Security Properties of Software Components

Khaled Khan, Jun Han, Yuliang Zheng

Peninsula School of Computing and Information Technology.
Monash University
McMahons Road, Frankston
VIC 3199 Australia.
Fax: +61-3-99044124
{khaled,jhan,yuliang}@mars.pscit.monash.edu.au

Abstract. This paper classifies security properties of software components into two broad categories: (1) non-functional security (NFS) properties, and (2) properties as security function (SF). Non-functional security properties are codified and embedded with the component functionality, whereas, properties as security functions are employed as external protection to the component. In most cases, users may add additional external protection to the binary form of the component. This classification could be used to determine how much the overall security of the component is dependent on the non-functional security properties of the component and to what extent the additional external protections are required in order to use the component in their specific application environment.
Keywords. Software component, Security properties, Component functionality

1 Motivation

Software component security is concerned with the protection of the resources, data, and computational properties of individual components and the enclosing system. The security of software component is very much dependent on the role that the component plays in a specific application environment. Software components need to be customised with the application environment where it is deployed. In most cases, third-party software components need external protection employed by the users to meet their specific application requirements. Due to the binary representation of the component, software composers are not able to modify the security properties that are embedded at the implementation level. We believe that overall security properties of software components should be classified into two categories, and this information should be available for users' inspection before a candidate component is selected.
There is a growing concern on the issue of characterising the component security properties in recent days as expressed in [?], [?], [?], [?] ,[?], [?]. The purpose of this paper is to identify various sources of security features that are codified with the component functionality. Most of the research conducted in recent days related to component security are involved in detecting and removing the

security flaws of components. We believe that security of component should be considered differently than the case of application systems because of the distributed usage of the component. In this paper, we are not going to propose any new security technique or assessment, rather we define different types of security properties that are placed various ways enforcing certain security objective of the component.

2 Classes of component security properties

In this section, we define two distinguished classes of security properties related to software components such as *properties as security function (SF)* and *non-functional security (NFS)* properties.

Properties as security functions (SF) are those that provide secure encasing to the components, protecting the component from any security threat from the outside of its boundary. External security features can be added to components as functions. We call them *security functions (SF).* On the other hand, *non-functional security (NFS)* properties are inherited in the internal implementation of the component, that is, some security features are codified with the functionality that the component provides. We term these features as *non-functional security properties (NFS).* Usually, NFS properties are some security enforcement mechanisms that are already embedded in various forms with the functionality of the component at the implementation level. NFS properties are, in fact, the implementation of highly abstract security objectives that are intended to counter certain security threats. NFS properties are attached with various aspects of the component functionality in different layers of implementation, each representing a specific level of abstraction to achieve certain security objective. Whereas, SFs are considered some kind of external protection mechanisms of components which are not directly related to the component functionality. SFs can be designed and added to the existing component. For example, a wrapper can be written to protect the component from a potential security threat. The authentication of component origin and identity is considered as a SF. This type of SF protects the enclosing system from being assembled with an unauthorised component. Particularly in a dynamic assembly scenario, a target component may be located in a remote server whose identities may or may not be authenticated.
NFS properties are embedded with the component functionality to prevent the threat of usage violation. A component may employ certain NFS properties to guard its sensitive data and functionality from being violated by other unauthorised entities. The NFS properties embedded with the component's functionality may have substantial impact on the entire security mechanism of the composed system. It is not enough to make a system secure by adding SF such as encryption technology or secure protocols for data transmission to the component if the implementation of the component has security flaws. A great effort in designing SFs such as a strong authentication and access control mechanism can

be easily ruined by a single security flaw embedded with the functionality [?]. If there is a weak NFS property in a sensitive operation, the rest of the strong SFs may not help much to protect the component. Authorisation mechanisms such as access control cannot always be guaranteed in a single access control mechanism, instead further protection may be required in terms of NFS properties to protect the functionality and the internal data of the component. In the next section, we will see how these two types of security properties enforce certain security objectives in various ways. To illustrate these two phenomena of component security properties we analyse a simple example in the following section.

3 An example

In our example, *a medical system used by the medical practitioners is dynamically assembled with a software component time to time as needed. The component caters the diagnosis reports on patients to the legitimate users. 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.*

The service of the component in which we are interested in is a set of operations supported by an interface structure. The interface name and its associated attributes are visible to the users, but the operations are not. The various layers of SF and NFS properties identified in the example are illustrated in Fig. **??**. We show in this figure how different security properties enforce certain security objectives by analysing the execution behaviour of the cited example. The circles denote various layers of security properties both as SF and NFS, and the arrows show the sequence of the execution behaviour of the component.

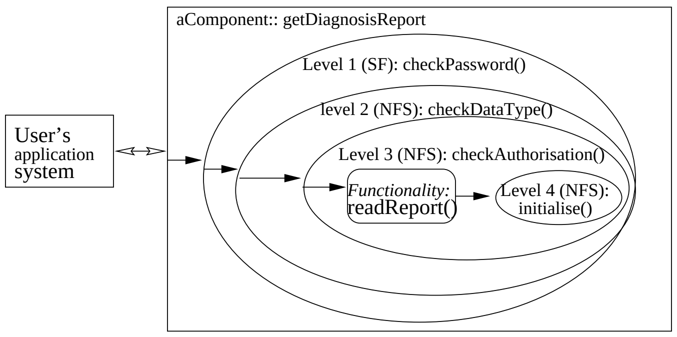
aComponent:: getDiagnosisReport
Level 1 (SF): checkPassword()
level 2 (NFS): checkDataType()
Level 3 (NFS): checkAuthorisation()
User's application system
Functionality: readReport()
Level 4 (NFS): initialise()

Fig. 1. SF and NFS Properties at Various levels

3.1 Identifying security properties

In the top level of the layer, we can find a SF concerning with the authentication and access control of the user to the component. The function is implemented by the operation *checkPassword()*, and used to protect the component from being accessed by any unauthorised entities. The function allows the user to assemble the component with the application system. It is not related to the functionality of the user system or the component, hence labeled as SF. Second level of the security is provided in terms of a NFS property implemented by the operation *checkDataType()*. This operation is closely related with the component functionality since it computes the output based on the input received from the user system. For a smooth processing, this NFS property is used to ensure that all input data values must be in correct type and range, but it has also a security side-effect that was not designed intentionally for security purposes. It is often reported that incompatibilities of the input values may lead to undesirable security problems of the component. Next level security in the hierarchy is provided by the operation *checkAuthorisation()* just before the main function *readReport()* of the component is executed. This operation does not employ any authentication or password to verify the authorisation, rather a tuple associated with the *patientID* in the database file verifies this. If the patient tuple in the database entry does not contain the *userID* value, the system would not authorise the user to read the data associated with the *patientID*. Finally, the security property supported by the operation *initialiseAllVariables()* in the layer 4 is a NFS that simply initialises all internal variables to enforce certain security objective, because users may leave a trail of *data shadow* with the contents of the data they manipulated [**?**].

3.2 Observations

We observe following characteristics from this simple example.

- some operations are implemented as underlying logical framework into the component functionality, and act as security side effects
- security properties propagate from high level to the low level security properties
- users interact directly with the SF
- users do not have direct interaction with the NFS properties
- users are unaware of the presence of the NFS properties unless any of their actions violates the NFS properties
- SF property acts as an external protection to the component
- NFS properties protect the internal data and operations as well as the functionality of the component.
- some of the NFS properties were not intentionally programmed for security purposes.

However, it is quite possible to uncover more NSF properties from this example if we analyse the implementation details of the operations, like call sequences, parameter passing, file permissions, and so on.

4 Summary and further work

In addition to the characterisation, all these extracted SF and NFS properties could be further assessed in terms of their strength and weakness. The presence of several SF and NFS properties in a functionality do not dictate that the function is secure, rather their relative strength and implementation mechanisms must be taken into active consideration. The characterisation and assessment of SF and NFS properties can be augmented with the *functional and assurance requirements* defined in the *Common Criteria for the Information Technology Security Evaluation, version 2.0 ISO/IEC 15408* [?]. If we can manage to characterise the SF and NFS properties, and use them as component label with proper sealing technique using a digital signature, it may increase the user confidence on the component technology. The security characteristics on the label can also guide the user to decide whether a component would be able to meet the specific security requirements of their application or not. The NFS information published on the component label would assist the user to design and add additional security protection to the component to meet their specific security requirements. Simply adding security functions onto an existing component may not be as effective as if they were embedded with the component functionality. It is the internal computing properties that determine the ultimate behaviour of the component. We are currently exploring the possibility of applying Common Criteria requirements to software components to characterise the security properties of components. Such an effort may lead to defining a formal characterisation model that would help us to specify properly all SF and NFS properties of software components. Our research objective is to formalise a concrete characterisation framework, which could be used as the basis of software component certification.