Security-Oriented Negotiation for Service Composition and Evolution

J. Han
School of Information Technology
Swinburne University of Technology
jhan@it.swin.edu.au

K. Khan
School of Computing and Information Technology
University of Western Sydney
k.khan@uws.edu.au

R. Kowalczyk
School of Information Technology
Swinburne University of Technology
rkowalczyk@it.swin.edu.au

1 June 2004
Abstract

This article introduces a framework for security-oriented software service composition and evolution. Key building blocks of the framework are a semantic model for specifying the security objectives and properties at the service and system levels, the negotiation and re-negotiation techniques for service composition and evolution, and the analysis techniques for checking the security compatibility between services and the satisfaction of system-level security goals. It focuses on developing techniques that allow system developers to design required security into service compositions with predictability and to maintain or adapt service compositions in changed security contexts. In contrast to the current practice, we view security from a software engineering perspective, and adopt a proactive and predictive approach to system security.

Keywords: software service composition, service evolution, security, negotiation.

1. Motivation

In recent years, there has been an increasing interest in deploying software applications as services over the Internet as Web and Grid services, to realize greater value of them and support new business ventures [1]. These services are aimed for direct use or for integration with each other within and across organisations to form Internet-based systems and perform cross-application transactions. It is the latter that poses the greatest challenges, where relevant services need to be dynamically discovered and composed with little human intervention to form a collaborative service-based system. The system needs to satisfy the user’s requirements in carrying out a transaction or forming a specific virtual organisation or community under a set agreement. The formation of such a collaborative service-based system and its continued viability should be based on the collective understanding and compatibility of the functional and quality guarantees and requirements of individual services and their compositions. Security is one such quality attribute that is of the greatest concern, especially in an environment as hostile as the Internet. In a service-based system, we need to achieve both the security compatibility between interacting services and the security objectives for the entire system. This is of critical importance in a wide range of application domains including commerce, business, government and healthcare systems.

To illustrate the main theme of this article, let us consider the following application scenario. A number of healthcare organisations, including medical practitioners, pathology labs, pharmacies and hospitals, willingly collaborate by integrating their IT systems, to deliver improved services to their customers in a seamless and secure way. These providers and their IT systems deal with a whole range of sensitive issues related to patient information such as diagnostic reports, MRI images, pathological test results, diagnosis reports, prescriptions, and so on. For example, a pathology lab may advertise its service as providing confidentiality through secure storage of test results. On behalf of its patients, a medical practitioner may require such confidentiality provided at a particular level (as guaranteed by specific encryption schemes with specific key lengths). The pathology lab may or may not satisfy the medical practitioner’s requirement, depending on how the confidentiality guarantee is realised. How do we dynamically and automatically ascertain this issue and form a compatible service composition? Furthermore, how do we know the allowable interactions between services in the composite application actually conform to some system-level security objectives that reflect government regulations? Individual services may change their security provisions, and government regulations may also change. How can the individual services as well as the entire application adapt to these changes dynamically and automatically? Current service-oriented technologies do not offer ready solutions to such advanced, and yet obvious requirements for security-aware service composition and evolution.

In the current practice, the security properties of services are usually not properly analysed or understood, and service compositions are formed with potential compromises to the customers’ and systems’ security requirements. Furthermore, the security provisions of
services and service compositions do not take into account the evolving nature of the security requirements for the services and system, leading to degradation of system security protection and increased customer exposure to security risks. Instead of providing security, this practice virtually forces the service consumers to take risk in order to receive a service by abandoning their preferred security requirements. This is very risky, particularly for service-oriented system applications deployed in the open hostile environment of the Internet, and is not acceptable. To provide better security support, first the individual services need to ensure certain security guarantees relative to their use contexts and security requirements. Second, a composition of these services should achieve the overall security objectives as required for the entire composite system. Third, in the light of inevitable changes relating to services, service compositions and the operating environments, maintaining the existing security properties and adapting to new security requirements for services and service compositions is a major challenge. Based on our work in the first two aspects [5, 6], we in this article examine the essential requirements for and introduce a security-oriented negotiation approach to service compositions and evolution.

2. A framework for service composition and evolution

Figure 1 shows the basic elements of our framework for security-aware service composition and evolution in the general context of system composition and evolution.

A typical framework for service composition and evolution normally consists of three major stages as shown with three rectangles: system architecture design, system instantiation and composition, and system execution and evolution.
and composition, and system execution and evolution. In system architecture design, system requirements are defined and a system architecture is created. In system instantiation and composition, the system is implemented with various services that are typically created and maintained by third parties. The services are located, selected and used by the system. In system execution and evolution, the composed system is in operation and delivers the services to the clients. At this stage, the application may undergo changes, including replacing existing services, modifying system requirements, and so on.

We enhance this framework with additional activities that are used to create security-aware service compositions and to support system evolution in all three stages of the framework. Related to system architecture design are the need to specify security objectives and properties for the composite system and individual services. Service instantiation and composition requires establishing security-oriented service contracts through negotiation. Service execution and evolution involves re-establishing security-oriented service contracts through re-negotiation. These additional security-oriented activities are the central themes of this article, and involve the necessary methods, techniques, and tools that support automated, negotiation-based security-aware service composition and evolution. The three security-related key aspects are:

1. **Definition of security objectives and properties**: A two-level formal model and notation for describing and publishing high-level security objectives and detailed security properties of services and service compositions;
2. **Security contract negotiation for service composition**: Automated negotiation techniques for service composition, including the checking of security compatibility between interacting services and satisfaction of system-level security requirements;
3. **Security contract re-negotiation for system evolution**: Automated re-negotiation mechanisms for system evolution aiming to re-establish security compatibility and security requirements satisfaction in a changed context.

We note that the definition of security objectives and properties associated with system architecture design is primarily the responsibility of the system architect, while the security-oriented contract negotiation and re-negotiation can be carried out in an automated fashion. Each of these high-level activities comprises several low-level tasks as shown in Figure 1. We discuss them in detail in the remainder of the article.

### 3. Definition of security objectives and properties

A service provided by a third party may have implemented its own security measures to protect its computational assets and business objects. The security measures implemented by a service can be viewed from two levels, i.e., high-level security objectives and detailed security properties. A **high level security objective** is the abstract representation of a goal of a security policy. Security policies have certain **security objectives** which are essentially defined to withstand identified security threats and risks. A **security function** enforces the security policy and provides required capabilities [6]. Examples of security objectives are integrity, confidentiality, authenticity, authorisation, non-repudiation from a user’s perspective.

For **detailed security properties**, we consider security measures such as passwords, private keys, public keys, secret keys, shared keys and digital signatures among others which are typically derived from security functions. Security properties are a collection of security elements that are used to realise security objectives. In relation to other services, a service usually has **ensured** and **required** security properties implementing the security objectives. In delivering its functionality and assuming its required security properties are satisfied by other relevant services, the service in turn ensures certain security properties.

The security-oriented service composition involves defining and publishing high level security objectives of the application at both the system or composition level, and the service level. The detailed security properties for a service are kept private to the service in the
service’s interest (e.g., for protection) and are used during composition negotiation (see below).

3.1 Defining system-level security objectives

At the architecture level, system-level security objectives are defined and specified along with the system architecture in such a way that they are publicly available along with the published functionalities. To illustrate this concept clearly we give three examples of defining system-level security objectives. In the example healthcare system given earlier: (a) patient names would be kept confidential in all transactions between services; (b) prescriptions must carry the verifiable identity of the issuing services; (c) diagnosis reports must not be modified or tampered with by any unauthorised services. These security requirements should be codified into security objectives in a way that they are machine readable, and can be reasoned about by other entities. We represent a security objective in the following form:

\[
\text{security_objective} (\text{entity})
\]

where \text{security_objective} can be any security goal such as confidentiality, integrity, non-repudiation, authorisation, authenticity, availability, and so on. The \text{entity} can be any business object such as a report, a file, a person, a data, etc. By applying this simple format to the above three examples, we have the following three security objective descriptions:

(a) confidentiality(patient name)
(b) non-repudiation(prescription)
(c) integrity(diagnosis report)

These security objectives are associated with specific functionalities offered by the system, and spell out the ultimate security goals about the entities of these functionalities, but ignoring the lower level details.

3.2 Defining service-level security objectives

An individual service has its own security objectives. These objectives are associated with specific service functionalities, and can be classified in the same way as the system-level security objectives. As such, the service-level security objectives are also similarly represented, published, read and used for reasoning by other services.

3.3 Defining service-level security properties

In addition to security objectives, we also define and capture detailed security properties of services. These properties are required to achieve the security objectives, and are often kept private but are exploited for service negotiation to satisfy compositional security requirements (see the next section for further details). We formulate a security property as follows

\[
\text{security_property} (\text{element}_1, \ldots, \text{element}_n)
\]

where \text{security_property} represents a particular type of security function such as encryption, digital signature, key generation and so on. By applying this format to our examples, we have the following security property specifications:

(a) encrypted(patient name, shared key)
(b) signed(prescription, secret key)
(c) encrypted(diagnosis report, public key)

The above properties state that the patient name is encrypted with a shared key, the prescription is signed with a secret key and the diagnosis report is encrypted with a public key. Most of the security objectives and properties can be defined on the basis of the functional security requirements of the international standard on Common Criteria for Information Technology Security Evaluation (see [4] for more details).

For illustration purpose, Figure 2 shows a simple example of the artefact model of security objectives at the architecture level, as well as security objectives and properties at the service
level for a healthcare system. The example has a very simple system architecture consisting of three service types offered by three different service providers. Note that to simplify the discussion, the relevant functionalities of the system and services are directly referenced as services in the remainder of the paper when there is unlikely any confusion. The system as a whole delivers a service called \texttt{HealthCare()}. It provides a patient with a diagnosis report based on his/her test results such as blood test and CT scan. The system also offers a prescription based on the diagnosis report from another specialist service. This system-level service has three associated security objectives that are made public.

The figure also shows that each service in the architecture offers a service, namely, \texttt{SetTestResult()}, \texttt{SetDiagnosis()}, and \texttt{SetPrescription()}; and each has a security objective, i.e., confidentiality of the patient name, integrity of the diagnosis report, and non-repudiation of the prescription respectively. These objectives are published along with the services by the service providers. However, examples of security properties that are defined at the individual service level are as follows.

\begin{itemize}
  \item $P_{\text{SetTestResult}} = \text{encrypted(patient name, shared key)}$,
  \item $P_{\text{SetDiagnosis}} = \text{encrypted(diagnosis report, public key)}$,
  \item $P_{\text{SetPrescription}} = \text{signed(prescription, secret key)}$.
\end{itemize}

Figure 2. A simple example of defining and specifying security objectives and properties

Note that the actual composition and instantiation techniques of Figure 2 are explained in the next section.

4. Security contract negotiation for composition

Based on the definition of security objectives and properties as discussed in the previous section, we address in this section automated security-aware composition involving service selection, negotiation of security properties, checking of security compatibility and goal satisfaction during the instantiation of service compositions.
4.1 Selecting candidate services

The first step during architecture instantiation is selection of a collection of candidate services that can potentially satisfy the security objectives of the overall system. The candidate services are selected on the basis of conformity of their security objectives with the system-level security objectives. In the example from Figure 2 the services `SetTestResults()`, `setDiagnosis()` and `SetPrescription()` are defined with the individual security objectives `confidentiality(patient name)`, `integrity(diagnosis report)` and `non-repudiation(prescription)` which are consistent with the security objectives of the system-level service `HealthCare()`. It should be noted that in general the candidate services can have more than one security objectives and there can be a number of alternative candidate services with equivalent security objectives. The candidate service instances are selected based on the compatible security objectives between services. Typically, each service might have a number of security properties that, individually or in combination with others, can fulfil its security objectives. The security properties (required and ensured) of a service need to match with those of the interrelated services in a composition and it may be that only some of their combinations can collectively satisfy the system-level objectives. In the next step, negotiation between the candidate services over their security properties is performed to reach a satisfactory service-oriented composition.

4.2 Negotiation based on security properties

In general, negotiation is a process of finding an agreement among two or more parties that exchange their preferences until a mutually satisfactory solution is found or a termination condition is reached resulting in an unsuccessful negotiation [8, 10]. Negotiation allows the participants to explore different options and trade-offs, and gradually reach a consensus that satisfies both their individual and global objectives. In our framework we propose automated negotiation agents [7, 10] to support services in negotiating security properties in security-aware composition.

Before a negotiation commences, each candidate service orders its security properties (and their combinations) according to their contributions towards satisfying all the security objectives at the service and system levels. It reduces the number of possible instantiations of the service with different security properties through eliminating the security properties that are not compliant with the system objectives (and possibly with the security objectives of other services). All security properties and their combinations are represented in a tree-like structure where the nodes are categorised into groups from the most preferable to the least preferable (but still acceptable) alternatives for instantiation with the service. Each node can also have associated conditional factors such as under which circumstances a particular node would be acceptable with the security properties supported (ensured or required) by other services.

Negotiation between the services is carried out through the iterative exchange of the preferred security properties supported by the candidate services in a composition. It is governed by a negotiation protocol that specifies the rules of interaction between the services. In our framework we adopt an iterative negotiation protocol involving a number of bi-lateral negotiations between pairs of the candidate services over multiple security properties simultaneously [10]. After negotiation the bilateral agreements are checked for their collective system-level security compatibility.

To illustrate the main principles of the negotiation process, let us consider a simple example in Figure 3. It presents two services `setDiagnosis()` and `setPrescription()` from our previous example. The services have the lists of their private security properties `p_{setDiagnosis()}` and `p_{setPrescription()}`, respectively, that are ordered according to their preference levels (note that in general they can be combinations of multiple security properties). For simplicity, consider that the services exchange their most preferable security properties `p_{setDiagnosis()}` and `p_{setPrescription()}`.
and $p_1^{\text{setPrescription}}$, which do not match with each other (i.e., they are not mutually acceptable to the services). Then the services exchange their second preferences for the security properties, i.e., $p_2^{\text{setDiagnosis}}$ and $p_2^{\text{setPrescription}}$, which also do not match with each other. However because the first proposal from the service $\text{setDiagnosis}()$ matches with the second preference of the service $\text{setPrescription}()$, it can accept that arrangement. As a result the services reach an agreement on $p_1^{\text{setDiagnosis}} \leftrightarrow p_2^{\text{setPrescription}}$ involving $\text{signed}(\text{prescription, secret key})$ and $\text{encrypted}(\text{diagnosis report, public key})$ that both services can together ensure and provide.

![Figure 3. Security properties of services and negotiated contracts](image)

In general the process of exchanging proposals between the services can continue depending on the negotiation strategies and acceptance levels of the services. If the negotiation between two services is unsuccessful, i.e., the services are unable to reach a satisfactory agreement, they can switch to other candidate services with equivalent security objectives and continue negotiation. The services can also choose to negotiate with different candidate services and reach a number of alternative agreements subject to making final contract. For example suppose there are two alternative services $\text{setPrescription1}()$ and $\text{setPrescription2}()$ that are candidates for composition with the service $\text{setDiagnosis}()$. Then if two agreements reached between them consist of $p_2^{\text{setDiagnosis}} \leftrightarrow p_4^{\text{setPrescription1}}$, and $p_1^{\text{setDiagnosis}} \leftrightarrow p_2^{\text{setPrescription2}}$ respectively, then the second agreement is clearly more preferable for the service $\text{setDiagnosis}()$ and also likely to be better for the composition. A decision to make a composition depends primarily on what satisfactory level of security a service supports, what satisfactory level of security it gets in return, and what is the trade offs between these two in terms of security achievement for an eventual contract.

The negotiation-based framework promises individual services much flexibility and enough autonomy to consider and re-consider alternative security properties, and allows them to re-position themselves during security-oriented negotiation with other services. It also allows the services to assess and re-assess various security alternatives automatically satisfying the security objectives of all the services and the overall system, hence promoting satisfactory security composition. It enables the services to engage in a meaningful negotiation with others in order to achieve a mutually satisfactory security composition without compromising security requirements of all parties.

### 4.3 Checking system-level security compatibility

After negotiation the security properties of the composite system are verified with the systems-level security objectives. It is necessary to check and validate that low level security agreements match the systems-level security requirements. Irrespective of the number of
individual services involved in a composite system, they collectively must comply with the system-level security objectives. It is important to ensure automatically that all security compositions are consistent with the system-level security objectives. We use model checking for this purpose. Model checking is a technique that relies on building a finite model of a system and checking that a desired property holds in that model [3]. The major advantage of model checking is that it is completely automatic with reasonable performance. After successful model checking the agreements can be contracted; otherwise, negotiation resumes.

4.4 Creating security contracts

In the final stage of the architecture instantiation the selected services are contracted to ensure provision of the agreed security properties during their execution. Based on the negotiations, each service is assigned a contract that spells out its service level agreement (SLA). The security contracts are managed over the life time of the composition. It involves creation, storage, monitoring, control and termination of the contracts that are common functions of SLA management in service-oriented compositions and as such are out of the scope of this article.

5. Security contract re-negotiation for evolution

In a dynamic environment, the underlying composition will inevitably evolve in the ways of service contract changes, service upgrade, service addition and service removal. Such evolution may be due to the changes in system requirements, in the operating environment, and/or in the availability and quality of services. For example, changes in security threats or new business rules may prompt additional system security requirements, and the degradation of a service’s quality may no longer fulfil the original security contract. In general, the initiators for system evolution are either humans (e.g., for changed security requirements) or service monitors (e.g., for the actual quality of services delivered).

As a consequence of system evolution, a service composition may require adaptation. For example, some original composition contracts between individual services may no longer be complied with. Certain system-level security properties may no longer be guaranteed. New system-level properties may be required. Certain services may need to be replaced with new ones. Even the original system architecture may need to be changed. In any case, compatibility between individual services in the system needs to be achieved or re-established, and system-level objectives need to be delivered. All these need to be reflected in a set of contracts that are either existing, adapted or new. Again, the satisfaction of security requirements is part of this adaptation process.

In general, system evolution results in one or more of the following scenarios in terms of establishing service contracts:

1. Adapt an existing service contract under different terms.
2. Establish a service contract with a newly selected service.
3. Change the existing composition architecture, and consequently instantiate it by adapting certain existing service contracts and establishing new contracts with new services.

As in the case for initial service composition, architectural change for the service composition should be primarily the responsibility of the system architect. Once the new evolved architecture is determined, it then can be re-instantiated taking into account the existing service contracts under the previous architecture. As such, we focus on the issues of automated security contract re-establishment that in general can involve (i) re-visiting the existing contracts, (ii) re-considering the candidate services, (iii) re-negotiating the security agreements, (iv) re-verifying the system-level security properties, and finally (v) re-writing the contracts.
Re-negotiation can be triggered by changes in the security requirements at the service and/or system levels. In any case the existing security contracts are analysed to determine the affected services and identify the required contracts for re-negotiation. If the security objectives of the existing services satisfy the new security objectives at the system level, re-negotiation on the security properties is carried out in a similar way as the initial negotiation described in Section 4. Let us consider the earlier example from Figure 3. Suppose that the service $setDiagnosis()$ is not able to support the security property $p_1^{setDiagnosis()}$ initially contracted with the service $setPrescription()$, which violates the contract $p_1^{setDiagnosis()} \leftrightarrow p_2^{setPrescription()}$. In order to re-establish the contract the services engage in re-negotiation through the exchange of the currently preferred security properties and find a new agreement, e.g., $p_4^{setDiagnosis()} \leftrightarrow p_1^{setPrescription()}$ following the principles described in Section 4.2. It should be noted that if more than one service is affected then re-negotiation is performed to find the corresponding new agreements, which then are validated for system-level security satisfaction and contracted as described before. In the case of unsuccessful re-negotiation a new service with the equivalent security objectives is selected for re-establishing a similar contract. In our example, the service $setDiagnosis()$ can be replaced with another service, e.g., $setDiagnosis2()$ for negotiation of a new security agreement with the service $setPrescription()$. In a more general case, a number of equivalent candidate services can be selected for negotiation of alternative security agreements, of which the best one is subsequently contracted.

When the security objectives of the contracted services do not conform to the changed system security objectives, new services are selected with the conforming security objectives. These services can be additional to, replacement for and/or cause removal of some existing services. Their selection follows the principles of the minimal impact on the existing security composition. The new security agreements are negotiated with new services and the existing services affected by the change. It should be noted that the impact on the services can be direct or indirect. The services are directly affected when their security objectives or contracts are explicitly linked to the new security requirements. However as a result of changing contracts during re-negotiation some other contracts may become invalid and require re-negotiation. Thus the services involved in those contracts are indirectly affected but in effect need to be engaged in re-negotiation of their contracts to collectively satisfy the new security requirements. For example, consider the scenario in Figure 2 involving three interrelated services $SetTestResult()$, $SetDiagnosis()$, and $SetPrescription()$ and two contracts $p_2^{setTestResult()} \leftrightarrow p_1^{setDiagnosis()}$ and $p_1^{setDiagnosis()} \leftrightarrow p_2^{setPrescription()}$. Assume that the security requirements for service $SetTestResult()$ changes. Then if the contract between the services $SetTestResult()$ and $SetDiagnosis()$ changes due to re-negotiation then the contract between the services $SetDiagnosis()$ and $SetPrescription()$ needs also to be re-negotiated even though the last service is not directly affected by the original requirement change. In a more general case re-negotiation can involve a number of alternative services, which results in an alternative set of security agreements. The agreements can then be contracted for a new composition after re-checking system-level security properties.

6. Conclusion

In a world where few IT/software systems stand alone, these systems are mostly interconnected via open/hostile networks. System security is of paramount concern in such a context. As the systems and services are independently developed but need to be integrated or composed with each other, revealing their security requirements and provisions with security objectives and properties is essential to gaining credible security assessment for the system or service compositions.
As we move towards dynamic system composition as exemplified by Web and Grid services, automated negotiation is critical to achieve and adapt service compositions on-the-fly. Services may become available or unavailable without notice and service qualities (including security) may change by the minute. Maintaining a service composition with defined functional and quality goals is a main challenge. Only with automated service composition and evolution we can realize the full potential of service oriented computing.

We note that our approach is applicable distributed computing technologies in general, including agent-based systems, component software, mobile networks as well as Web and Grid services. We also note that our approach calls for and relies on the standardization of the format and semantics of security objectives and properties for services and service compositions so that the security definitions are consistent across all service providers through the use of uniform notations. As such, our work contributes to the current standardization efforts in all the relevant areas.

References


Biography

Jun Han is Professor of Software Engineering in the School of Information Technology at Swinburne University of Technology, Australia, where he directs research into component software and enterprise systems. His research interests include component-based software systems, enterprise systems engineering, and software engineering tools and methods. Jun received his BEng and MEng degrees in computer science and engineering from Beijing University of Science and Technology, and PhD degree in computer science from the University of Queensland. He is a member of IEEE Computer Society. jhan@it.swin.edu.au

Khaled M. Khan is a senior lecturer in the School of Computing and Information Technology at University of Western Sydney, Australia. His research interests include software component, software maintenance, and software metrics. He received his bachelors and master degrees both in computer science and informatics from the University of Trondheim, Norway. He also holds a second bachelors degree from the University of Dhaka, Bangladesh. He is a member of IEEE Computer Society. k.khan@uws.edu.au

Ryszard Kowalczyk is Professor of Intelligent Systems in the School of Information Technology at Swinburne University of Technology, Australia and Director of the Centre for Intelligent Agents and Multi-Agent Systems. His research interests include autonomous agent negotiations, coordination and adaptation in complex dynamic systems. Ryszard received his MSc and PhD degrees in engineering and computer science from University of Zielona Gora and Silesian University of Technology in Poland, respectively. rkowalczyk@it.swin.edu.au