Assessing Security Properties of Software Components: A Software Engineer's Perspective

Khaled M. Khan
School of Computing and Mathematics
University of Western Sydney
Locked bag 1797
S. Penrith D.C.
NSW 1797 Australia
k.khan@uws.edu.au

Jun Han
Faculty of Information and
Communication Technologies
Swinburne University of Technology
PO Box 218, Hawthorn, Melbourne
Vic 3122 Australia
jhan@ict.swin.edu.au

Abstract

The paper proposes an assessment scheme for the security properties of software components. The proposed scheme consists of three stages: (i) a system-specific security requirement specification of the enclosing application; (ii) a component-specific security rating; and (iii) an evaluation method for the scored security properties of the candidate component. The assessment scheme ultimately provides a numeric score indicating a relative strength of the security properties of the candidate component. The scheme is partially based on ISO/IEC 15408, the Common Criteria for Information Technology Security Evaluation (CC) and the Multi-Element Component Comparison and Analysis (MECCA) model. The scheme is flexible enough for software engineers to use in order to get a first-hand preliminary assessment of the security posture of candidate components.

1. Introduction

With the rapid development in the field of component based software engineering, and the increasing recognition of security risks involved with using third-party software components, there is a need for an assessment scheme for the security properties of components. A component may be used in varieties of application running in numerous types of environments in its entire life time, and may play various roles. The security provided by a third-party software component does not usually address the security requirements of all possible application types in all execution environments. It is not realistic either to expect such universal capability of a component.

A component may be proved secure in one application in a particular operating environment, but the same component may not be considered secure at all in a completely different application. For example, a component considered reasonably secure in an application for car manufacturing plant, may not be secure in an air traffic control application because the use contexts and the security requirements are different although the functionality provided by the component remains same for both applications. We argue in this paper that the security posture of a candidate component should be assessed against the security requirements of the specific enclosing application before it is integrated into the system.

Generally speaking, software engineers lack confidence in third-party components because they cannot assess the security compatibility of the components for their applications. Making a mere security claim such as 'secure' on the component label would not help much to develop software engineers' confidence in third-party components. Often the security provided by the component does not exactly match with the security requirements of the enclosing application, and the software engineers would rather like to include additional security features to the components. Software engineers need an assessment scheme with which they can have a first-hand assessment about the security services provided by the candidate components for their applications. Such a scheme helps them to select a particular component which will likely satisfy the security requirements of their enclosing application.

The Common Criteria (CC) [6] provides evaluation measures of security. CC provides a common set of requirements of the security functions of a system, and a set of evaluation measures. The entire approach is quantitative, and it issues a formal certificate to the sys-

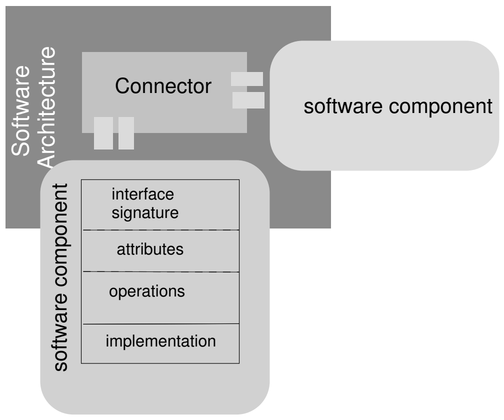

Figure 1. Software Component and Composition

tem. However, CC suffers from some distinct limitations. *First*, it does not address the evaluation methodology under which the criteria and assurances measures may be applied by the evaluator. *Second*, it also does not provide any process for the preliminary assessment of the security functions of a system before a system is built or integrated as a complete software. *Third*, the assessment of systems security capabilities applying CC assurance measures requires huge efforts, and expertise. Many enterprises cannot afford the costs required for the CC evaluation.

To address these deficiencies, this paper proposes a framework for evaluating security properties of software components before integrating a complete system. This paper evolves around three issues: (i) specifying system-specific security requirements; (ii) rating component-specific security properties; and (iii) an evaluating method. The scheme would enable the software engineers to infer whether a candidate component would likely be able to meet their security requirements.

A first-hand assessment would help the software engineers to decide what additional security measures might be required for their specific application systems. The security level that an application system requires may not comply with the available features provided by a candidate component. If a critical security function shown in the assessment appears to be weak, adapters such as glue codes could possibly enforce additional security policy to overcome the weakness. Software engineers may require to write security wrappers to enforce additional security policy to the components. A complete security assessment profile of software components could be used to identify incompatible components from compatible one available in the open market.

It is also vital to put in place an assessment scheme by which an independent certification authority can verify the security properties already in-built into the component, and their suitability for a specific application. By evaluating a component security profile, software engineers could develop a level of confidence in the components.

The paper is organised as follows. The next section addresses the fundamentals of components, compositions and their security concerns. Section 3 proposes an assessment scheme and its associated stages. Section 4 presents an example to demonstrate the applicability of the scheme. A summary of the related work in this field is briefly presented in section 5. Finally, we close with a conclusion in section 6.

2. Software components and composition

Software components may be available in many different forms ranging from procedure and object libraries up to stand alone applications. A software component may be already composed of other components. In this paper, we are interested in the atomic component that are not built on other components. Component based development has two major aspects: components and composition. Each individual component has a scope of its own security responsibility. Similarly, the compositional architecture has also its defined scope of responsibility regarding the security.

In Figure 1, we show *a component* is integrated with another component through *a connector* in an compositional *architecture*. During the execution, the component may bind

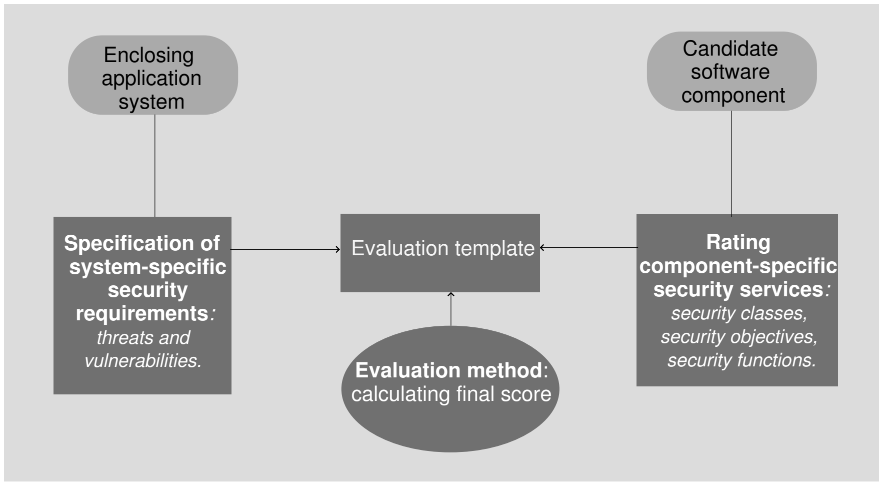
Enclosing application system
Candidate software component
Specification of system-specific security requirements: threats and vulnerabilities.
Evaluation template
Rating component-specific security services: security classes, security objectives, security functions.
Evaluation method: calculating final score

Figure 2. The Assessment Scheme

the local resources of other components and could access the resources by invoking methods on the resources. Information and control data is typically transferred from component to component through a connector. Events, procedure calls, pipes, shared memory, shared databases, object request brokers are the examples of connectors. Connectors basically provide infrastructure for communication between components. The security responsibility for the connector is outside the scope of the individual component.

Interface signature is the access point of the component where all communications to the component take place. When there is no standard interface available in some applications, composers use adaptors or wrappers to control the interface. In some cases, however, object adapter or wrapper may be involved in authenticating the request made by other components. The security properties and features of the wrapper is also not the concern of the individual component. The security mechanism of the connector and the wrapper is the prime responsibility of the composing architecture. The *attributes* and *operations()* are the property of individual components. The security of component is concerned with these properties of component.

3. Assessment scheme

Our proposed assessment scheme has three stages related to the evaluation of security properties of software components as depicted in Figure 2. The first stage, specification of system-specific security requirements for individual functionality of the enclosing system, involves identifying threats, vulnerabilities, risk etc. of the system. The selection of candidate components for the enclosing system is based on the security requirements of individual functionality in the overall architecture of the system. For example, an enclosing system s requires functionalities $f_1,\ldots,f_n$ from different components. Security requirements r_i are specified relative to the individual functionality f_i of the enclosing system. In this paper we are not going to address this issue as it is assumed this specification is already defined.

Regarding the second stage, the *rating* is a term used in this context to assign a numeric value or score to each *security service provided* by a candidate component. The score is assigned only to those properties which are relevant to the system-specific security requirements identified for the enclosing system in the first stage. This is necessary in order to find the relevancy of the component's security to the application's requirements of a particular functionality. We use a template in order to rate the security properties which we call an *evaluation template*. The third stage is an *evaluation method* of grading the overall security properties of the component in terms of their strength and weakness. We discuss the rating of *security services*, and the *evaluation method* of the services in detail in the following subsections.

3.1. Security properties and evaluation template

The *evaluation template* for the security properties of a candidate component is based on the security classes defined in the Common Criteria (CC). CC is a standard, namely ISO/IEC 15408, concerning the security evaluation of IT products. CC describes the security behaviour or functions expected of an IT system to withstand threats. The

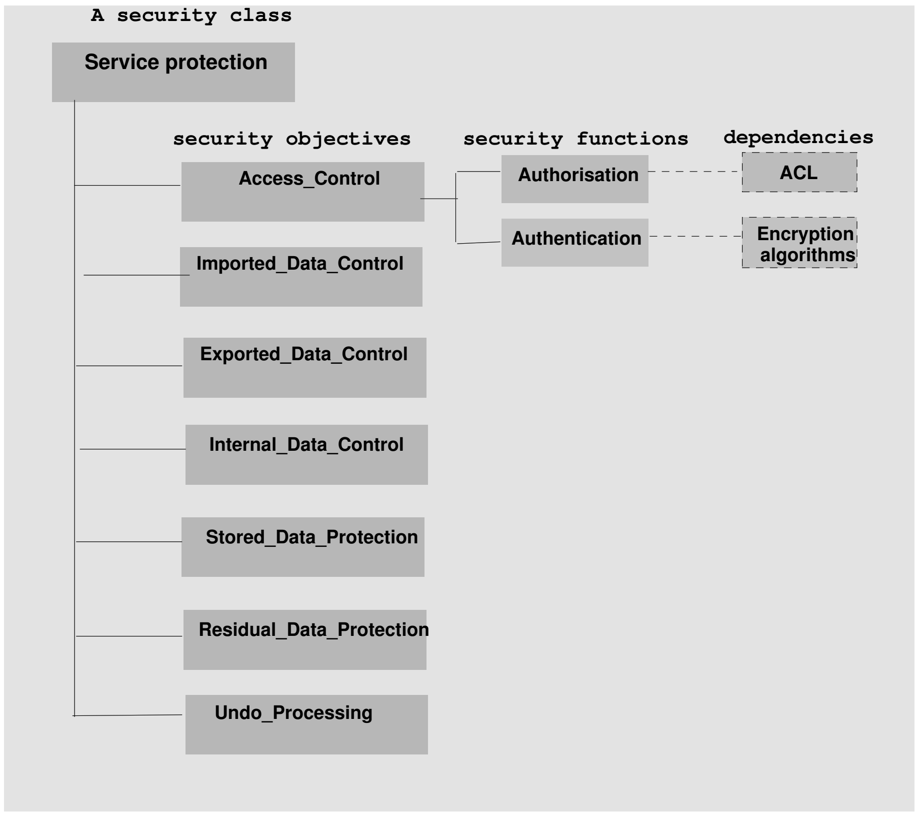

Figure 3. An example of an evaluation template

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 IT 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)'.

Similar to CC, in our proposed approach an evaluation template representing a security class comprises several *security objectives* relevant to the class as the top element in the hierarchy, a collection of defined *security functions* supporting the corresponding security objective, and an arbitrary number of *dependencies* for the security functions. Figure 3 represents an example of an evaluation template of a security class called *service protection*, and its associated objectives and the supporting security functions.

A component may support more than one security class, similarly, a system-specific security may span more than one class. In that case, one template represents one class. Note that only those security classes, their associated objectives and functions of the component are represented in the template which are relevant to the security requirements of the enclosing application.

A *security objective* corresponds to a collection of *security functions* or properties that are used to achieve a prede-

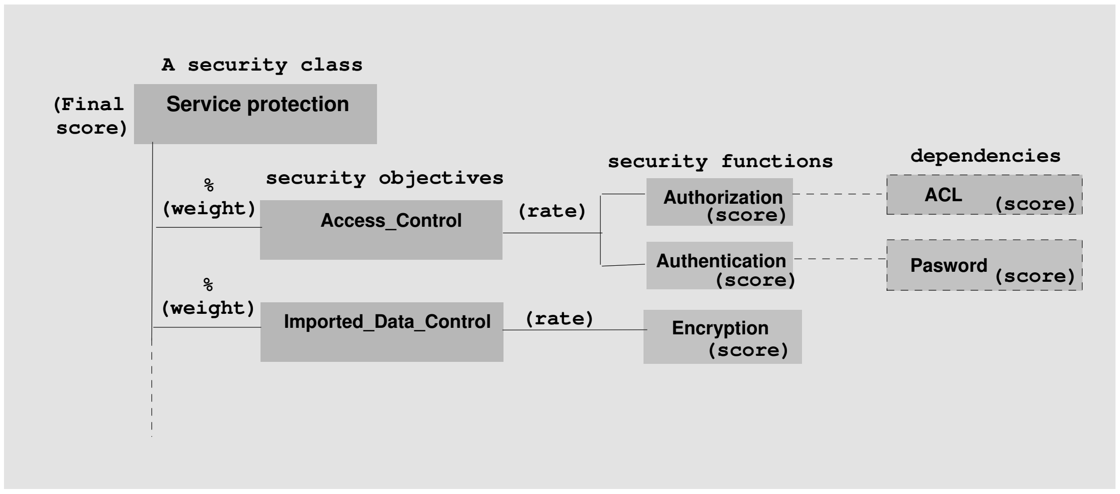

Figure 4. Structure of the evaluation method

fined security goal. In other words, the implementation of a security objective is represented in terms of one or more security functions. A single or a collection of security functions may be used to withstand one or more threats defined in the security objective. A security function may be dependent on other security properties. In that case, a *dependency* exists between a function and other security properties. Dependencies are the most lowest level elements in the template hierarchy. Dependencies are usually not the direct implementation of security objectives, rather they are specific techniques, protocols, files, or standards. The example in Figure 3 shows that the class *service protection* consists of seven defined *security objectives*. The security functions of the security objective *Access_Control* are *Authorisation* and *Authentication*. The dependencies of the functions *Authorisation* and *Authentication* are the Access Control List (ACL), and *Encryption algorithm* respectively.

This stage identifies the security functions of a component in terms of events to the component functionality, and the effect of each of this event. A security function or property is stimulated by either an external or internal events. The identification of relevant security functions also includes the source and entity of this stimulus input [1]. The behaviour of a security function elicited from the stimulus is recorded. It also notes the condition under which a security function could possibly be bypassed. Note that software engineers can identify any number of security classes, the associated objectives and security functions according to the system-security requirements of their applications.

The rationale behind each of the identified security function may be examined to determine which threat is to be addressed with this property, and how it addresses the threats or vulnerabilities specified for the enclosing system. Software engineers are in far better position to identify the possible threats to their application systems. They could decide which threats are redundant, that is, whether threats are covered by another security property in the same functionality of the candidate component. It is quite possible that one single security property may address one or more threats of the application.

3.2. Rating component-specific security services

To compute the final score for the overall security properties identified in the rating process, we have adopted the Multi Element Component Comparison and Analysis (MECCA) [4] method. The method has been used as a system evaluation technique. The underlying idea was first proposed by the mathematician Zangerneister in 1970. The technique was also employed in the assessment of software maintenance tools [8]. The structure of the evaluation method is shown in Figure 4. The *dashed rectangle* in the figure denotes the dependent functions, and the *rate* shown in the figure is used to contain the *score* for a security function.

The evaluation method uses a percentage weighting to the security objectives. A percentage weight is assigned to each security objective through out a given class. Each class is also given a percentage weight relative to the importance of other security classes. At a class level, the percentage of the weights of all classes would add up to 100, although we have not discussed other security classes in this paper. Similarly, accumulated percentage weighting of the security objectives in a given class would be always 100. The percentage weighting is defined by the software engineer depending on the importance of the individual security objectives.

Each security function is assigned a direct numeric score if it does not have dependencies. A higher score reflects a higher strength. The numeric score ranges from 0 (unknown or nil compliance), 1 (weak compliance), 3 (moderate compliance) to 5 (strong compliance). Scoring for a particular security function or dependency is based on the deviation between the software engineer's expectation for the security requirement of her application and the provided security function of the candidate component. For a wider gap the score should be at the lower value of the scale 0 - 5. This gap could be determined based on various security properties and sub-properties of the provided security function of the component.

Gap between the 'as-is' of the component security and the 'required' of the enclosing system is translated into this measuring scale. A zero gap between the component-specific 'as-is' security properties and system-specific security requirements may incur a highest score of 5, which is translated as fully complied. If a particular security property scores 4, it means the property has better compliance than average. A zero score may indicate that either the property does not exist, or has a nil compliance.

Consider the following example. A cryptographic security function could be assessed in terms of three dependencies: (i) the encryption algorithm used to generate the key; (ii) the standard or protocol used; and (iii) the key length. Suppose, a candidate component generates a key of 512 bits length with the algorithm $MD5$ using the standard $S/MIME$. If these properties match the security requirement of the application, a score of 5 is assigned to the security function. If the expectation of the key length is 2048 bits instead of 512 bits, a lower score than 5 is assigned to the security function. The following generic guidelines could aid the scoring process:

- Find a security function f if it does not have any dependency, or find its dependency d specified in the template.
- Check if the candidate component c provides this security function f or dependency d.
- If c does not offer f or d, assign a score 0 to f or d.
- If c offers f or d, break down the properties of f or d into sub-properties such as $f_1, f_2 \dots f_n$ or $d_1, d_2 \dots d_n$.
- Compare the identified sub-properties with the required security properties of the system for the same security function.
- Assign a lower score if the deviation is greater.

No direct score is assigned to a security function if it has dependencies. In such case, the score for the security function is calculated based on the assigned scores to its dependencies. At the component level, the final computation generates a single score. This final score has to be in a scale from 0 to 5, where 5 is the most desirable, and 0 is the least desirable.

The process of assigning scores directly to the security functions and their dependencies is based on a subjective evaluation of the properties. It is necessary to use such an approach in this case because only the software engineers know what security provisions are required for their enclosing applications since security is a moving target as the threat scenario changes very frequently, and it varies application to application. Based on the advertised security functions of the candidate components, the evaluator assigns the score to specific security function relevant to her application. For example, a security assurance of a component for a banking system is quite different than the assurance of the same component in a manufacturing plant.

Regarding the weighting, a scale of 1 to 100 is used in terms of relative importance of different security requirements of a class. For example, a security class has three security objectives which have been identified as the security requirements of an application system. All three objectives have the same level of importance for the application. In this case, the weight should be 33.33% for each of the objectives. It suggests that each objective has equal level of importance as others.

The process of weighting in the proposed method also encourages a subjective assessment of the properties because the evaluators are in a better position to know the actual security requirements of their application and their relative importance of the security requirements. The weighting of a security requirement also depends on the value of the data assets to be protected, the attackability scenario and so on.

3.3. Evaluation method

The general equation to calculate the final score of a security class is,

$$F_{i,j} = MINIMUM(D_{i,j,k}), k \geq 1 \tag{1}$$

$$O_i = MINIMUM(F_{i,j}), j \geq 1 \tag{2}$$

$$C = \sum_{i=1}^{N} W_i O_i \tag{3}$$

where, C is a class, O is an security objective, F is a security function, and W is the percentage weight of an objective. A dependency is represented as $[i, j, k]$, that refers to j security function of the security objective i. Thus, W_i denotes the scores of security function N of the security objective i, and $D_{i,j,k}$ denotes the scores of the dependencies k of the security function j of security objective i.

Software engineers could alter the scoring scale $(0, 1, 3, 5)$ and apply any other scaling method suitable for their applications. They have the freedom to choose more liberal equation such as *mean*, or *maximum* score instead of the *minimum* in order to calculate the score. This flexibility helps developers to find the alternative score emphasizing either on mean value or maximum value of the weighting. However, component developers are not free to modify the principles of the evaluation methods. The three basic principles of the proposed methodology such as the assessment scheme, the evaluation template, and the evaluation method are not subject to change. Note that a *maximum* function is the most weakest form of calculation. The process of assigning score to each individual security function would depend on the various factors such as the capability of the security function to withstand certain threats, reliability, value of data assets, and so on.

Regarding the dependencies, the score of the parent function would be computed from its dependencies. The *minimum* score of all its dependencies would be the score assigned to the parent function. The *minimum* function is used instead of *mean* or *maximum* in order to address the security rule that *a chain is as weak as its weakest link*. A security objective may have more than one security function. In such case, the minimum score of all functions would be the ultimate value for the corresponding security objective.

The proposed method is intended for software components in black-box form. The metrics are collected from the advertised security functions of the components, component functionalities, certificate, available users' guide, and from enquiry, much of which are described in the component interface. In this regard, the proposed security characterization in [7] if implemented, could be used in order to know the actual security profile of the component.

4. An example

The applicability of the proposed assessment model is illustrated with an example. Assume we are going to evaluate the security properties of a component for a banking system. Suppose the enclosing banking system called 'Smart Banking System' requires a *software component* which offers functionalities such as debit and credit functions on an account over the internet and storing the account information. In addition to these functional requirements, the system requires several security assurances from the component. Also assume that quite a number of components in the market can offer the same functionalities that the banking system requires with varied security provisions.

In the *first stage* of our assessment scheme, we identify the following system-specific security requirements: (i) access to the account is restricted. Only authorised entities can operate on the account; (ii) The account information or operation received from the outside of the system boundary must be encrypted and authenticated. These are necessary to ensure that data is not observed by other entities, and the sender of data is authenticated; (iii) account information transmitted to the outside of the system boundary must be digitally signed and encrypted in order to guarantee the authenticity of the bank and the confidentiality of the data; and (iv) account data stored in the storage devices such as hard disk or tape must be protected in such a way that no unauthorised entity can have access to data.

In the *second stage*, we map the above system-specific security requirements into an evaluation template for the rating of component-specific security properties. For simplicity, we assume that all specified security requirements of the functionalities required by the 'Smart Banking System' fall in one security class called 'data protection'. Each of the four security requirements is defined into a security objective as shown in Figure 5. The template in the figure addresses the system-specific security requirements specified in the first stage. Notice that some of the requirements are represented as security functions.

In the *third stage* of the assessment scheme, we assign percentage weight to each security objective based on the importance, and give scores to the security functions if there is no dependencies, otherwise we assign scores to the dependencies. The assigning percentage weights to the security objectives depend on the relative importance of each security objective compared to others. The scoring is done based on the information captured about the candidate component, component manual, component developers, and examining the component security capability by stimulating its security related events.

We have defined the corresponding percentage weight of each of the security objectives and assigned scores to the security functions and dependencies as shown in Figure 6. The security objective $Access\_Control$ has one security function called $Authorisation$ with two dependent functions: one is ACL and the other one is $Password$. ACL is assigned a score of 5 (strong), and $Password$ is given 3 (moderate). The *minimum* of these two dependent functions is 3, and assigned to the parent security function $Authorisation$ according to $equation\ (1)$. The relative importance of the associated security objective of this class is 30%. Hence, the calculated score of the

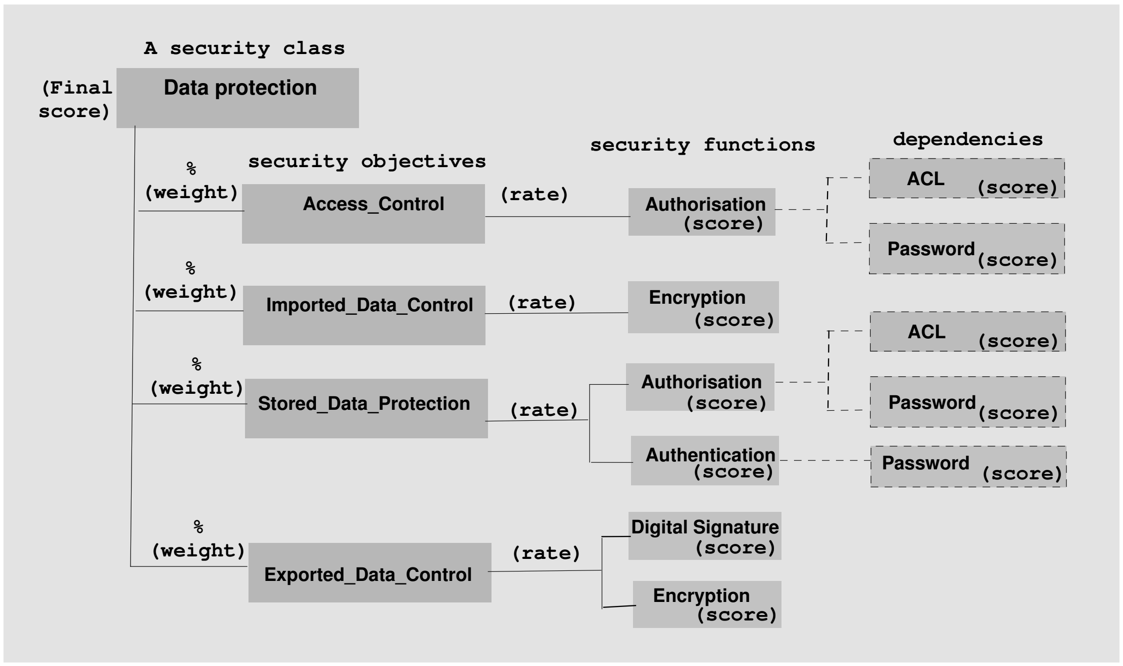

Figure 5. Evaluation template for the banking system

security objective $Access\_Control$ would be 0.90 according to $equation(2)$.

The next security objective $Imported\_Data\_Control$ has one security function, $Encryption$ with a score of 1 (weak). The importance of the objective is 30%, hence the value is calculated as 0.30 for $Imported\_Data\_Control$.

Security objective $Stored\_Data\_protection$ is based on two independent security functions: $Authentication$ and $Authorisation$. $Authentication$ has one dependency called $Password$, and it is assigned a score of 3 (moderate). Hence, $Authorisation$ is given the score 3 based on its dependent functions. $Authorisation$ function has two dependencies: ACL and $Password$, and are assigned 5 and 3 respectively. The score of the parent function is 3 based on the minimum of the two dependencies. The weight for the corresponding security objective $Stored\_Data\_Protection$ is .60 based on its 20% importance.

The fourth security objective $Exported\_Data\_Control$ has two security functions. $Digital\_Signature$ and $Encryption$ are assigned 5 and 5 respectively. The rating is 5, calculated based on the minimum of these two scores according to $equation(2)$. The corresponding objective $Exported\_Data\_Control$ has a relative weight of 1.00.

According to $equation(3)$, the accumulated score of this class is 2.8 which is considered not a strong security measure in a scale of 0 (unknown or none) to 5 (strong). This example demonstrates that this particular class does not have a strong security properties provided by the candidate component as a whole. Especially, the security objective 'Imported_Data_Control' is weak, and its score 1 has contributed to a lower rating of the component security. Obviously, with this score this component will unlikely be used by the software engineer. However, if the software engineer still decides to use it, she needs to add additional security wrapper or adapter to increase its capability score from 2.8 to a higher value. After adding the additional security features, the component is to be reassessed using the same scheme.

The metric provides valuable indication on the candidate component's likelihood of satisfying the security requirements of the enclosing system. The final score could be used as an indicator on how satisfactory a component security is for the enclosing system's security requirements. If a set of components are assessed against a specific security requirement of an application, the software engineer is in a better position to have a comparative ranking of the components' ability to conform the requirement. For example, a component may score 4.5 and an-

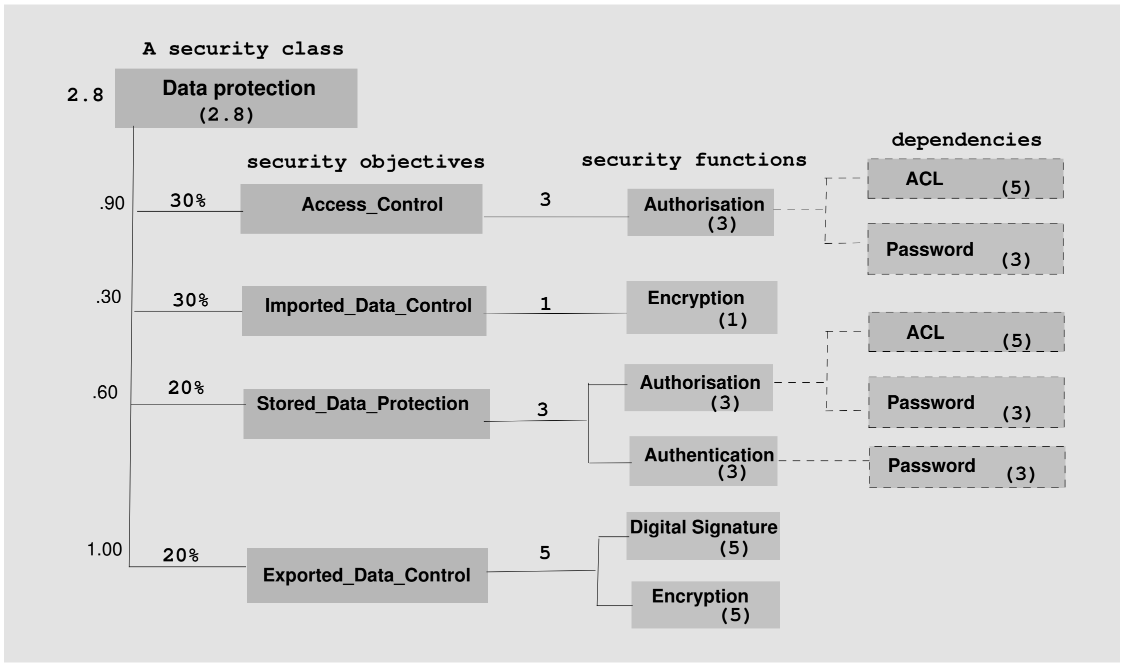

Figure 6. Component-specific security properties and rating

other has a score of 3.5. This suggests that the component with 4.5 is a better option than 3.5. Without such metric, software engineers are unable to rank them.

Numeric values are used to assess the security properties provided that numeric values express the relative measure of the quality of the function. The score could be generated from the judgement on the gap between the observed property defined in the evaluation template and the required properties defined by the software engineer for the application. We define the 'level of strength' as the degree of the likelihood of its effect and capability in a given context. A security function would be considered strong (highest score) if it is capable of withstanding all identified threats, and satisfies all identified security requirements.

The numeric score assigned to the individual security function dictates its relative strength and weakness. During the scoring of the functions, each of them would be judged based on the intended application, implementation of the policy, technology used, possible limitation, potential value or assets that they protect, the threats countered, the possible contradictions to other security policies, and so on. For example, an encryption function could be assessed in terms of the algorithms used in its implementation to compute the key such as RSA, DSA with cipher, DES, ElGamal, HMAC; the length of the key such as 56-bit, 128-bit, or 1056-bit; the cryptographic standard used such as IPSec, SET, S/MIME, SSL.

The assessment scheme, the evaluation template, and the evaluation method are independent of any particular application contexts. Any application specific security objectives, security functions and dependencies could be used in the proposed template without any modification.

The proposed methodology might not be suitable for assessing other non-functional properties such as efficiency, usability, maintainability, portability etc. This particular method is specific to security because the strength of the method lies in the breakdown of the security-specific properties into the objectives, functions and dependencies. Other non-functional properties may not be divided into similar sub properties. A further study is required to examine whether an extended version of the evaluation template could be used for other non-functional properties.

5. Related work

There is a lack of literature in the public domain on this field. The FoundScore [3] is a security metric that can be used as a guide to measure the risk and the business value of expenditures to information security. The paper addresses issues such as why an organisation should use security metrics to understand risk, and how to track security improvements using *FoundScore*. FoundScore is a security rating system that compares security aspects of an organisation against best practices in order to quantify the security risk. The approach assesses vulnerabilities and risk of an organisation, and calculates the cost for the security measures taken to address the identified vulnerabilities and risks.

A seven-step methodology has been proposed in [9] to guide the process of defining security metrics. The approach basically yields an understanding of the purpose of the security metrics program, its deliverables, and how, by whom and when these deliverables will be provided. Berionto [2] has recently reported on five security metrics proposed by Andrew Jaquith. It argues that a constant measure of security incidents is a great indicator of the security posture, and it could be used to quantify the efficiency of the deployed security functions.

National Institute of Standards and Technology (NIST) defines security metrics guide for information technology systems [10]. The document provides guidance on how an enterprise through the use of metrics identifies the security needs, security controls, policies and procedures. The approach guides management of an organisation in deciding where to invest in security. An approach to measure the attack surfaces is recently reported in [5]. It proposes metrics to count and measure a system's attack opportunities. This count is used to indicate a new security metrics called the system's attackability. The approach uses three abstract dimensions to describe a system's attack surface such as targets and enables, channels and protocols, and access rights.

6. Conclusion

In this paper, we have proposed an assessment scheme for security properties of software components. The scheme is based on a rating process and an evaluation method. Component security profiles are built by identifying provided security functions of the candidate component, and then the captured properties are assessed against system-specific security criteria. The evaluation method presented here is considered flexible enough as some of the calculation and measuring scale could be altered by software engineers as they see appropriate for their assessment. Producing a security profile as proposed in this paper for assessing component security properties could be a viable supporting method to other security evaluation approaches. The security testing such as dynamic black box testing of component or the process of fault injection could be augmented with our proposed assessment scheme. By knowing the security properties of the components, testers can design their test cases to find the security holes, fault tolerances, and weakness of the component.

An assessment profile provides the software engineer and certifier a framework to determine whether the security properties would pass their required security threshold or not. This would also allow them to select a suitable component from a collection of competing components. This may ultimately inspire the component vendors and developers to build better quality components regarding their security.

However, the evaluation of security profile of components may not give any such guarantee that the component is secure, instead, the information could be easily evaluated by the user themselves against their application requirements. The goal of our work is to provide a security assessment scheme such that software engineers would be able to know and judge *a priori* how a candidate component would pass the threshold defined for the overall security of their enclosing system. Our scheme is based on the compatibility between the security performance of the component and the required security of the enclosing system.

References

[1] Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, Boston, 2003.

[2] Berinato, S., "A Few Good Metrics", CIO-Asia Magazine, September 2005.

[3] Foundstone, "Information Security Metrics", White paper, Foundstone Strategic Security, April 2003.

[4] Glib, T.: Software Metrics, Cambridge MA, Winthrop, 1977.

[5] Howard, M., Pincus, J., Wing, J., "Measuring Relative Attack Surfaces", Chapter 8, in Computer Security in the 21st Century, Springer-Verlag, 2005, pp. 109-137.

[6] ISO/IEC 15408. Common Criteria for Information Technology Security Evaluation. NIST, USA, http://csrc.nist.gov/cc/, June 1999.

[7] Khan, K., Han, J.,"A Security Characterisation Framework for Trustworthy Component Based Software Systems", Proceedings of the COMPSAC'03, IEEE Computer Society, 2003, pp. 164-169.

[8] Khan, K. M., Ramakrishnan, M.K., Lo, B., "Assessment Model for Software maintenance Tools: A Conceptual Framework" Proceedings of PACIS Pasific Asia conference on Information systems, QUT, Brisbane, 1997, pp. 527-536.

[9] Payne, S., "A Guide to Security Metrics", SANS Institute 2002.

[10] Swanson, M., Bartol, N., Sabato, J., Hash, J., Graffo, L., "Security Metrics Guide for Information Technology Systems", National Institute of Standard and Technology (NIST) Special Publication 800-55, July 2003.