The runtime monitor detects a certain application state through the aforementioned predicates. Upon matching, the wanted behavior is read from policies and spread to concerned systems to satisfy and realize security properties.

```
1  message_in (issuer, msg):
2    issuer in (s:sensors) => verify_integrity (s,
         msg), verify_origination (s, msg)
3    msg.taint(UNSAFE) #Default
4    msg:integrity, msg:non_repudiation => msg.taint(
         SAFE)
5
6  verify_integrity (msg):
7    msg.contains(integrity), integrity.check(msg) =>
         msg:integrity
8
9  verify_origination (msg, issuer):
10   msg.contains(sign), sign.issued(issuer) =>
         verify_origination (msg, sign,issuer), msg:
         non_repudiation
```

Listing 1. Policy snippet for integrity and non-repudiation check

Our framework heavily relies on aspect oriented paradigm. The runtime monitor is able to detect system state using a distributed aspect pointcut language. On matching system state, advices are executed with the system's context through context exposition mechanisms. For example, a service can expose information about inputs, outputs, service origination, as well as other security-related information.

The proposed architecture allows definition of security policies for service systems at a global level that are then enforced at a local level in an semi-automatic mode. We propose to decouple definition of specific security properties from the base application, and let declaration through rules respecting application owners' needs.

VI. Application Example

We describe two scenarios illustrating when our framework can be applied in a SCM application. The long term scenario (cf Fig. 2) is a military container that is sent to supply a camp thanks to a boat transport. Shipment is not direct and the container has to pass through several intermediaries, to refuel, change boat, etc.. Thus, it has been decided by the army to frequently track and check containers when they stop in harbors. The communication between containers and the army system is made through Web Services and use the Harbor system to certify the shipment advancement.

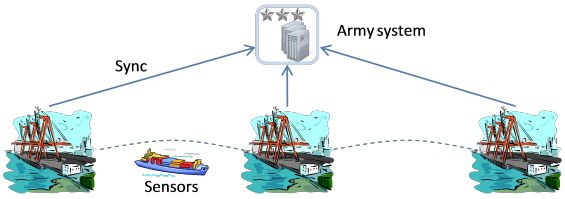

Figure 2. Army shipment and control system

The first scenario leveraging our framework is on adaptation and tracking of sensitive data. It highlights integrity and non-repudiation scenario and how it impacts the existing architecture on the army systems. Over time, the army deployed containers with protection mechanisms to detect failure or intrusion composed of sensors and nodes. Different software solutions are shipped with containers with different security capabilities.

Maintaining applications, both in nodes and back-end systems, is costly. It requires business owners to specify each possible use case, at a time t. Upon release of new version, they need to extend existing specifications and activate their development team. The development team has to correctly implement the solution and to not break the previous solutions. The release cycle can be counted in weeks or even months and is error prone. A suitable solution is to have a framework that knows what to do given a certain situation. For example we want to allow communication between all nodes versions and the back

 ISBN: 978-1-61208-144-1

end system while keeping track of sensitive nodes - those which do not implement security mechanisms. An example of policy we might define to detect various versions of protocols is described in Listing 1. The policy language used is not yet fully developed and is used as an illustration.

```
1  @Aspect
2  Class Verification{
3    //...
4    @Covers(SPL.verify_origination) //Provided by
         framework
5    boolean verifyOrigination (Byte[] msg, Byte[]
         msgdsig, Identity issuer){
6      //get public key of issuer
7      X509EncodedKeySpec pubKeySpec = new
           X509EncodedKeySpec(Security.getPubKey(
           issuer));
8      KeyFactory keyFactory = KeyFactory.getInstance
           ("DSA", "SUN");
9      PublicKey pubKey = keyFactory.generatePublic(
           pubKeySpec);
10
11     //get message signature
12     Signature sig = Signature.getInstance("
           SHA1withDSA", "SUN");
13     sig.initVerify(pubKey);
14     sig.update(msg);
15
16     //verify
17     boolean verifies = sig.verify(msgdsig);
18     return verifies;
19     }
20 }
```

Listing 2. Java snippet for proof of origin as aspect

Figure 3 shows the sequence diagram of two containers notifying the back-end system. Containers one (C1) and two (C2) both send the same type of information to the army system. But C2 uses a newer protocol which includes a proof of origin to avoid tamper risk on transmission. The rectangles and their attached dotted-lines in the figure are points in architecture where our framework intervenes and injects mechanisms. With our framework, the back-end system intercepts data coming in the system and verifies it, thanks to a runtime monitor agent. It detects security protections from containers and provide to back-end services the data formatted accordingly. A taint mechanism marks data depending on its state and policy in place. Detection is made through platform implementation, such the one described in Listing 2. The listing respects policy declaration, as shown in Line 4 which bind the code - hence the behavior with the policy. The method signature is extracted from the leaf policy $verify\_origination(msg, sign, issuer)$. It then allow verification of signature. The message is tainted depending the method execution result. The piece of code is processed only upon correct matching and return information understandable by the "runtime engine". In our example, the origination of $data1$ cannot be verified. As the policy in Listing 1 expresses, the data is marked as $UNSAFE$. When headquarters request this data, the army system knows the data is unsafe and can propose notification mechanisms to warn user about data uncertainty.

The second use-case shows authorization and confidentiality check with our framework. The adaptation relies heavily on the context. We first explain the authorization part. The army system receives a document composed of parts with different authorization level : L1 and L2. We are in the context of a Mandatory Access Control thus strong hierarchy and definition of who is allowed to perform which action on which resource.The resource is composed of two parts. One that contains logistics information, such as freight id, container weight or event history of harbour. The second part contains details about freight : composition of the fret, final destination and usage, etc. The second part contains strategical information, that only high-ranked militaries can consult. As shown in Figure 4, a lieutenant and the logistics officer try to access resources sent by the container. The former can access all parts while the latter can only access the L2-part. When the lieutenant accesses the sensitive part of the data, the runtime monitor detects usage of a sensitive data and adapt the platform to provide confidentiality between involved peers : encryption for the lieutenant before data transmission and decryption mechanism after transmission.

Our framework intervenes from security rules upon detection of data from a specific container, the runtime monitor triggers a code that marks different parts of the data. Then, it introduces a behavior when sensitive information is transmitted. The concrete implementation of mechanisms is made locally. For instance, when the lieutenant requests the sensitive part of the data, the encryption/decryption mechanisms are executed. Systems agreed upon a behavior then implementation and execution of security mechanisms is made locally.

VII. Related work

We divide the related work in two separate categories addressing security-related solutions with aspects, or aspects for services. The former often imply modelling of security properties beforehand to latter enforce them correctly on the system. Translation mechanisms are often hand-written. The second category focuses on AOP and how to introduce its underlying concepts in services. To the best of our knowledge, no concrete work has been done to address security concerns that pervades both applications and services while proposing decoupling from the business code.

In [8], Baligand uses AOP with Web Services to introduce non-functional requirements, following a policy. The difference with our work is they do not cover simultaneous orchestration of different services to realize one capability. [9] has the same goal but proposes an XML-centric approach to specify pointcuts and advice whereas we rely on automatic matching from policy rules. In [10], Ganesan *et al.*. addresses an aspect model for composite services. They

 ISBN: 978-1-61208-144-1

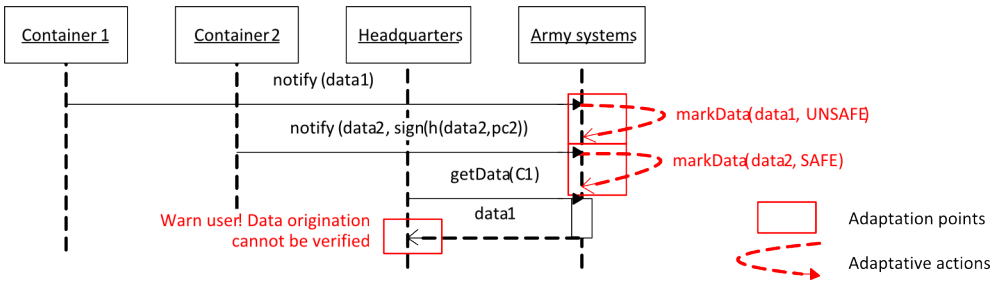

Figure 3. Multi platform adaptation

introduce a specification language to design non-functional requirements as distributed aspects, but they do not cover security *per se*. In [11], Mostéfaoui *et al.* also address a framework to decouple security concerns with aspects on web services. They use frames concept to have a configuration including both composite and component level. In [12], the authors provide an architecture to have distributed aspects to modularize and adapt non-functional requirement but only for composite services. Also, their approach implies advice code to be already on target platform for execution. In [13], Jakob *et al.* use AOP to secure distributed systems. Their approach is rather to specify early security properties thanks to pointcut language tied to an architecture diagram. [14] exposes a concrete use case of applying authentication with AOP in a SOA-based sensor architecture. Whereas it provides concrete mechanisms as we aim to do, we go a step further by binding these mechanisms with policies. It makes policy analysis way more consistent over time. In [15], Mourad *et al.* use an AOP-based language for security hardening. The language introduces concepts close to pointcuts. Therefore, the language does not cover services.

VIII. Conclusion

Addressing cross-cutting concerns that pervades services with strong focus on security lead us to a new architecture proposal. We have seen through our example that this architecture can be applied to an SCM use case. It gives tools and methods from early phase of application design to implementation and maintenance of sensors to gather accurate context information. From modelling information, ones decide what are specifications that have to be enforced during the execution of the application. In other words, the proposed architecture allows definition of security policies for service-based systems at a global level that are then enforced at a local level in an semi-automatic mode. We propose to decouple definition of specific security properties from the base application, and let declaration through rules respecting application owners' needs. A prototype is under development to address the runtime part - modification of different execution environment with Aspects to introduce security features. Currently, we limit complexity to one platform at a time. We want to investigate modifications across platforms in future work - platforms not located under a same administrative domain. It requires trust and mechanisms to ensure synchronisation, guaranties that security is effectively implemented to mention a few.

Acknowledgment

This work has been carried out in the CESSA (Compositional Evolution of Secure Services using Aspects) project, supported by the ANR, the French national research organization (project id.: 09-SEGI-002-01).

References

[1] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard, "Web services architecture,"

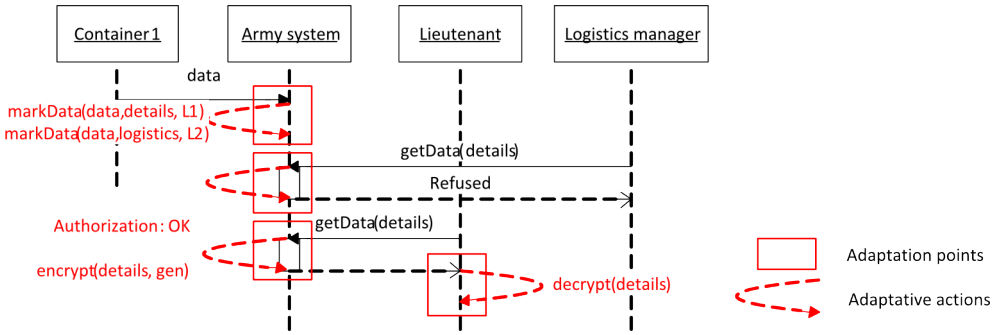

Figure 4. Authorization and confidentiality mechanisms

http://www.w3.org/TR/ws-arch/, vol. 99, no. 7, pp. 1–100, January 2004.

[2] R. Hull and J. Su, "Tools for design of composite web services," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '04. New York, NY, USA: ACM, 2004, pp. 958–961.

[3] M. Jensen, N. Gruschka, and R. Herkenhöner, "A survey of attacks on web services," *Informatik - Forschung Und Entwicklung*, vol. 24, pp. 185–197, 2009.

[4] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *ECOOP*, ser. Lecture Notes in Computer Science, M. Aksit and S. Matsuoka, Eds. Springer Berlin / Heidelberg, 1997, vol. 1241, pp. 220–242.

[5] C. V. Lopes, "AOP: A historical perspective (What's in a name?)," in *Aspect-Oriented Software Development*, R. E. Filman, T. Elrad, S. Clarke, and M. Akşit, Eds. Boston: Addison-Wesley, 2005, pp. 97–122.

[6] O. G. Selfridge, "Pandemonium: a paradigm for learning. In Mechanisation of Thought Processes," in *Proceedings of a Symposium Held at the National Physical Laboratory*. London: HMSO, 1958, pp. 513–526.

[7] L. D. B. Navarro, M. Südholt, W. Vanderperren, B. De Fraine, and D. Suvée, "Explicitly distributed aop using awed," in *Proceedings of the 5th international conference on Aspect-oriented software development*, ser. AOSD '06. New York, NY, USA: ACM, 2006, pp. 51–62.

[8] F. Baligand and V. Monfort, "A concrete solution for web services adaptability using policies and aspects," in *Proceedings of the 2nd international conference on Service oriented computing*, ser. ICSOC '04. New York, NY, USA: ACM, 2004, pp. 134–142.

[9] M. M. B. Hmida, R. F. Tomaz, and V. Monfort, "Applying aop concepts to increase web services flexibility," in *Proceedings of the International Conference on Next Generation Web Services Practices*, ser. NWESP '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 169–.

[10] K. Ganesan, S. K. Mohalik, and C. Raj, "A distributed aspect model for composite service," in *International Workshop on Service-Oriented Engineering and Optimization*, 2008.

[11] G. K. Mostéfaoui, Z. Maamar, N. C. Narendra, and S. Sattanathan, "Decoupling security concerns in web services using aspects," *Information Technology: New Generations, Third International Conference on*, vol. 0, pp. 20–27, 2006.

[12] K. Ponnalagu, N. Narendra, J. Krishnamurthy, and R. Ramkumar, "Aspect-oriented approach for non-functional adaptation of composite web services," in *Services, 2007 IEEE Congress on*, july 2007, pp. 284 –291.

[13] H. Jakob, N. Loriant, and C. Consel, "An aspect-oriented approach to securing distributed systems," in *Proceedings of the 2009 international conference on Pervasive services*, ser. ICPS '09. New York, NY, USA: ACM, 2009, pp. 21–30.

[14] S. V. Patel and K. Pandey, "Soa using aop for sensor web architecture," in *Proceedings of the 2009 International Conference on Computer Engineering and Technology - Volume 02*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 503–507.

[15] A. Mourad, M.-A. Laverdière, and M. Debbabi, "A high-level aspect-oriented based language for software security hardening," in *SECRYPT*, J. Hernando, E. Fernández-Medina, and M. Malek, Eds. INSTICC Press, 2007, pp. 363–370.

 ISBN: 978-1-61208-144-1