Characterising User Data Protection of Software Components

Khaled Md. Khan[1] Jun Han Yuliang Zheng
Peninsula School of Computing and Information Technology
Monash University
McMohans Rd, Frankston
VIC 3199, Australia
{khaled, jhan, yuliang}@mars.pscit.monash.edu.au

Abstract

This paper makes an attempt to propose a scheme to characterise non-functional security properties that are embedded with the functionality of software components. The security properties are attached with various aspects of a component such as resource allocation, user data protection, communication, and so on. In this paper we are particularly interested in characterising the user data protection of software components. It is often reported that software components usually suffer from security and reliability problems. It is now widely recognised that characterisation of security properties of software component is an important issue to boost the confidence and trust on component technology. To address this issue, the characterisation of security properties of component is the first challenging step. The work proposed in this paper is partially based on the functional requirements defined in Common Criteria for Information Technology Security Evaluation endorsed by NIST. The applicability of the proposed scheme is demonstrated with a simple example.

1 Motivation

With the rapid growth of the information technology, component-based software development has become increasingly attractive. Although the notion of component is not entirely new, its applicability is recently getting new momentum both in research and practice. In component-based software engineering (CBSE), an information system is considered as a set of interacting separable stand-alone software components. A component may be assembled with an application either statically or dynamically over the Internet. In a dynamic assembly scenario, the specification of the functionality and the quality attributes of a target component located in a remote server may not be available to the user. In addition to that, software composers do not have any control over the behaviour of the components due to the unavailability of the source code and the design artefact. The underlying non-functional quality attributes of the components are either unknown or not trusted as claimed. In such a scenario, components may not be completely trusted by the user. We believe that security risks involved with using 'foreign' software components are quite different from those associated with customised application. A key to the success of a viable component market is its ability to build confidence and trust on the components developed by third parties [7], [8]. To create a certain level of trust, the quality attributes of software components must be well specified with their interfaces.

To fully materialise a trusted component model, software components must be *certified* in terms of their security properties [3]. The certification is a type of quality approval that can specify and provide guarantee of the component quality with a given degree of confidence. The certificate itself is not a security property, rather it is a trust model to boost user confidence on the product. The certification process includes the characterisation and assessment of security properties of components. In facilitating the *certification* of components, we need to be able to characterise the functionality and quality properties of components [2], [15]. The issue of characterisation of component functionality has already been well studied in [5], but the aspect of characterisation of non-functional attributes such as security, scalability, reliability of component has not been properly addressed yet. In this regard, security is considered as one of the most important quality attributes of software components. The security features that a component provides are the

[1] The author is currently with the School of Computing and IT, University of Western Sydney, Nepean. His email address is : k.khan@uws.edu.au The work was also partially funded by the DSTC Australia.

0-7695-0631-3/00 $10.00 © 2000 IEEE

most important contributing factors to build the trust. Components assembled perfectly in a software architecture may not provide an acceptable level of security. Therefore, it is vital that we must be able to scale up security properties of components and their possible impact on the enclosing system. To address this pressing issue, a formal scheme to characterise the security properties of existing components is urgently needed as expressed in [4], [6], [8], [9], [10], [11]. A quick browse of the current published works suggests that research issues such as protecting the enclosing system from the unsafe component, finding and removing security flaws of components dominate the field. However, we believe that if the trust related security properties are well specified with the component interface signature, and available to the user, only then components can be used with more confidence by the application developers.

The paper is organised as follows. The scope and objectives of this paper are described in section 2. We have proposed a preliminary characterisation scheme based on the functional requirements of Common Criteria [1] in section 3. Section 4 describes an example that is used in our scheme. The result of the application of our scheme on the cited example is also focused in this section. Section 5 outlines the implementation possibility of our scheme in component technology. Finally, we have summarised and outlined our future research direction in section 6.

2. Scope of the paper

Software components have, in fact, two aspects related to their security: *specification*, and *characterisation*. The *specification* of security properties is needed during the design of components, but the *characterisation* is required during assessing and certifying the existing components in terms of their security properties. The main focus of this paper is on defining a scheme that can be used to characterise the security properties of software components. The scheme proposed in this paper, however, could be used as a guidance to specify the security requirements of new components as well. It can act as a guiding vehicle for the component designer to specify the security requirements of software components. However, the application of our proposed scheme is restricted in this paper only to the characterisation of existing software components.

This paper identifies the security enforcement mechanisms that are already embedded in various forms with the functionality of components at the implementation level. These properties are inherited in the implementation of the component functionality. We believe these properties are the abstract implementation of the security policies that are intended to counter certain security threats. These objectives must be translated into formally specified security properties that can be subjected to be assessed for the trustworthiness of the candidate component. The security properties may be attached with various properties of a component such as resource allocation, user data protection, communication, and so on. Our characterisation scheme for user data protection is partially based on the *functional requirements* defined in the *Common Criteria (CC) for Information Technology Security Evaluation, version 2.0* [1]. Common Criteria provide a schema for evaluating IT system, and enumerate the specific security requirements for such systems. The *functional requirements* in CC describe the security behaviour or functions expected of an IT system to counter threats. These requirements consist of eleven *classes*. Each of these classes comprises its members called *families* based on a set of security requirements. Each of the family in a class can be assessed in terms of their strengths and weaknesses. In our approach we have used particularly one such class called *Class FDP: User data protection* as an experiment for our study. The families of this particular class specify various requirements for security functions and security policy related to the *user data protection* of a target IT system. It is worth to note that these requirements are actually defined for IT products in general, not for software components. Therefore, the main purpose of this study is to investigate the possibility to apply some of these requirements to software components in characterising the security properties of components. We assume that all of the requirements in CC may not be applicable directly to software components due to the distributed nature of software components and their compositional complexity. Therefore, we have extensively modified the structure and properties of the class FDP to accommodate the basic nature of software components in our scheme.

3. Structure of the scheme

The structure of our scheme comprises a *security class*, a collection of *security objectives* related to the *security class*, a collection of *security functions*, and *entity* and *action* used by the security functions. Two types of entities are used in a security function: *subject* and *object*. The term *security class* is used to group a collection of *security objectives* that shares a common focuses while differing in coverage of *security functions*. In our case in this paper, a *security class* may deal with the security properties of protecting user data; hence the class is called *User Data Protection*. The *security class* called User Data Protection consists of seven *security objectives*. *A security objective* is represented by a collection of *security functions* that are used to achieve a predefined security goal. For example, using a security function like verifying

passwords can be used for the authentication of a user. A single or a collection of *security functions* may be used to counter one or more threats defined in the *security objective*. A *security function* takes *entities* and *actions* as a predicate form to express a complete security function. The structure of our scheme is shown in Figure 1.

A *subject* is an entity within the application or component which causes an *action* to be performed on an *object*. The subject can be in any form such as users, a component, a procedure, or a method. The subject can be defined as

SUBJECT{user, procedure, program,..,component}.

An *object* is an entity that is used by a *subject*. It can contain and receive information upon which a *subject* performs *action*. The object can be in any form such as a variable, a piece of data, a procedure, a method, or a data file. Some objects may also act as subjects depending on their roles in a specific context of application. For example, a function or a procedure is a *subject* when it calls or uses other functions. However, a function can be an *object* when it is called or used by other functions. The *object* can be defined as

OBJECT{component, program, data,...,file}

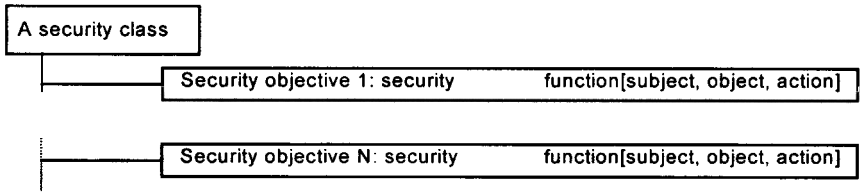

Figure 1. The structure of the characterisation scheme

An *action* is applied on an *object* by a *subject*. A subject can perform an action such as read, write, delete, execute, use, connect and so on. Different types of actions can be defined as

ACTION {read, write, use,....,execute}

We now define some categories of security functions that are used as security properties such as *authorisation, authentication, validation* and *protection*. To keep our study in a manageable size this work is restricted to only these four types of security functions. Table 1 shows the format of these four functions with corresponding examples.

Each function may incorporate a rating based on the degree of strengths and weaknesses of the function in a particular context of application. The accumulated ratings of all functions of a particular security objective would express the ultimate strength of that security property. Similarly, the total ratings gathered from all security objectives would make a final rating to a particular *security class*. However, We have not defined this rating structure in our scheme at this stage.

Now we are in a position to formulate some security objectives and their functions related to the security class *User Data Protection.*

ACCESS_CONTROL : (1)
AUTHENTICATION [subject],
AUTHENTICATION [object],
AUTHORISATION [subject, object, action]

An *access control* is required when two components intend to make a contract for exchanging services. Three security functions associated with this security objective describe the purpose of this objective. This suggests that the *subject* and *object* must be authenticated before they are used in a contract. Note that the *object* also might be required to be authenticated before it is used in some context. This is necessary particularly when a component

Security functions	*Security Issue covered*	*Examples*
AUTHORISATION [*subject*, object, *action*]	Can the *subject* perform the *action* on the *object*?	The *userB* can *execute componentX*
AUTHENTICATION [*subject*].	Is this *subject* is the right person, or has proper identification?	*Is userA* the right user?
VALIDATION[*object*]	Is this *object* in correct type and range?	*EmployeeID* is in correct *type* and *range*
PROTECTION[object]	Is this *object* guarded?	*salary.dat* file can only be modified by the *userY*

Table 1. Four types of security functions

is assembled dynamically over the Internet, for example. The subject may pose a question such as "Is this the right component that is going to be assembled with the application system? " We will see this aspect in our example shortly. However, in addition to that, an access control may have additional functions that may formulate the expected behaviour during the interaction between two contracting components.

IMPORTED_DATA_CONTROL: (2)
VALIDATION [object]

This security objective is used to validate data imported from the application system that is outside of the control of the component. The function protects the internal data of the component from being corrupted by the imported data. It checks the type and range of "input" data particularly. An incorrect data type may have serious implication on the system. The correct matching of data format in both ends of the contract is an important aspect in component technology. It is reported that a common error occurred in system execution is the out of range value during an unprotected type conversion [12], [13]. Incompatibilities of the input and output values may lead to an undesirable state of the entire system. Therefore, components should be carefully matched to the context of their use in terms of their compliance with the enclosing system [12].

EXPORT_DATA_CONTROL: (3)
VALIDATION [object]

This security objective is concerned with the security of the data to be exported to the application which is outside of the control of the component. It involves in ensuring the correct form of data and their range, accurate and unambiguous information, and so on. Typically, the security function under this objective deals with the output data to be sent from the component after processing.

INTERNAL_DATA_TRANSFER_CONTROL: (4)
AUTHENTICATION [object],
VALIDATION [object],
PROTECTION [object]

Table 2. Security objectives with associated functions of the security class User Data Protection

1	ACCESS_CONTROL: AUTHENTICATION [subject], AUTHENTICATION [object] AUTHORISATION [subject, object, *action*]
2	IMPORTED_DATA_CONTROL: VALIDATION [object]
3	EXPORT_DATA_CONTROL: VALIDATION [object]
4	INTERNAL_DATA_TRANSFER_CONTROL: AUTHENTICATION [object], VALIDATION [object] PROTECTION [object]
5	STORED_DATA_PROTECTION: PROTECTION [object]
6	RESIDUAL_DATA_PROTECTION: PROTECTION [object]
7	UNDO_PROCESSSING: PROTECTION [object]

This security objective addresses the issues of protection, validation, and authentication of data when they are transferred between two internal entities which are logically separated from each other within a component.

This involves, for example, use of global variables in a procedure may have security impact on the data. Use of global variables in a procedure may not hide the memory address of the data properly. This security objective deals with the prevention of disclosing data during transferring to other parts of the component. The objective can be

accomplished by using one or more of these three functions.

STORED_DATA_PROTECTION: (5)
PROTECTION [object]

This objective is defined to protect the user data while it is stored in a file or in a database. It also verifies the integrity of the data that is stored. Database systems usually have some built-in features such as integrity constraints, referential constraints and so on. These can enforce certain level of security objectives in the databases.

RESIDUAL_DATA_PROTECTION: (6)
PROTECTION [object]

This objective addresses the issue of protecting data that has been logically deleted or released by a processing unit of the component. It ensures that all of the deleted data is unavailable to all other entities. For example, the exit condition of an function implies that it must initialise all its local variables containing input and temporary data which are not required by any other processing unit of the component. Users of a system may leave a trail of *data shadow* with the contents about what the last data they manipulated [14].

UNDO_PROCESSING: (7)
PROTECTION [object]

There are situations which are beyond the control of a component such as abnormally disconnected session by the user either accidentally or intentionally. In such a situation, the undelivered output data including all temporary data must be discarded immediately, or the system can undo the previous action to preserve the integrity of user data before the component is connected to other users. This security objective is intended to protect the user data from being misused by other units due to an interruption of the session. For example, if a function exits abnormally, it must undo all its actions by discarding all output produced by the function. The scheme of the security class: user data protection is summarised in Table 2 for a ready reference.

4. A Case Study

The applicability of our scheme is demonstrated by using it to a small example. First we describe the system scenario in the example. We will then apply our scheme to this example, and try to deduce the security properties of the system.

4.1 The system description

"A medical system used by the medical practitioners is dynamically assembled with a software component time to time as needed. The name of the component is **diaR**. This component caters the diagnosis reports on patients to the legitimate users. Most of the users of the system are medical practitioners. Not all users are authorised to retrieve the report of patients. Only selected users have access to the reports of his/her own patients although they may have access to the other services provided by the component. When an user sends a connection request to the component from the application system, the component then identifies itself with its own address, security label, and its interface structures. After connecting with this component the user supplies the password and the User ID number in order to access to the component services. The component then consults its internal configuration file to determine if the request from the user is permitted. The component validates the request and generates an accept or reject message depending on its processing. This security function is directly concerned with how users get access to the component. This particular component provides several other functions to various types of users according to their role. However, we are interested in one particular service, that is, *generating diagnosis report*".

The service of the component in which we are interested in is a set of operations supported by an interface structure. The interface structure of the component is outlined in Figure 2. In this system, the user's application system communicates with the component through unidirectional channels in terms of interfaces and their attributes. In this case the attributes and the name of the interface together have formed the entire interface structure without considering their implementation. The information that passes through these channels is a collection of input, output and state variables in varying data formats ranging from atomic to complex data types. Three distinct types of data that enter and leave the component are namely: *in*, *out*, and *state*.

The interface and the attributes are visible to the user, but the operations used to support the service are not. All these six operations are required to provide the service getDiagnosisReport() of the component. We assume that the component and the user both have their own unique identification such as diaR and 99999 respectively. The patient number is 87878 in a particular session.

4.2 Results

A close examination of the cited example in the previous section reveals various functional and non-functional security properties that can well match with our scheme. More on the issue of functional and non-functional security properties can be found in [16]. The complete scenario of the scheme and the results of its application to our example are summarised in Figure 3. At the lower half of the Figure the dashed circles denote security properties, whereas the solid line circles denote component functionality. The corresponding level of each of these security functions is referred as L1 (level 1), L2 (level 2), L3 (level 3), and L4 (level 4). These suggest that the security of the component propagates from the high level to low level security properties. The various layers of security properties shown in the Figure are application specific. It demonstrates that security can be embedded with the functionality of the component in various layers of abstractions. We discuss all these four levels in brief.

L1: This is the most outer level of security in the layer. Whenever the component receives a connecting request from the user it has to identify itself by supplying its address and the label. This can be translated into our scheme in security objective (1).

ACCESS_CONTROL: (1)
AUTHENTICATION [diaR]

If the user system is satisfied with the authentication information of the component diaR as used as an *object* in this case, it is then connected with the component. The user is then asked to supply the password and the UserID number. This security function can be translated into our scheme as

ACCESS_CONTROL; (1)
AUTHENTICATION [99999]

```
// interface structure visible to the user
getDiagnosisReport(); //name of the interface that is visible to the user.

// attributes of the interface visible to the user.
long UserID; // the unique ID number of the user as "in" data
long patientID; // (data range:111111-99999) required "in" data;
string diagnosisReport; //provided "out" data;
int state; // (data range: 0,1) a Boolean data type returns the execution status;

// the following operations are attached with the interface but NOT visible to the user
checkValidDataRange(); //check the type and rang of the data - a security constraint.
existPatient(); //check whether the object Patient exists in the data file
checkAuthorisation(); //check whether the subject has proper authorisation to perform the next operation - a
fundamental security constraint.
readReport(); // the main function to retrieve the data.
dispatchReport(); //sent the report to the user's system.
InitializAllLocalVariables(); // just after the dispatch of the report all local variables must be initialized to binary zero or
NULL.
```

Figure 2. Attributes, interface signatures and operations of the component

If it is determined by the component that this particular *subject*, 99999 in this case, is authorised to use this component then he/she gets access to the component. This access control can be translated into our scheme as

ACCESS_CONTROL; (1)
AUTHORISATION [99999, diaR, execute],

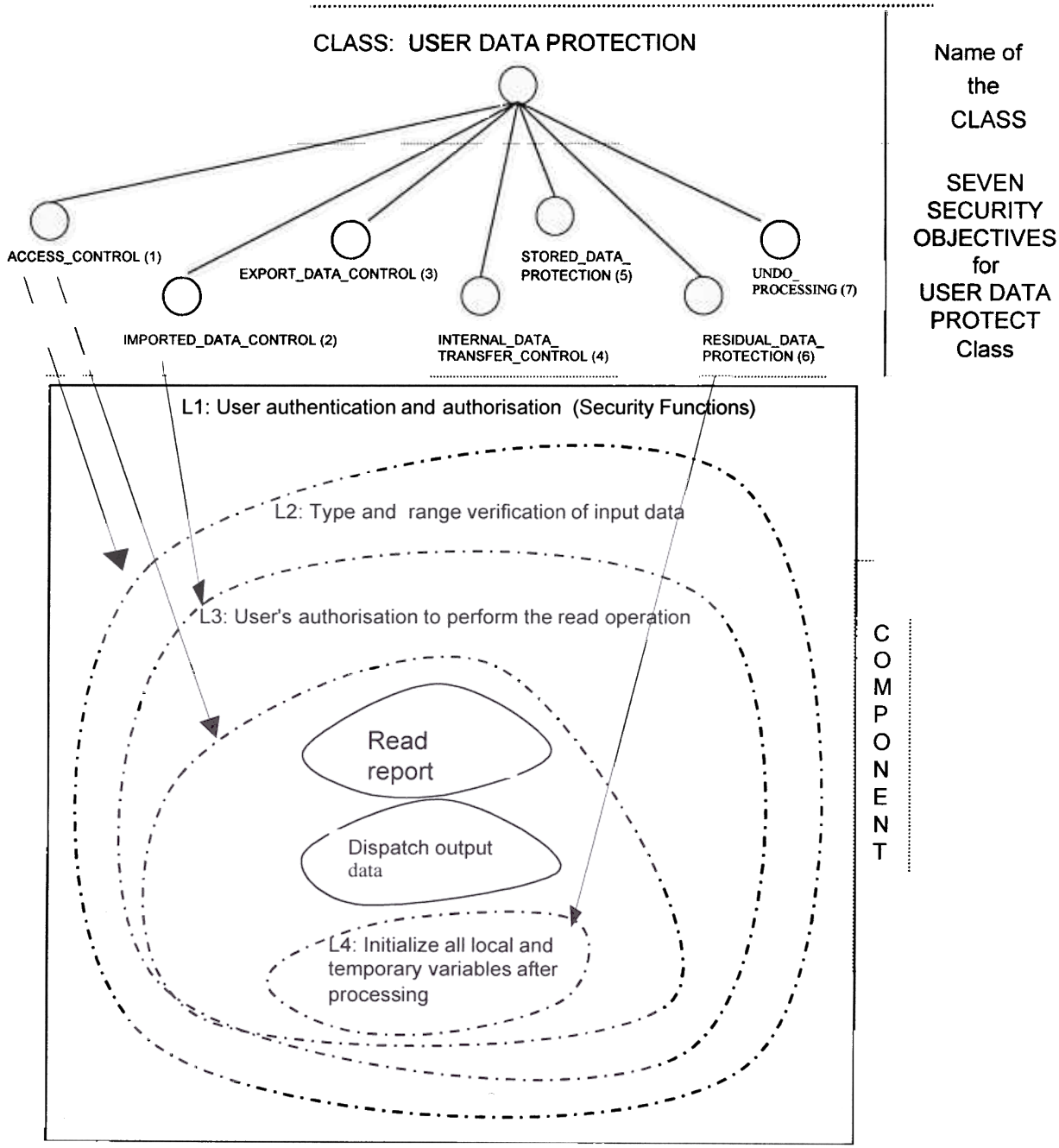

Figure 3. The Application of the Characterisation Scheme and the Result

where 99999 is the subject, diaR is an object, and execute is an action. These functions are considered as pure security functions, that is, they are not embedded with the component functionality. From these three security functions we can derive a complete security objective regarding the access control of two contracting software components.

i. The user system must be protected from being assembled with any unauthorised component
ii. The component must have a security function to protect itself from an unauthorised entity.

L2: The non-functional security property in the layer can be derived from the operation checkValidDataRange(). This corresponds to the security objective (2) in our scheme as

IMPORTED_DATA_CONTROL: (2)
VALIDATION[99999],
VALIDATION[87878]

In this security objective, two security functions are used, where both 99999 and 87878 are considered as objects in this case. Some may argue that this function is more inclined to the fields of software assurance and reliability. We do agree with this opinion. However, we also believe that the data type and range checking function can also be extended to as a security function as we see in this example.

The security objectives derived from these two functions are

i. The value of the userID must be in correct range and type as an input before it is further processed by the component
ii. The value of the patientID must be in correct range and type as an input before it is further processed by the component.

L3: This is an important non-functional security property in this application. The property is derived from the operation checkAuthorisation() and are translated as

ACCESS_CONTROL: (1)
AUTHORISATION [99999, readReport(), execute],
AUTHORISATION [99999, 87878, read]

This objective involves two functions. The former one uses 99999 as a subject, readReport() as an object, and execute as an action. The second function also follows the same format. The security objectives of these functions can be interpreted as

i. The access to the component diaR must not automatically allow all users to execute readReport() operation of the component.
ii. The access to the readReport() operation must not allow a user to read report of any patient they like. Users must be restricted to read only the authorised report.

L4: This is the last level of security mechanism in the layer. The derived non-functional security properties from the operation initializAllLocalVariables() can be transformed as

RESIDUAL_DATA_PROTECTION: (6)
PROTECTION [99999],
PROTECTION [87878],
PROTECTION [report...]

The security objective has three functions, each has a single object. UserID 99999 has been used as a subject in other functions so far. But in this case, the first function uses 99999 as an object. These three functions are completely internal to the component. The operation initialises all its local variables to NULL. The derived security objectives from these three functions can be described as

i. The value of the patientID, report and the userID must be removed after processing of final output and just before the exit of the operation.
ii. All local and temporary variables must be set to binary zero or NULL.

However, it is quite possible to uncover more security properties in this example if we analyse the implementation details of the operations like call sequences, parameters passing between operations, file permissions, and so on. A thorough analysis may uncover more security objectives such as

STORED_DATA_PROTECTION,
UNDO_PROCESSING,
EXPORT_DATA_CONTROL

as defined in our scheme. A functionality may contain multiple security objectives supported by several security functions.

The example also reveals that some operations are implemented as underlying logical framework into the component functionality, and act as security side effects. These operations might not be intentionally programmed for security purposes during the design of the component.

5. Application of the scheme

The component developer may apply our scheme to specify the security requirements of the components. The specification may be attached with the component interface for runtime access by contracting client components. For example, OMG's CORBA may integrate the security properties into its interface repository to be discovered at runtime by a client component. Similarly, Sun's Java built-in reflection mechanism may dynamically discover the security properties along with the component structure.

The security of software components depends on the usage context of the components, and the role they play in a specific application environment. Software components must be customised with their application environment where they are deployed. A security property of a component may appear strong in one context, but the same security feature may not be adequate in another context because the component may play a completely different role in another application environment. Quite often, third party software components need additional

external protection from the users to meet their application requirements. If the underlying security properties are exposed along with the component interface signatures, users could easily employ additional external protection to the component or to their application system. Adding security properties to the interface signature would certainly help users to evaluate the security strength of the candidate component. The certifying authority may also use the scheme to assess and certify components in terms of the security and reliability properties of the product.

6. Summary and further work

In this paper, we have demonstrated using our scheme that non-functional security properties such as user data protection are embedded in various ways in different layers of the security mechanism, each representing a specific level of abstraction to achieve certain security objectives. Such a scenario drawn using our scheme may give us an understanding of the security properties of components and their potential system-wide impacts during software composition. We can further expand our scheme such a way that it can capture the relative strengths or weaknesses of the security functions depending on the implementation mechanism of the security objectives. Each of these security properties may be given a relative rating regarding the strengths and weaknesses of the implemented features. We are currently exploring other classes and their families in CC for their applicability to software components. Our complete characterisation scheme is to be formalised to capture the whole range of security properties of software components. The ultimate goal of our research is to establish a formal model that could be used to identify and quantify the detailed security properties embedded with the services that a component provides. This model would be a useful tool for the assessment and certification process of software components developed by third parties.

References

[1] 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.

[2] Han, J., Zheng, Y. (1998). *Security Characterisation and Integrity Assurance for Software Components and Component-Based Systems*, Proceedings 1998 Australian Workshop on Software Architecture, Monash University, Melbourne, pp 83-89.

[3] Han, J. (1999). *An Approach to Software Component Specification*, Proceedings of International Workshop on Component-Based Software Engineering, San Francisco, 1999.

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

[5] Han, J. (1998) *A comprehensive interface definition scheme for software Components*, Proceedings of 1998 Asia Pacific Software engineering Conference, IEEE Computer Society press, Taipei, December, pp. 110-117.

[6] Voas, J. (1998). *The Challenges of Using COTS Software in Component-Based Development*, IEEE Computer, June 1998, pp. 44-45.

[7] Szyperski, C. (1998). *Component Software -Beyond Object-Oriented Programming*. Addison-Wesley, 1998.

[8] Meyer, B., Mingins, C. (1998). *Providing Trusted Components to the Industry*, IEEE Computer, May 1998, pp. 104-105.

[9] Lindquist, U., Jonsson, E. (1998). *A Map of Security Risks Associated with Using COTS*, IEEE Computer, June 1998, pp. 60-66.

[10] Voas, J. (1998). *Certifying Off-the-Shelf Software Components*, IEEE Computer, June 1998. pp. 53- 59.

[11] Talbert, N. (1998). *The Cost of COTS*, IEEE Computer, June 1998, pp. 46- 52.

[12] Jajodia, S., Ammann, P., McCollum, C. (1999). *Surviving Information Warfare Attacks*, IEEE Computer April 1999, pp. 57-63.

[13] Lowry, M., (1999). *Component-Based Reconfigurable Systems*. IEEE Computer Society, April 1999, pp. 44-45.

[14] Yen, I-Ling, Paul, R., Mori, K. (1998). *Toward Integrated Methods for High Assurance Systems*, IEEE Computer, April 1998, pp. 32- 34.

[15] Beugnard, A., Jezequel, J., Plouzeau, N., Watkins, D. (1999). *Making Components Contract Aware*, IEEE Computer, July 1999, pp. 38-46.

[16] Khan, K., Han, J., Zheng, Y. (1999). *Security Properties of Software Components*. 1999 Information Security Workshop (ISW99), Kuala Lumpur, Proceedings, Springer, Lecture Notes Computer Science, 1999, vol. 1729, pp. 51-56.