

Specifying Security Goals of Component Based Systems : An End-User Perspective

Khaled M. Khan
Department of Computer Sc. and Engineering
Qatar University
PO Box 2713, Doha Qatar
k.khan@qu.edu.au

Jun Han
Faculty of ICT
Swinburne University of Technology
Melbourne, Australia
jhan@it.swin.edu.au

Abstract

This paper treats security from a software engineering point of view. Security issues of software components are usually handled at the two levels of development abstractions: by the security experts during the component design, and by the software engineers during the composition of an application system. Security experts identify the threats of the component, define the security policies and functions. On the other hand, the software engineers are more interested in the compositional impact and conformity of the security properties designed and implemented by the security experts. This paper identifies a third level of abstraction: security from the end-users' perspective. This paper argues that the end-users of the system should know the specific security objectives actually achieved at the system-level. This paper makes the following three specific contributions in this regard: (i) a need for a separate view of security at the end-user level; (ii) the formulation of security goals; (iii) the derivation of security goals for automatic processing.

1. Introduction

A component based software system is composed of several independent software components developed by third parties. The components are usually available from distributed sources for runtime execution. The security issues of 'third-party' software components and their compositions with application systems have become a major challenge. In a highly fluid distributed environment, software engineers are virtually forced to compose systems with third-party services of which they have only partial or no knowledge about their underlying security properties. When components are acquired from the Internet and composed with an application system by the software engineers, it is unclear to the end-users what type of ultimate secu-

rity objectives are achieved with the independent components. End-users are more interested in the high level security goals achieved with the components.

In a component based system, we need to achieve both the security compatibility between interacting components and the security objectives of the entire system. When the entire composite system is running and providing meaningful services to the users, the security objectives achieved (as opposed to implemented details of security) need to be characterised for the end-users. End-users are usually not interested in the technical details of the security jargons. It is also not enough to claim that a composite system is 'secure'. The term 'secure' is over used and somehow misleading because it does not state the specific type of security achieved. Let us consider an example. A composite software processing online credit card payment is claimed to be 'secure'. The claim does not spell out whether (i) the card number will be kept confidential; (ii) the amount will be confidential; (iii) the amount will not be tampered, (iv) the payment beneficiary cannot deny later that the payment was received; (v) the card holder also cannot deny the payment at a later stage; or some other security issues. The paper proposes a framework that will allow the software engineers to specify the specific security objectives achieved in a easy-to-understand term.

The security of a composite system can be viewed from three related perspectives: (1) *the security experts' perspective*, which focuses on the technical details of security of the component system such as cryptography, threats and security policies; (2) *the software composers' perspective*, that is, the actual *implemented security properties* and their impact on the composite system; (3) *the end-users' perspective*, the ultimate *security goals* which are achieved in association with a particular functionality provided by a collection of components. In order to achieve the latter perspective, we need the first two perspectives satisfied. Security achievements can only be derived from the actual security

properties realised in a composition. The major focus of this paper is on the third one.

To characterise the security goals of component based system, our approach is based on the following: (i) define high level security goals based on the security properties defined in Common Criteria (CC)[10] (*end-users' perspective*); (ii) identify the actual security properties of the participating components used in a composition (*software engineers' perspective*); and (iii) formulate rules to characterise the security goals specific to a composition using the actual security properties defined by the *security experts*.

This paper is organised as follows. Next section gives a quick introduction to the security property characterisation of software components, that is, the software engineers' view. Section 3 provides fundamental concepts of security goals. The procedures to define the security goals are presented in section 4. Section 5 shows the applicability of the approach with an example. A brief discussion on related work is given in section 6. Finally we conclude in section 7.

2. Security: software engineers' perspective

This section enables readers to familiarise the terminology we use in this paper. First we very briefly describe the fundamentals of our security property characterisation work proposed in [5] which provides readers necessary background knowledge to understand the major theme of this paper. Security properties of independent software components and their composition are used by the software engineers during the composition. Our proposed approach is based on the implemented security properties.

The security properties achieved at a compositional-level between two components are referred to as compositional security contracts (CsC). In other words, a CsC is based on the compatibility between the required security properties of a component and the ensured security properties of another components and vice versa [6]. A CsC is always established between two participating components satisfying each others' security requirements.

To automate the security characterisation and to reason about the compatibility of compositional security contracts, our language support [5] is based on *logic programming* and *BAN logic* [3]. In this approach, security properties are represented and expressed in a symbolic notation based on *atoms*. An *atom* consists of a predicate name (a security function) with a tuple of *variables* or *constants* (elements). Atoms usually express relationships between elements. An atom is of the form $p(t_1, \dots, t_n)$ where p is a predicate symbol and t_1, \dots, t_n are terms [11]. A term could be either a variable, a constant, or a function representing an element such as an entity, data, a password, a key. *Variables* are used to generalize objects such as X for which the

inference rules find or substitute an element.

The predicate name of an atom represents a security property such as *encrypted*, *key-generated*, *signed* etc. An example of an atom is $encrypted(p,x)$. It states that an element identified as p encrypts the object x . The entity p could be a human, a program or a component. The object x could be a message, a piece of data, or a file. Another example of an atom is $signed(p,x,k)$. It states that the entity p digitally signs the object x with the key k .

In addition to the representation of security properties and their associated elements, inference rules are used to check the conformity of the security properties represented in atoms. A rule is composed of a head and a body. The symbol (\leftarrow) is read as 'is derived from'. The left hand side of \leftarrow (the head of the rule) is the conclusion or consequence that represents the ensured security properties. An ensured security property is a conclusion of an expression. The right hand side of \leftarrow (the body of the rule) is called a premise or condition [1]. It represents the required security properties in our characterisation context. Examples of such rules are:

$$\begin{aligned} signed(tax, k^{-y}) &\leftarrow encrypted(form, k^y). \\ CsC(x, y) &\leftarrow signed(tax, k^{-y}). \end{aligned}$$

In the above expressions, the first rule states that the object tax is digitally signed with the private key (k^{-y}) of y if the object $form$ is encrypted with the public key (k^y) of y . Note that the terms k^{-y} and k^y represent private and public keys of the component y respectively. The second rule states that a compositional security contract (CsC) between entities x and y is established if tax is digitally signed by y with its private key k^{-y} .

Rules make an inference with the knowledge associated with the security properties. If each atom in the body of a rule is *true*, then the inference rule concludes a *true* to the head. Rules capture the security facts of the problem domain. A set of inference rules are applied to prove whether a CsC is achievable or not. Based on these fundamentals, in the following sections we present the *security goal* characterisation of composite systems which is the main theme of this paper.

3. Security: end-users' perspective

A security goal is the ultimate security objective of a security property. It is achieved as a result of the implemented security properties in the components. For example, a security goal could be defined as the *integrity* of an object or a piece of data; *confidentiality* of a message; *authentication* of an entity or user; *authorisation* for an operation on certain object or data; or *nonrepudiation* of a message and so on. From an end-users' perspective, the ultimate yet specific security goals achieved in a

particular functionality can be provided by a collection of independent components in a composition. The security goals of the composite system expose the ultimate security objectives achieved in the system. Let us first consider an example. Assume that a software is composed of two independent components. The security properties used in this composition are shown below specified in a format described in Section 2:

Example - 1a. (Software engineers' view)
 $composition(x, y) \leftarrow encrypted(data, y),$
 $digitally_signed(data, y).$

This example specifies that a composition is established between two components identified as x and y based on the fact that the $data$ is encrypted and digitally signed by the component y . These two required security properties are internal to the participating components. However, at the end-user level such detailed technical information about the security properties such as *digital signature* and *encryption* may not be necessary due to two main reasons: (i) users may not understand the technical details of security, thus such information is irrelevant; (ii) technical details of security functions are not required at the user level. The users may be more interested in the specific security goals achieved from the underlying compositions.

In our above example, the possible security goals of the two security properties (i.e. encryption, digital signature) could be the *authenticity* of the component y , and the *confidentiality* of the object $data$. Users may rather know which security goals are to be achieved in a particular functionality. The example in 1a does not state what type of security goal is actually achieved in the composition between components x and y . We argue in this paper that each concluding composition should specify at least one underlying security goal which is easily understood by all end-users. We need to spell out the underlying security goals achieved by the actual security measures implemented in the composition.

In Example 1a, we can specify several security goals of this composition. Firstly, the *confidentiality* of the object $data$ is achieved with the atom $encrypt(data, y)$; and secondly, the *authenticity* of the entity $data$ is established with the property $digitally_signed(data, y)$. A *nonrepudiation* security goal is also achieved by this property. We now redefine Example (1a) as:

Example (1b).
 Implementation view (security properties):
 $composition(x, y) \leftarrow encrypted(data, y),$
 $digitally_signed(data, y).$

The corresponding end-users' view (*Security Goals*):

$$confidentiality(data) \leftarrow encrypted(data).$$

$$authenticity(Y) \leftarrow digitally_signed(data).$$

$$nonrepudiation(Y) \leftarrow digitally_signed(data).$$

The last three characterisations in Example 1b state the ultimate security goals achieved in this composition. The atom $confidentiality(data)$ characterises that the object $data$ is confidential while being transferred between the entities x and y because it is encrypted. Similarly, the atoms $authenticity(y)$ and $nonrepudiation(y)$ characterise the authenticity and nonrepudiation of the entity y respectively because y digitally signs the $data$.

From the end-user's perspective, a security goal characterises the specific security behaviour that users of the system can easily understand from the enclosed system. An achieved security goal is mostly dependent on two issues: (i) the technical details of the security properties implemented in the individual component (i.e. the software engineer's view); (ii) how an application system is composed and used by the end-users in terms of functionality (service).

4. Defining security objectives

To facilitate an understanding of the composite security properties in terms of their objectives, we propose a format as:

$$goal(X) \leftarrow security_property(X, Y, Z)$$

The security goal of the term X is characterised at the left-hand side of the arrow \leftarrow in relation to the detailed security properties specified in the right-hand side of the arrow. The detailed security properties are the technical representation of the actual implemented security properties. The term X must be used in the security properties of the two participating components in the composition. X could be an entity such as a file, a piece of data, a person, a component, an object etc. For example, a composition is established between two components if a *certificate* is digitally signed. In this case, the security goal could be the objective which is achieved from the fact that the *certificate* is digitally signed. The goal could be authenticity and/or the nonrepudiation of the *signer*. A composition may have more than one security goal depending on the security properties involved with the composition. The goal is used here to show the user the actual security objective achieved as a result of the composition formed between two components conforming each other's security requirements.

A goal is the ultimate security achieved with the implementation of the associated security properties of a

composition. The predicate *goal* could be defined with any of the following: (1) *confidentiality*: the secrecy of an object or message; (2) *integrity*: the untampered value of an object; (3) *availability*: an object, service, or operation is available to the authorized users; (4) *authenticity*: confirms the identity of an entity; (5) *nonrepudiation*: the undeniable evidence of actions performed by an entity; (6) *auditability*: the auditing and monitoring of security events; and (7) others.

A security property might have more than one term, and hence may have multiple security goals. A goal must reflect the achievement of the security property involved. Note also that a security property may have more than one security goal. Consequently, the rule to identify the security goal is: (1) identify and associate a security goal to each type of security property, (2) the term used in the atom of the goal must be a term used in the corresponding security properties of the composition. For the first rule, we use the security specifications defined in the ISO/IEC 15408 Common Criteria (CC) [10]

The Common Criteria (CC) [10] provides a common set of requirements of the security functions of a system, and a set of evaluation measures. CC describes the security behaviour or functions expected of an IT system to withstand threats. The security functional requirements in CC consist of eleven ‘classes’ for generic grouping of similar types of security requirements: security audit, communication, cryptographic support, user data protection, identification and authentication, security management, privacy, protection of system security functions, resource utilisation, system access, and trusted path and channels.

The members of a class share a common focus while differing from each other in coverage of security aims. Each of these classes comprises members called *families*. A ‘family’ is a grouping of sets of security requirements sharing a common security objective but differing in emphasis and rigour. A family is based on ‘components’ (not software components) which are the smallest sets of security requirements.

The Common Criteria definitely gives a comprehensive catalogue of high level security requirements and assurances for software products. It harmonises the European ‘Information Technology Security Evaluation Criteria (ITSEC)’, the Canadian ‘Trusted Computer Product Evaluation Criteria (TCSEC)’, and the United States ‘Federal Criteria (FC)’. Since CC is essentially a natural language description of security properties without any representation for automatic processing, we have formulated these properties in the machine readable format described in section 2. We examined each security class and their characteristics and redefine them into a format suitable for our automatic language support proposed in [5]. We further develop corresponding security goal for each formulated

Table 1. Security goals for cryptographic functions

Ensured properties	Descriptions
$owned(K, P)$	Confidentiality
$encrypted(X, K)$	Confidentiality
$signed(X, K)$	Authenticity, Integrity, nonrepudiation
$believes_produced(P, Q, X)$	Authenticity, nonrepudiation

security property as briefly outlined in the following subsections.

4.1. Cryptographic support

Cryptographic techniques used in computing provide for the confidentiality of the data exchanged between entities and authenticity of the sender of the data. Table 1 outlines the security goals of a few cryptographic properties defined in CC. Due to the space limitation, we show only a handful of properties along with their corresponding security goals. Some properties address more than one security goal as shown in the table such as $signed(X, K)$, because it relates to the authenticity of the sender of a message as well as the integrity of the message if encrypted with the digital signature and nonrepudiation. Once the message is signed, the sender cannot deny it.

4.2. User data protection

The protection of user data is mostly concerned with data integrity and confidentiality. Whatever security technique we use to secure the user data, it has two major security goals: (1) *data integrity*, which means that the data has not been modified by any unauthorised entity; (2) *data confidentiality*, which means that the data has not been observed by other entities. In Table 2, we list security goals of some security properties related to user data protection defined in CC.

4.3. Identification and authentication

The security goals of this class are related to the authentication of entities and the authorization of operations. The authentication technique confirms the identity of an entity. The authorisation processes the grant or reject of a request by an entity to perform certain operations on an object. Table 3 shows some of the related security goals of this class.

Table 2. Security goals for user data protection

Ensured properties	Descriptions
$known(X, P)$	Availability
$sees(P, X)$	Availability
$believe_sees(P, Q, X)$	nonrepudiation, Availability
$storage_integrity(X)$	Confidentiality, Integrity

Table 3. Security goals for identification and authorisation

Ensured properties	Descriptions
$user_identity(P, R, W)$	Authenticity
$acl(P, R, X, O^c)$	Availability, Authorization
$authenticated(Q, X, O)$	Authenticity, Authorization

4.4. Security audit and resource utilisation

All properties defined in this class are related to the security audit. These have more to do with the security management and protection mechanisms of the system as a whole. Some security goals in this class are listed in Table 4. This class deals with the security audits, none of

Table 4. Security goals for security audit and resource utilisation

Ensured properties	Descriptions
$audit_response(P, T, E)$	Auditability
$generates_audit(P, X, E)$	Auditability
$analyses_security(P, M, X)$	Auditability
$selects_audit(P, E, X, C)$	Auditability

the properties concern security goals such as confidentiality, integrity or authenticity. Note that these are considered the functional rather than non-functional properties of the system. Security auditing is a service a system can offer as opposed to a security attribute of the system.

4.5. Privacy

All properties in this class address the confidentiality of objects or operations. Table 5 shows the security goals

Table 5. Security properties for privacy class

Ensured properties	Descriptions
$believes_anonymity(P, O, X)$	Confidentiality
$unlinkability(P, O, X)$	Confidentiality
$unobservability(P, O, X)$	Confidentiality

along with the corresponding security properties defined for this class.

4.6 Trusted channels

The objectives of the security properties in this class are largely data confidentiality and data integrity. Table 6 shows the security goals of this class in which both properties are concerned with confidentiality and integrity: These essentially deal with trust.

Table 6. Security properties for trusted channel

Ensured properties	Descriptions
$trusted_channel(P, Q, M)$	Confidentiality, Integrity
$trusted_path(P, Q)$	Confidentiality, Integrity

5. An illustration with an example

This section describes a distributed computation system for business tax calculations as an example to illustrate our proposed approach. A dispatcher component identified as d used by the business community to dispatch their tax forms can be dynamically assembled with a tax calculating software component g over the open communication network. Users can also deposit money to a banking component e . The payment can be two types: either the tax payments owing to g by d , or d just makes a deposit for its own purposes. The component credit provider f gives a credit certificate to d if d produces a payment receipt obtained from e .

We assume each of the contracting components is spatially separated, and communicates with the others via a completely connected point-to-point network. We also assume that the supporting infrastructure, for example, public communication channels, the connectors/middleware, operating system and communication protocols, exists to support the application. Figure 1 illustrates the basic topology of this dynamic composite system. The arrows in the figure denote bidirectional transmissions between the two components concerned.

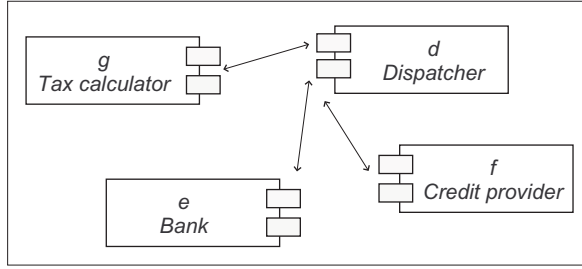


Figure 1. A system topology for tax and payment computation

This section motivates our presented approach with four different compositional scenarios for the system described above. Scenarios are used to capture four different compositional contracts of the system. Each of the scenarios, with their functionality offered at the user level, is discussed with the corresponding security goals derived from the underlying compositions.

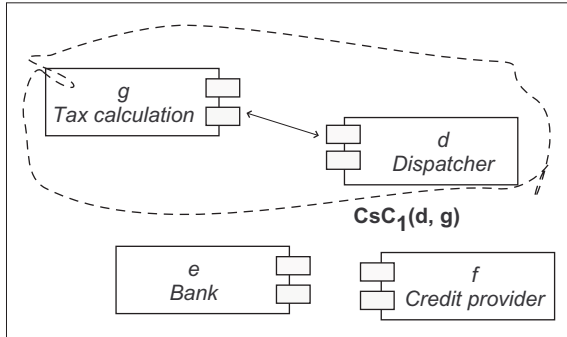


Figure 2. A compositional contract between components *tax calculator* and *dispatcher*

5.1. Scenario 1

In this scenario, we assume the functionality of the system is $processTaxReturn()$ which is achieved with the composition between the components d and g as depicted in Figure 2. On the other hand, Figure 3 shows the software engineers' view of the security properties in this composition. Briefly, these properties state that the tax form is encrypted with a key which belongs to d ; and the tax report is digitally signed by the component g if the tax form is signed by d . The component d believes that the tax

$encrypted(tax_form, k).$ $owned(k, d).$ $signed(tax_report, k^{-1}) \leftarrow encrypted(tax_form, k),$ $believes_produced(d, g, tax_report) \leftarrow$ $signed(tax_report, k^{-1}),$ $owned(k^{-1}, g).$
$CsC_1(d, g) \leftarrow believes_produced(d, g, tax_report).$

Figure 3. Security properties of the composition between d and g

report has been produced by g if it is signed by the private key of g .

Based on these properties and the security goal tables of the classes, we define following three security goals derived from the rule $CsC_1(d, g)$. The *authenticity* and *nonrepudiation* goals are achieved from the rule $believes_produced(d, g, tax_report)$. In the body of the rule, the object tax_report is signed by the entity g . The entity g cannot deny that it produced the object tax_report to the entity d . Therefore, this predicate establishes a *nonrepudiation* security goal. Similarly, since tax_report is signed with g 's private key, entity d can verify the authenticity of g with its corresponding public key. Hence *authenticity* is achieved with this predicate as well. These two security goals are dependent on the *confidentiality* established from the required properties of the rule $signed(tax_report, k^{-1})$ as shown in Figure 3. We conclude that the functionality $f_{processTaxReturn()}^{TaxSystem}$ has three security goals:

$$\begin{aligned}
 &authenticity(tax_report) \leftarrow CsC_1(d, g). \\
 &non_repudiation(g) \leftarrow CsC_1(d, g). \\
 &confidentiality(tax_form) \leftarrow CsC_1(d, g).
 \end{aligned}$$

In other words, users of this functionality know that their tax_form is not seen by any unintended entities (confidentiality), and the tax_report object they receive is from an authenticated entity (authenticity and nonrepudiation) which cannot later deny the fact that it had sent tax_report .

5.2. Scenario 2

This scenario provides the functionality $depositMoney()$ between the components d and e as shown in Figure 4. The detailed security properties of this composition is outlined in Figure 5. Based on the $CsC_2(d, e)$ established in this scenario, the *authenticity* and *nonrepudiation* security goals are achieved from

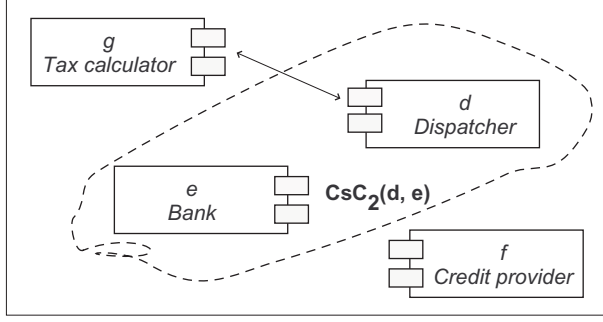


Figure 4. Scenario for the compositional contract between dispatcher and bank

$ \begin{aligned} & encrypted(amt, k1). \\ & owned(k^{-1}, e). \\ & owned(k1, d). \\ & sees(d, ack) \leftarrow encrypted(ack, k1), owned(k1, d). \\ & believes_produced(d, e, ack) \leftarrow \\ & \quad signed(ack, k^{-1}), \\ & \quad owned(k^{-1}, e). \\ & CsC_2(d, e) \leftarrow believes_produced(d, e, ack). \end{aligned} $
--

Figure 5. Security properties of the composition between d and e of a deposit function

the atom $believes_produced(d, e, ack)$. The object ack , representing an acknowledgement, is signed by the entity e .

The entity e cannot deny that it produced the object ack and sent to d . Therefore, it establishes a nonrepudiation security goal. Similarly, since ack is signed with the e 's private key, entity d can verify the authenticity of e with its corresponding public key, so the authenticity of e is established. The *confidentiality* of the objects amt , representing an amount, and ack is achieved with the atoms $encrypted(amt, k')$ and $signed(ack, k^{-1})$ respectively. The security goals of this functionality are:

$$\begin{aligned}
& authenticity(e) \leftarrow CsC_2(d, e). \\
& nonrepudiation(e) \leftarrow CsC_2(d, e). \\
& confidentiality(amt) \leftarrow CsC_2(d, e). \\
& confidentiality(ack) \leftarrow CsC_2(d, e).
\end{aligned}$$

In this functionality, users find that the amount they deposit is not seen by any other entities except e (confidentiality), and the acknowledgement of their deposit

they receive from e is genuine and is, indeed, from e (nonrepudiation and confidentiality).

5.3. Scenario 3

In this scenario, with the functionality $depositTaxReturnDues()$, users can pay bank e their tax debt owed to the tax office g . In this case, $CsC_3(d, e)$ provides *authenticity* and *nonrepudiation* for the entity e . Notice that this CsC is different than the CsC_2 because it requires tax_report from the component g . This CsC provides for the confidentiality of the objects tax_report and amt . See Figure 6 for the security properties of this composition.

$$confidentiality(ack) \leftarrow CsC_3(d, e).$$

$ \begin{aligned} & encrypted(tax_report, k'). \\ & owned(k', d). \\ & owned(k^{-1}, e). \\ & signed(ack, k^{-1}) \leftarrow \\ & \quad believes_produced(d, g, tax_report), \\ & \quad encrypted(tax_report, k'), \\ & \quad owned(k', d). \\ & believes_produced(d, e, ack) \leftarrow \\ & \quad signed(ack, k^{-1}), \\ & \quad owned(k^{-1}, e). \\ & CsC_3(d, e) \leftarrow sees(ack, d), \\ & believes_produced(d, e, ack). \end{aligned} $
--

Figure 6. Composition between d and e for the payment of tax dues

$$\begin{aligned}
& authenticity(e) \leftarrow CsC_3(d, e). \\
& nonrepudiation(e) \leftarrow CsC_3(d, e). \\
& confidentiality(tax_report) \leftarrow CsC_3(d, e). \\
& confidentiality(amt) \leftarrow CsC_3(d, e).
\end{aligned}$$

Notice that this composition ultimately includes three components: d , e , and g . Component g is transitively related in this composition because the tax_report is produced by g and authenticated by g as shown in Figure 7.

5.4. Scenario 4

Figure 9 depicts that three components are involved in this composition. With the functionality in the scenario $getCreditCertificate()$, users get a credit certificate from a credit provider. The composed CsC in this scenario provides security goals such as :

$$confidentiality(credit) \leftarrow CsC_4(d, f).$$

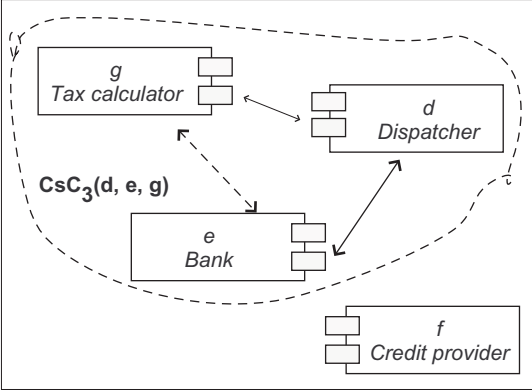


Figure 7. A composition includes d , e , and g

$confidentiality(ack) \leftarrow CsC_4(d, f)$.

In this scenario, it provides confidentiality of the object *credit* certificate with the predicate $not\ sees(I, credit)$ and $encrypted(credit, k2)$. Note that the term I represents a hostile entity. Both security properties have the common security goal. The first property ensures that the *credit* is encrypted. The latter one guarantees that *credit* is not seen by any unintended entities. Figure 8 defines these two rules. The confidentiality of the object *ack* is also established with the predicate $encrypted(ack, k)$. In Figure 9, although

$encrypted(ack, k1).$ $shared(k2, d, f).$ $believes_produced(d, f, credit) \leftarrow$ $encrypted(credit, k2),$ $shared(k2, d, f),$ $not\ sees(I, credit^k), I \neq d, I \neq f.$ $CsC_4(d, f) \leftarrow believes_produced(d, f, credit).$

Figure 8. Security properties of the component d and f

component e is not directly included in the composition, its provided data *ack* with the security property is used in this composition between d and f . Therefore, component e is transitively related in this composition. The scenario of $CsC_5(d, e, f)$ is shown in the figure.

6. Related work

Current practices and research for the security of software components consist of several defensive approaches such as firewalls, trusted operating systems, security

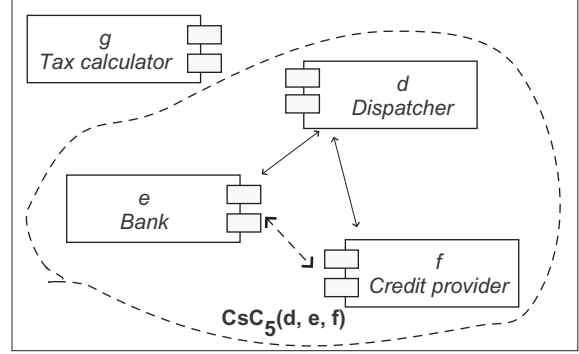


Figure 9. A scenario for the compositional contract between d , e , and f

wrappers, secure servers etc. A thorough analysis of the published work suggests that the issue of protecting the enclosing application system from unsafe and unreliable components dominates this field. It basically deals with finding and removing security flaws of software components.

In Web services area, WS-Security addresses Web service security by using existing security standards and specifications. WS-Security is simply a framework that adds existing mechanisms such as XML Encryption, XML Signature and XML Canonicalization into a SOAP message. XML Canonicalization prepares the XML messages to be signed and encrypted. In other words, WS-Security is a specification for an XML-based security metadata container [9]. What WS-Security actually does is that it enables security properties to be defined independently in the message in terms of composable message elements to exchange SOAP messages securely. From this perspective, WS-Trust, WS-SecureConversation, Security Assertion Markup Language (SAML) are not much different than the WS-Security. None of these, in fact, introduces a new technique or approach to address security composition and specification at the end-users level.

The composable security application (CSA) [12] in the Infrastructure for Composability At Runtime of Internet Services (ICARIS) [7] is a Jini and JavaBeans based implementation deploying point-to-point security associations in order to introduce security services to the application. The security association can be built based on the demands of the client and server for a particular security association. However, it requires human assistance. It does not facilitate automatic dynamic negotiation for a security association and expressing security goals.

The use of Semantic Web technology based on DAML

[4] is a promising technology, although it is still in its infancy. In this regard, the KAoS Policy Ontologies (KOP) [2, 13] designed to analyse policies for agent environments is heavily based on DAML description-logic-based ontology of the computational environment. KAoS services allow enforcement of policies, reasoning of actions based on policies and resolution of conflicts. This work is not security specific, however, the idea could be used to analyse and specify security policies at the end-users level.

The commercial distributed frameworks such as CORBA, DCOM, .Net and Enterprise JavaBeans (EJB) are used for effective composition between objects or components. What is lacking in those technologies is an automatic mechanism for specifying the ultimate security objectives in easy terms which could be understood by any users. Although CORBA and EJB form the basis of many functional aspects of the component model, these tools support only some limited sets of non-functional properties such as persistence and access control. CORBA message specification provides a Quality of Service (QoS) framework to define QoS policies for CORBA application [8]. These do not provide any mechanisms for composing non-functional properties along with the functional composition.

7. Conclusion

This paper has presented an approach to specify the security goals of composed software systems. From the end-users' perspective, security goals associated with different security properties of a composite system are specified based on the actual implemented security properties. The paper has also illustrated different security goals with a running example with four different scenarios. The main contributions of the paper include: (i) identification of a need for the separation of end-users' perspective from the software engineers' perspective in relation to specifying security properties of composite systems; (ii) a formulation technique of security goals based on security properties defined in Common Criteria; (iii) an approach and the rules for deriving security goals from the implemented security properties in a composition using the goal table.

Component developers can use our approach to specify the security goals associated with each security property they employ in the component. They could spell out which security goals are actually achieved with the related security properties. A security goal is ultimately based on the internal security properties (ensured and required) of the components and their compositional properties.

We acknowledge that more work remains to be done especially in the areas of standardization of security properties which could be universally used across the application domain, and the acceptable format of defining rules and rea-

soning engine of the derivation approach. At the same time the scalability of the proposed techniques needs to be examined with more realistic application systems.

References

- [1] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.
- [2] J. M. Bradshaw, editor. *Software Agents*, chapter KAoS: Towards an Industrial-strength Generic Agent Architecture, pages 375–418. AAAI Press/MIT Press, 1997.
- [3] M. Burrows, M. Abadi, and N. R. A logic of authentication. *ACM Transactions on Computer Systems*, 8:18–36, 1990.
- [4] DAML-S. Semantic markup for Web services. Technical report, DAML, 2003.
- [5] K. Khan and J. Han. A security characterisation framework for trustworthy component based software systems. In *IEEE Proceedings of Computer Software and Application Conference (OMPSAC)*, pages 164–169. IEEE Computer Society, 2003.
- [6] K. Khan, J. Han, and Y. Zheng. A Framework for an Active Interface to Characterise Compositional Security Contracts of Software Components. In *Proceedings of the Australian Software Engineering Conference*, pages 117–126. IEEE Computer Society Press, 2001.
- [7] D. Mennie. *An Architecture to Support Dynamic Composition of Service Components and Its Applicability to Internet Security*. M. Eng. thesis, Carleton University, Ottawa, 2000.
- [8] D. Schmidt and S. Vinoski. Object interactions: Overview of OMG CORBA messaging QoS framework. *C++ Magazine*, 2000.
- [9] S. Seely. Understanding WS-Security. Technical report, Microsoft, 2002.
- [10] Standard. Common Criteria for Information Technology Security Evaluation, ISO/IEC 15408. Technical Report v2.0, Nat'l Institute Standards and Technology, Washington, 1999.
- [11] T. Syrjaanen. Implementation of local grounding for logic programs with stable model semantics. Technical report 18, Helsinki University of Technology, 1998.
- [12] V. Tasic, D. Mennie, and B. Pagurek. On dynamic service composition and its applicability. In *Proceedings of the Workshop on Object-Oriented Business Solutions at ECOOP 2001*, pages 95–108, 2001.
- [13] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, M. Bunch, S. Johnson, J. Kulkarni, and J. Lott. KAoS policy and domain services: Towards a description-logic approach to policy representation, deconfliction, and enforcement. In *Proceedings IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, pages 93–98. IEEE Computer Society, 2003.