Security Characterisation of Software Components and Their Composition

Khaled Md. Khan
School of Computing and IT
University of Western Sydney Nepean
PO Box 10 Kingswood
NSW 2747 Austarlia
k.khan@uws.edu.au

Jun Han Yuliang Zheng
School of Network Computing
Monash University
McMahons Road, Frankston
Vic 3199 Australia
{jhan, yuliang}@pscit.monash.edu.au

Abstract

The paper proposes a security characterisation structure of software components and their composition. The structure provides a preliminary modelling of security properties of stand-alone software components and some of their compositional primitives. In this paper, we are particularly interested in security properties related to user data protection of software components. The proposed compositional specification makes an attempt to model the resulting effect between security attributes of two contracting components. The compositional specification structure can capture the results of combined security specifications of two participating components in a contract. Our security specification syntax is based on four compositional elements: identities of contracting components, actions to be performed in a compositional relationship, security attributes supported by components, and resources to be used by other components. The structure is used in an example of secure interactions over a network to illustrate the applicability of the proposed work.

1. Motivation and introduction

With the rapid advancement of web based technology, software systems are becoming increasingly heterogenous in terms of their formation and execution. An application system can be composed with several stand alone software components readily available from various distributed sources for run time execution. In a component-based system, a composition consists of multiple autonomous geographically dispersed components with no shared memory [4]. A software component is autonomous as it has its own executable code, and uses its own data or files. A composition can be dynamic or static depending on how components are connected to each other. Some of the components are downloaded from the Internet, and executed dynamically with the application system.

The use of software component is appealing because they support reusability of code, and far efficient utilisation of network resources [5]. However, as components are reused in far more greater scale, there is a need for application security in component based e-commerce, e-business systems, and other similar critical application systems. Same identical components can be assembled in different ways to construct different types of software systems [7]. When components acquired from Internet, and deployed in an application system, one may have valid reason to worry about the possible security impact of the combined system. In such a highly fluid distributed environment, the compositional impact of security properties of candidate components located in a remote server may be unknown to the user, or untrusted as claimed. The security properties of a component in isolation may differ from the compositional impact of the same component with an application system. In such a scenario, much required trust may not be established on the components as anticipated. We believe that security properties of software components need to be adequately specified in such a way that other components as well as human user should able to read those features, and reason their impact on the combined system before a composition takes place. A component must be very clear about what the security

Proceedings of the 36th International Conference on Technology of Object-Oriented Languages and Systems
0-7695-0875-8/00 $10.00 © 2000 IEEE

contract across the connection with other component is, what security provisions each component provides and ensures, and what would be the ultimate security of the composed system.

There are newer technologies such as JavaBeans, DCOM, OMG's CORBA for effective composition. What is lacking in those technologies is a mechanism on how to specify the compositional security properties of the participating parties. Component technologies such as DCOM, CORBA, and JavaBeans emphasis on defining interface signatures for effective structural communication among components [7]. However, the definition of interface signatures does not describe the expected security behaviour and effect expected from a composition. To inspire reasonable trust on software components, the need for a security characterisation structure along with the interface signature and the certification of the security properties of the components are acute as expressed concerns in [9], [10], [11], [13].

The security characterisation syntax proposed in this paper provides a simple structure for specifying the security properties of individual components on which compositional relationship among components can be established. The characterisation can expose the nature of security considerations of a component to other components. Our main research goal is to combine the security specification of one component with that of another component in order to model a compositional specification. At the composition level, only the higher levels of security instances such as security function interfaces are visible. The security function interface of a given component can be viewed as an instantiation of the component. The result of the composition between two components is a new security property called compositional specification that could be regarded as the specification of the security effect generated from the combined specification of two components. Compositional specification defines rules about the impact of combining two security specification in a given time. The relationship between a client and a server that incorporates quality of service (QoS) property is called a *QoS contract* [6].

In what follows, a brief discussion on the compositional relationship (CR) is presented in section 2. A system description of an example is given in section 3. The example will be used to demonstrate the application of the proposed ideas. The security characterisations of individual component and their compositional primitives are proposed in section 4. We outline the possible application of our proposed work in section 5. We close with further work and a conclusion.

2. Compositional relationship

The security mechanism of a software component may be based on some fundamental logical design parameters such as security policy, security function, and security attributes used in IT products [2]. We use same parameters proposed in Common Criteria [2] to characterise the security properties of software components. Common Criteria (CC) provide a schema for evaluating IT product in general, not directly for software components. We have slightly modified those parameters to accommodate the security properties of software components. The security policy of a software component can be made of multiple security function policies (SFP). Each of the SFPs may have a scope of control that defines under which conditions a *client* component can or cannot perform which *actions* on a *server* component [1]. A security function policy is implemented by one or more security function (SF). Each of the SFs has a scope of control. Each component has a set of component security function interface (CSFI) that regulates how other external entities would interact with the component. A CSFI can be viewed as a set of interaction specifications, whether interactive or programmatic, through which the resources of a component are accessed by other components. In other words, CSFI enforces interface related CSP, and closely related with the structures and semantics of interface signature of a component [3]. In this paper we will use CSFI, SF and SFP along with our proposed security characterisation structure.

A composition between any two given components is governed by one or more *compositional relationship*. The compositional relationship (CR) is determined by the intention and purpose of the contracting components. To make an intended relationship, a client component makes a request to a server component. The request may include operations such as execute message, receive message, send message, create object, and destroy object. The server may honour the request, and grant the intended operations depending on the nature of requested relationships and the security attributes of the client.

The establishment of a compositional relationship is based on four fundamental elements: *(i) originator and recipient of the compositional request, (ii) type of the requested actions, (iii) nature of security attributes, and (iv) type of resources*. In turn, we now briefly explain each of these elements.

2.1 Originator and recipient of the compositional request

The main goal of this element is to establish the identities of the requested-originator and the requested-recipient. There are two main types of entities involved in a composition: *server* components - might be a recipient of a request, and *client* components -might be an originator of a request. However, a server component may also make a request, in such case the component would play the role of a client. A *client* is an entity that may cause an *action* to be performed on *resources* that belong to a *server* component. The terms 'client' and 'server' are used to model the role that a component plays in a particular use context.

2.2 Type of requested CR

The main objective of identifying the 'what' is to establish the purpose or intention of a component for a compositional relationship. The compositional relationship can be determined by assessing the type and nature of actions requested by a component. An *action* is applied on a *server* component by *a client* component such as read data from a file owned by the server, write data on a file or entities that belong to the server, send messages, receive messages, connect request, create an object, destroy an object, and so on. The server component may check its security policy to ensure that the action is permitted to the specified client with the *required security attributes*.

2.3 Nature of the security attributes

Security attributes are used to authenticate components, and authorise their actions. Examples of *security attributes* can be passwords, private keys, secret keys, public keys, shared keys, digital signatures among others. Security attributes can also be used to encrypt and decrypt component resources such as output data and input data respectively. We can classify two different types of security attributes such as *ensured security attributes* and *required security attributes*. A *server* component may have some *ensured* security attributes as well as *required* security attributes for a compositional contract [12]. A *required* security attribute is an invariant in a sense that it is the required property of a component that other interested parties must satisfy during the composition according to the contract [8]. It is a precondition the component must ensure that the security attribute is provided, and its validity is ensured. Similarly, an *ensured* security attribute is a postcondition in a sense that it is the responsibility of the component to maintain the committed security *assurance* during the composition of the contract.

2.4 Type of resources

A compositional relationship depends on the type of resources to be affected by the requested *actions*. A server component may specify different access constraints for different client components for the same actions on its resources. In a runtime composition, the server component may enforce specific controls over access to its various objects and methods.

Based on these four compositional elements we can formulate a unique simple structure that can capture the security requirements and assurances of software components. The security properties of individual component can be specified with a predicate-like structure as

security_function({Entities}, {Actions}, {Security_attributes}, {Resources})

where, *Entities* can be a set of active entities such as client or server components, or both. *Actions* are a set of operations the *Entities* can execute such as read, write, create, destroy, print. S*ecurity attributes*

can be password, private keys, public keys, secret keys, and so on. Each of the arguments contained in the structure is a set of attributes of the argument, and can be further decomposed as

Entities=(entity_ID, entity_URL, entity_developer's_ID)
Actions=(operation_name, operation_duration, operation_frequency)
Security_attributes =(key_standard, key_length, algorithm_used)
Resources=(resource_value, resource_type, resource_range).

To model some simple security functions we may not need the detail decomposition of the structure. Some of the arguments defined in the structure may be considered optional or not applicable in some use context. Figure 1 shows a hierarchical structure of the security specification model.

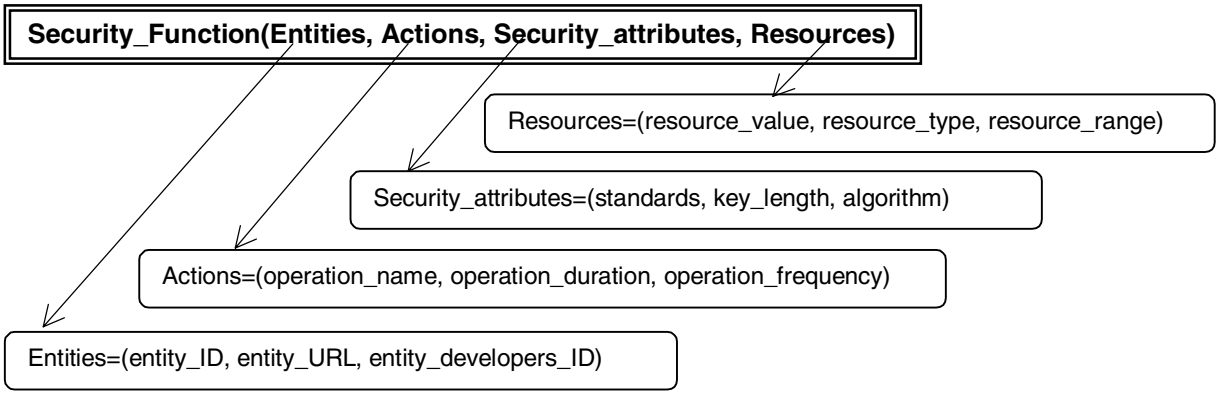

Figure 1. Hierarchical decomposition of the security specification structure

Similarly, the structure of the composition primitive is defined as

composite_security_function({participating_Entities},{permitted_Actions}, {agreed_Security_attributes}, {allocated_Resources}) :: strength of the security_attributes.

Each compositional specification can also specify the degree of the security strength of its composition. The degree of strength is based on the conformity between the *required* and *ensured* security properties, and the structure of the security attribute used for this conformity. The strength of the security attributes entirely depends on the type of the attribute, standard, and algorithm used to generate the security attributes. The issue has not been discussed in this paper due to space limitation.

3. An example for business tax calculation

In this section, we describe a distributed computation system for the business tax calculation as an example that will be used as a case study in the subsequent sections. Individual business application systems used by the business community can be dynamically assembled with a tax calculation software component over the open communication network. The name of the component is *TaxCalc*, a server component. The component calculates tax from a sale, and returns the result to the legitimate users. Most of the users of the component are business people located in various remote places. However, any individual system is allowed to use the component *TaxCalc*. When an user sends a connection request for a compositional relationship to the component from their application system (the client system), the component then identifies itself with its own address, and its interface structures. The component also broadcasts its public key to the user for secure transmission of data to the component. The user system also sends its system-identity in terms of its URL address, or any other identification convention, and non-system identity such as Australian Business Number.

If both parties agree to communicate information, the user system (the client) also sends its public key if there is any, to receive secure data from the TaxCalc. It is assumed that the data exchanged between the TaxCalc and the user system is considered confidential. To make the data confidential, all messages sent to the TaxCalc must be encrypted with the public key of the TaxCalc. In contrast, messages sent to the user systems from TaxCalc may not be encrypted because of the unavailability of

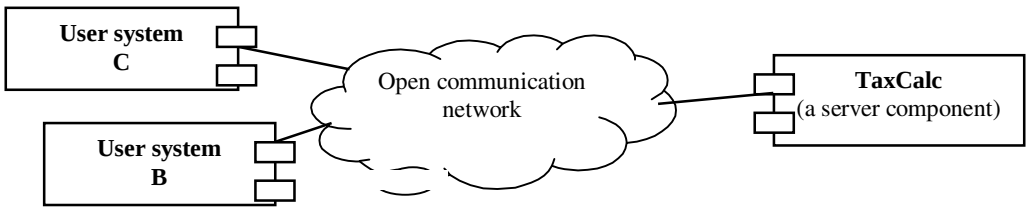

Figure 2. A scenario of distributed computation

any encrypting key of the user system. The user system may not necessarily have any public or private key at all. In such case, the messages would be sent uncrypted from the server component. The user system also supplies the total amount of sale and the expenses both are in terms of dollar. The TaxCalc calculates the tax of the total sale and sends the result to the user system. However, TaxCalc rejects all messages that are not encrypted with its public key. The distributed nature of this scenario is cited in Figure 2.

For the sake of simplicity, we assume a composition is always a binary relationship between two components in any single time. We assume each of the contracting components is spatially separated, and they communicate to each other via a connector or a completely connected point-to-point network. The possibility of multicast communication is not included in this scenario.

4. Security characterisation and compositional primitives

In this section, we are now going to present the characterisation structure and apply the proposed characterisation to the example cited in the previous section to model the security characterisation of components and their composition. Our following characterisation model is applied on two system scenarios of the case study described in the previous study. Most part of this section focuses on the system scenario-1. Later in this section, we describe system scenario-2 with a varied security properties of the user system "B", and apply some compositional primitives.

4.1 System scenario-1

In our above example, we assume that the user system "*C*" possesses a private key which is secret, and a public key which is not secret. *TaxCalc* has also a secret private key and a public key. "C" sends its sales and expenses data to the TaxCalc. The server component "TaxCalc" calculates the tax based on the received data, and transmits it to "C".

4.1.1 Atomic Spec 1: Access authorisation

We can now characterise the high level security interface of a component or CSFI. Before any contract is made between an identified client component and a server component, the authorisation must be granted by the server component. This prior authorisation for such a "*compositional relationship*" can be modelled in a server component as

SF: ***authorise_access(client_ID, requested_actions, security_attribute, resources)*** Spec 1 (a)

SFP: Predicate 1(a) specifies that a client makes a request to perform an action on the server component. Requested_actions can be one or a set of actions to be performed on the server component. Security_attribute can be an authentication data of the client such as a password or a key.

In our example, following information can be characterised in Spec_1(a) as

access_request("C", "get_tax_report", $(password)_C$, NULL)

This example shows that server component TaxCalc authorises the user system "C" to perform the operation "get_tax_report" using its password. "C" is the abstract identity of a user system. This can be

further decomposed according to our structure as "C" = ("CCC Enterprise", "http://102.340.87.56", "NULL"). It specifies a complete identity of the client system "C" including the business name, and its URL. The third argument is a NULL value as it may not be available. The request generated by the client component, on the other hand, can be characterised as

SF: ***access_request(server_ID, requested_actions, security _attribute, resources)*** Spec. 1(b)

SFP: The predicate 1(b) access_request specifies a client requesting the right to perform the action on a server, and ensures the security attribute password or key as required by the server component.

In the context of our example we can find Spec_1(b) as

access_request("TaxCalc", "get_tax_report", $(password)_C$, NULL)

We can further zoom on the entity "TaxCalc" to get more identity information such as

"TaxCalc" = ("Tax Calculation Ltd.", "http://900.534.23.34.8", "Easy Tax Solutions")

This characterisation describes that the identity of the component *TaxCalc* is Tax Calculation Ltd., the URL of the component is http://900.534.23.34.8, and the original developer of this component is Easy Tax Solutions.

4.1.2 Composite_Spec 1

This compositional specification is based on Spec_1 (a) and (b), and could be more formally defined as

composite_access_authorisation({client_ID,server_ID}, permitted_actions, agreed_security_attribute, allocated_resources) **::** strength of the security attribute /* i.e. number of characters used in the password, whether alphanumeric or numbers, key length and algorithm used, and so on.*/.

This characterises a composition between a client and a server components on the permitted actions based on the agreed security attributes. Assume a server component has a *required* property such as Spec_1 (a). A client component would send some information to the server and *ensures* the security attribute *required* by the server component. The client is authorised to perform specified actions on the server component. Note that this compositional specification does not characterise how the security of the exchanged information or data would be maintained. It only specifies the agreement between two components based on the password or key as the security attributes supplied by the client and the nature of the action. By applying this compositional specification in our example, we can capture the following security compositional primitive.

composite_access_authorisation({"C", "TaxCalc"}, "get_tax_report", $(password)_C$, NULL)

This scenario is shown in Figure 3. It describes that the server component *TaxCalc* and the client system *"C"* have agreed to establish a compositional relationship on the basis of the action *get_tax_report* using the password of the client "C".

4.1.3 Atomic Spec 2: Data authentication

A server or a client component may have a security function to check the integrity of an input data, and this can be characterised as

SF: **validate_in_data(sender_ID, actions, security_attribute, resources)** Spec_2

SFP: validate_in_data predicate characterises an evidence or guarantee of authenticity of the data contents in_data in terms of associated security_attribute. security_attribute can be a key used to decrypt the received incoming data. Examples of such attributes are private keys, secret keys, shared keys, and so on.

In our example, we can charaterise the following security information using this specification.

validate_in_data("C", decrypt, $(key)_{TaxCalc^-}$, {sales_amount, expenses_amount})

In this example, the ($(key)_{TaxCalc^-}$) is the private key of the server component "TaxCalc" that receives the encrypted data from the component "C". Only the recipient component "TaxCalc" can decrypt the message using its private key to get the actual sales and expenses amount of the client.

4.1.4 Atomic Spec 3: Export of data to outside control of components

A server or client component may have a security function to encrypt an outgoing data with proper security attribute, and that can be characterised as

SF: **protect_out_data(receiver_ID, actions, security_attribute, resources)** Spec_3

SFP: The protect_out_data predicate characterises the type of security_attribute used to encrypt the outgoing data produced by the sender component. Examples of security_attribute can be public key, shared key and so on.

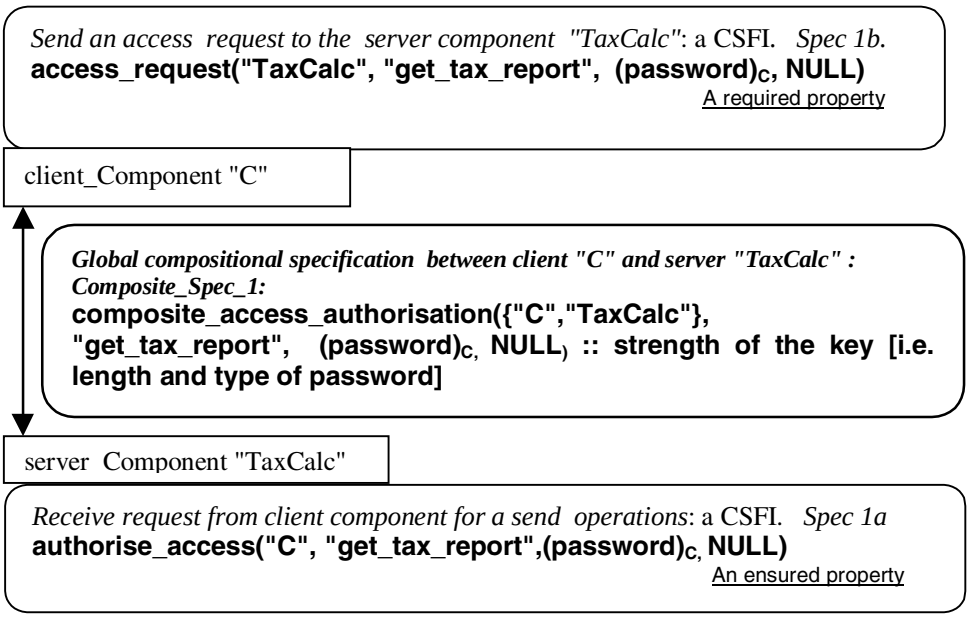

Figure 3. An example of composite_Spec_1

In our example, this can capture the following security property.

protect_out_data("TaxCalc", encrypt, $(key)_{TaxCalc}+$, {sales_amount, expenses})

The $((key)_{TaxCalc}+)$ is the public key of the component "TaxCalc" to whom this data is to be sent. It shows that the "sales amount and expenses amount" sent by the client is encrypted with the public key of the recipient component "TaxCalc". However, the out_data may not be always encrypted, in such case, the value of the security_attribute will remain NULL.

4.1.5 Composite_Spec_2:

This compositional specification is based on two specifications, Spec_2 and Spec_3. This composition can be formally modelled as

composite_out_in_data(participating_entities,permitted_actions, agreed_security_attributes, allocated_resources) **::** strength of the security attribute in terms of bytes and algorithms used.

From the example, the following compositional specication can be modelled using the composite_Spec_2 as

composite_in_out_data({"C", "Tax_Calc"}, {encrypt, decrypt}, , {$(key)_{TaxCalc}+$, $(key)_{TaxCalc}-$}, {sales_amount, expenses_amount}) :: strength of the TaxCalc's private key in terms of number of bytes and algorithms used.

This specification has four arguments, each having two elements. The elements in the fourth argument is applicable to all other arguments. The first elements of each of the three arguments are related in a specification. For example, we can read the above specification as: "C" encrypts

Proceedings of the 36th International Conference on Technology of Object-Oriented Languages and Systems
0-7695-0875-8/00 $10.00 © 2000 IEEE

sales_amount and expenses_amount with the security attribute $(key)_{TaxCalc+}$. The second specification embedded in the above composition can be read as: "TaxCalc" decrypts the sales_amount and expenses_amount data received from "C" with the security attribute $(key)_{TaxCalc-}$. This specification tells us that the client component "C" sends sales and expenses data protected by the key of the server TaxCalc component. And the server component TaxCalc receives input data protected by its own key from the client component. It can decrypt the message using its private key. Figure 4 shows this compositional specification.

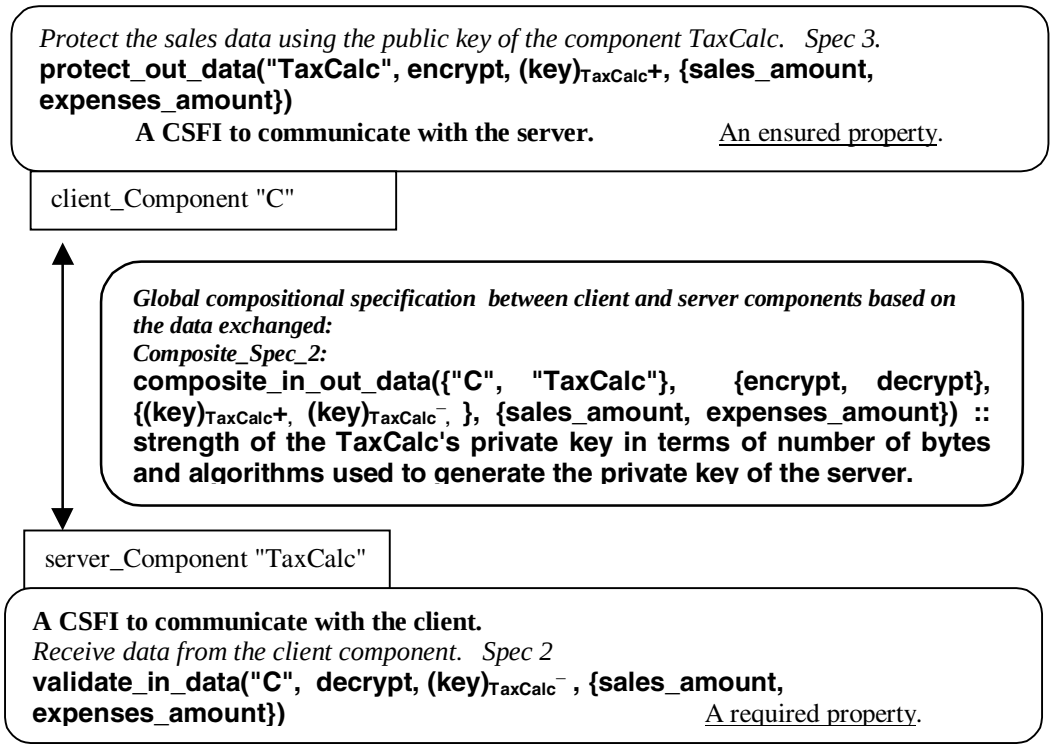

Figure 4. An example of composite_Spec_2

The server component can also transmits the calculated tax to its client, and this can also be modelled as

composite_in_out_data({"TaxCalc", "C"}, {encrypt, decrypt}, $\{(key)_{C+}, (key)_{C^-}\}$, tax_data) :: strength of the private key of "C" in terms of number of bytes and algorithms used.

This instance shows that TaxCalc encrypts the tax_data with the public key of "C", and "C" can decrypt the message using its private key.

4.2 System scenario -2

In this scenario, we assume another user system called "B" establishes a composition with the component "TaxCalc". "B" does not have any private or public keys. After the establishment of a compositional relationship with "TaxCalc", the user system "B" sends its sales and expenses data to the TaxCalc" without encrypting the data. This specific scenario can be modelled using our compositional primitive such as

composite_in_out_data({"B", "TaxCalc"}, {NULL, decrypt}, {NULL, $\{(key)_{TaxCalc-}\}$, {sales_amount, expenses_amount}) (Composite_Spec_2)

The data sent by "B" has not been encrypted with the public key of "TaxCalc". In this case, the server component "TaxCalc" does not accept the data received from "B", because "B" has not ensured the required security property defined by "TaxCalc", although "B" has already been permitted to send the

data to "TaxCalc". Now assume that "B" again sends the same data to "TaxCalc" encrypting the data using the public key of "TaxCalc". This scenario can be captured as

composite_in_out_data({"B", "TaxCalc"},{encrypt, decrypt}, {(key)$_{TaxCalc}$+, (key)$_{TaxCalc}$-}, {sales_amount, expenses_amount}) (Composite_Spec_2)

"TaxCalc" receives and accepts the data this time and decrypts it with its private key. The component calculates the tax, and sends the data to the user system "B". Since "B" does not have any public key, "TaxCalc" sends data to "B" without encrypting the data. This can be characterised as

composite_in_out_data({"TaxCalc", "B"}, {encrypt, decrypt}, {NULL, NULL}, tax_amount) (Composite_Spec_2)

Note that the user system "B" accepts the uncrypted data from TaxCalc because "B" does not have any required property. In such situation the security property of the data is nil. The tax_amount received by "B" can not be considered secure.

Assume if "B" sends this tax_amount data further to a third component, say, to "C" with encrypting it with the public key of "C", the security property of this data is still nil. This can be seen as ternary compositional relationship among "TaxCalc", "B", and "C". A compositional relationship with more than two components can be complex to be characterised. We are currently exploring the ternary relationship and their security characterisation.

All these examples demonstrate that each component may have a distinct access control structures and each may be administered and maintained independently. In this work we have not included other security aspects of compositions such as non-deducibility, non-leakage of information, and non-interference. We have only considered the access control and validity of data in our work in this paper.

5. Application

Our security characterisation and their compositional primitives can be the part of component introspection in such a way that a component may know its required and assured security properties, and can communicate this knowledge to other interested components. With support for introspection, a component can be queried about its security properties (assured, and required). The availability of such meta-information along with the component interface signature may facilitate the prior understanding of the security nature of components and their possible compositional effect in dynamic run-time discovery and execution. The security characterisation can be stored in a meta-object which can be integrated with the object references and control access to the corresponding component. If a well structured security characterisation scheme is attached with the object reference, then the interested client may be able to decide whether it will use the object or not.

IDL of CORBA provides the specification of interface signature for structural matching between components. A complete security characterisation structure could be attached with the IDL of CORBA to allow client components to verify the conformity of the security properties provided by the server components. This information can be integrated into the interface repository to be discovered dynamically by a client component. The IDL can be extended with the security characterisation structure so that security properties of a component can be specified in its interface signature and stored in the interface repository. Such an extended CORBA's dynamic interface support can be used by client components to retrieve the security properties of the candidate component. A client component may run a 'satisfaction test' between the ensured security properties and the required properties before an actual composition takes place. Similarly, the security characterisation structure can be codified with the JavaBean retrospection mechanism (BeanInfo) and Java's reflection capability for Javabeans.

6. Further work and conclusion

We have proposed a specification structure to capture and characterise the security properties of independent software components and their compositional specification. The structure is based on four fundamental compositional elements. Each of the elements can be further decomposed to specify more

security related information of a component. This model can expose the nature of security considerations of a component to other components. We have shown with a simple example how a component can have a specified ensured security properties corresponding to required security properties of another components. We have also demonstrated how the same structure can also be used to characaterise a binary compositional realtionship. Our ultimate objective is to compose multiple specifications that include several components. We will explore other security classes proposed in CC, and their possible compositional specification in our future work. We are now working to formalise our approach for a complete model of security characterisation.

The major limitation of the work presented in this paper is that we have not defined how a more complex compositional security specification would be built on these security sepcifications. We are currently working in that direction to formalise some complex compositional rules. Based on a binary relationship between two specifications we may be able to formulate a more complex multiple relationships among a group of components. We realise that the security strength of two interacting components is more complex than the specification of individual component in isolation. Our current work is intended to combine the security specification of multiple components in order to model a complete compositional security specification in a given time.

References

[1] Khan, K., Han, J., Zheng, Y.,"Characterising User Data Protection of Software Components", Proceedings Australian Software Engineering Conference 2000, IEEE Computer Society press, Canberra, April 28-29, 2000, pp 3-11.

[2] ISO/IEC-15408. (1999). Common Criteria project. Common Criteria for Information Technology Security Evaluation, Version 2.0.NIST, USA. http://csrc.nist.gov/cc/, June 1999.

[3] Han. J., "A Comprehensive Interface Definition Framework for Software Components", Proceedings of 1998 Asia Pacific Software Engineering Conference, IEEE Computer Society press, Taipei, December 1998, pp. 110-117.

[4] Arafeh, B., "A Graph Grammar Model for Concurrent and Distributed Software Specification-in-Large", Journal of Systems Software, Elsevier Science Inc. 1995, 31:7-32, pp. 7- 22.

[5] Pandey, R., Hashii, B., "Providing Fine-Grained Access Control for Mobile Programs Through Binary Editing", Proceedings 13th ECOOP, Lisbon, Lecture Notes in Computer Sciejnce-Verlag, Springer, 1999.

[6] Selic, B., "A Generic Framework for Modelling Resources with UML", IEEE Computer, 33-6. June 2000, pp. 64-69.

[7] D'Souza, D., Wills, A. :Objects, Components, and Frameworks with UML - The Catalysis Approach, Addision-Wesley, 1998.

[8] Meyer, B., "Applying "Design by Contract", IEEE Computer, vol. 25, no. 1992, pp. 40-51.

[9] Meyer, B., Mingins, C., "Providing Trusted Components to the Industry", IEEE Computer, June 1998, pp. 104-105.

[10] Thomson, C.: Workshop Reports. 1998 Workshop on Compositional Software Architectures, Monterey, California http://www.objs.com/workshops/ws9801/report.html

[11] Lindquist, U., Jonsson, E., "A Map of Security Risks Associated with using COTS", IEEE Computer, June 1998, pp. 60-66.

[12] Beugnard, A., et al., "Making Components Contract Aware", IEEE Computer, July 1999, pp. 38-46.

[13] Michener, J., Acar, T., "Managing System and Active-Content Integrity", IEEE Computer, 33-7, July 2000, pp. 108-110.