INTRA-COMPONENT SECURITY CERTIFICATION

Dissertation submitted to Jawaharlal Nehru University
in partial fulfillment Of the requirements
for the award of the degree of

Master of Technology

in

Computer Science and Technology

by

BALACHANDU GUDAVALLI

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI – 110 067
JULY 2007

CERTIFICATE

This is to certify that the dissertation entitled **"INTRA-COMPONENT SECURITY CERTIFICATION"** being submitted by **Balachandu Gudavalli** to the School of Computer and Systems Sciences, **Jawaharlal Nehru University**, New Delhi in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Computer Science and Technology**, is a bonafide work carried out by him in **School of Computer and Systems Sciences**. The matter embodied in the dissertation has not been submitted for the award of any degree or diploma.

[illegible]

Prof. R.G. Gupta
Professor, SC&SS
Jawaharlal Nehru University
New Delhi – 110 067

[illegible]

Prof. N. Parimala
Dean, SC&SS
Jawaharlal Nehru University
New Delhi – 110 067

G. Balachandu.

Balachandu Gudavalli
M.Tech, SC&SS
Jawaharlal Nehru University
New Delhi- 110 067

<u>**CERTIFICATE**</u>

This is to certify that the dissertation entitled "**INTRA-COMPONENT SECURITY CERTIFICATION**" being submitted by **Balachandu Gudavalli** to the School of Computer and Systems Sciences, **Jawaharlal Nehru University**, New Delhi in partial fulfillment of the requirements for the award of the degree of **Master of Technology** in **Computer Science and Technology**, is a bonafide work carried out by him in the **School of Computer and Systems Sciences**. The matter embodied in the dissertation has not been submitted for the award of any degree or diploma.

[illegible]
Prof. R.G. Gupta
Professor, SC&SS
Jawaharlal Nehru University
New Delhi – 110 067

[illegible]
Dean, SC&SS
Jawaharlal Nehru University
New Delhi – 110 067

Prof Pa imala N:
De n
School of Compu er & Sy tem Sciences
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067

ACKNOWLEDGEMENTS

I am very glad to express my sincere gratitude to my supervisor, Prof. R. G. Gupta for his valuable guidance in completing this thesis successfully. I feel it is a great privilege to have had the opportunity to work under his prestigious supervision. His constant encouragement and support helped me a lot.

I am thankful to Prof S. Balasundaram for providing a congenial environment and System software lab facilities in the school.

I am also thankful to Anurag Dixit for his valuable suggestions provided during my dissertation work.

Finally, I would like to thank to Ansu Kumar Sinha, Sonia Wadhwa and all other friends for their discussions and help at all times during the course of my project work.

G. Balachandu

Balachandu Gudavalli

CONTENTS

ABSTRACT

This dissertation deals with the up gradation of security certificate of a component. As security certification is necessary for third-party components, taking certification from the beginning whenever a component is updated turns out to be a redundant process. It leads to wastage of time especially in large component based software systems. To remove that redundant certification we proposed a CERTDRIVER.

The proposed CERTDRIVER takes the component, its required security properties as security goals and version number specified by user as input and gives output a certified component providing its security in terms of percentages. The whole process centers on version number. In the whole process, it will also take the help of test cases which can be useful while finding out errors in the component.

Chapter 1
INTRODUCTION

1.1 Security classes

Security in components is going to be an essential feature because components are available as third-party products and these have to be assembled with other components. Thus the main requirement lies in providing security to a component with respect to other component or a group of components. So, there is a need for two levels of security, one at component level and other at compositional level. Security will be given to components in terms of their properties [1]. So, there are two security properties namely Security Function (SF) and Non-Functional Security (NFS).

The SF properties are those which provide inter-component security. They will provide security to one component from other components. For giving this type of security, the external security features can be added as functions to that component. These external features can be name of a component as it distinguishes this component from others and encryption technology the component uses as it also distinguishes its communications to specified components with its digital signature on it. SF properties protect the enclosing system from being assembled with unauthorized components. These properties are particularly significant in dynamic assembly scenario where target component may be located in a remote server whose identity can be a questionable one. So, a greater effort is needed in designing these properties. Any violation can cause a serious flaw in design of system.

NFS properties provide intra component security i.e. security within the component. Intra-component security provides security to data within the component.

These security mechanisms are inbuilt. 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. A component may employ certain NFS properties to guard its sensitive data and functionality from being violated by other unauthorized entities. The NFS properties embedded with the component's functionality may have substantial impact on the entire security mechanism of the composed system. 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.

To get a better understanding about these properties, we will take an example of a BookAllotment component. This component can be used in library applications in dealing with allotment of books. First, we will look into the basic structure of this BookAllotment component.

Title of Book
Author of Book
Boolean Variable (Used to denote whether it is allotted)
Allotted Student ID
Allotment and Submission Date
Underlying Implementation

Fig 1.1: Component: BookAllotment

The BookAllotment component retrieves some vital information when it is called. If a book has to be allotted to a student, this component will be called by library application. This component connects user and database and retrieves some information from both ends. While doing all these operations, it maintains some security restrictions in terms of properties.

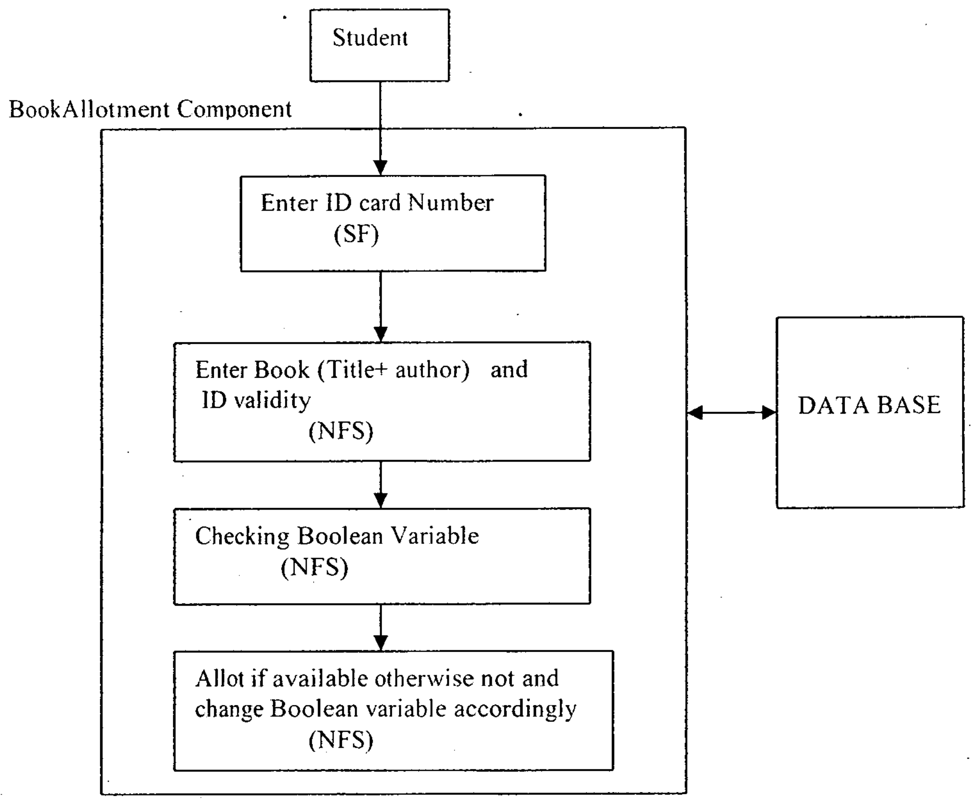

Student
BookAllotment Component
Enter ID card Number (SF)
Enter Book (Title+ author) and ID validity (NFS)
Checking Boolean Variable (NFS)
Allot if available otherwise not and change Boolean variable accordingly (NFS)
DATA BASE

Fig. 1.2: Book Allotment Process

In the above book allotment process, first the BookAllotment component checks the authorization. This we can call as a SF property. We can do this authorization check before calling component also. If this check was done by another application, then that application calls this component, there also this BookAllotment component checks the authenticity of that component. The remaining like checking validity of ID (limit to the number of books allotted), Boolean variable used to denote whether the candidate book is allotted or not, access and modification restrictions on that Boolean variable will come under NFS properties as component itself should protect this from internal operations after retrieving this from database until it writes back to database.

1.2 Secured component

A secured component is necessary to get a secured system. But the complexity lies in what areas the component should be secure. In component based software engineering, there are different contexts in which security has different role to play. To get good understanding, let the contexts be divided into three categories.

- Security based on functionality
- Security based on whole system design
- Security based on communication

In the first category, the main base to say that a component is secure enough is simply checking its functionality. If it is functionally correct then it is secure enough to use otherwise not.

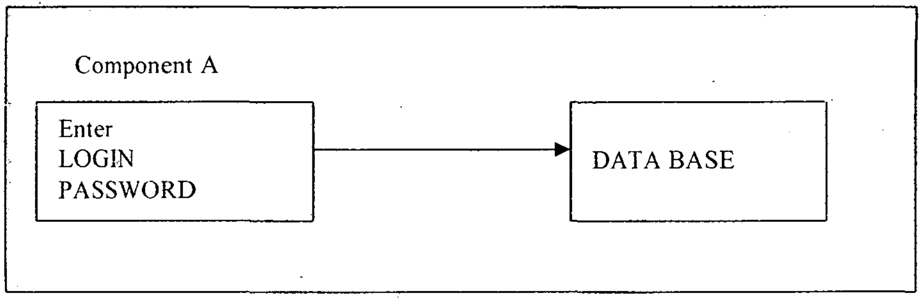


Fig. 1.3: A component interacting with database

In the above diagram, the functionality of component A is to check the validity of the entered LOGIN and PASSWORD with the existing database. Here the functionality of A is to allow any user to enter his login name and mask his password with any symbol and then check with database. If that component A is doing its pre defined task correctly, then it is functionally secure to use it for any application which requires LOGIN and PASSWORD check.

In the second category, the component should be verified according to total system design. If we take the same example discussed above, we have component A which is functionally correct for only a particular database say oracle. But according to system design, the database is of type Microsoft access. Then that component can not function in that database. So, that component is not secure according to specific system design though it is functionally correct.

In the third category, the communications between components are checked to say whether they are secure enough or not. If we consider the same example, the LOGIN and PASSWORD should be encrypted to check with existing database. There may be many components in a system and the communications between them should be encrypted whenever necessary. To get a clear understanding, consider a bank transaction between accounts A and B. How much money is transferring between A and B, from whom to whom, and all information should be secured from other accounts or any other components. Generally, components communicate themselves according to which composition design they belong to. These compositional designs can be of three types. They are shown in below diagram.

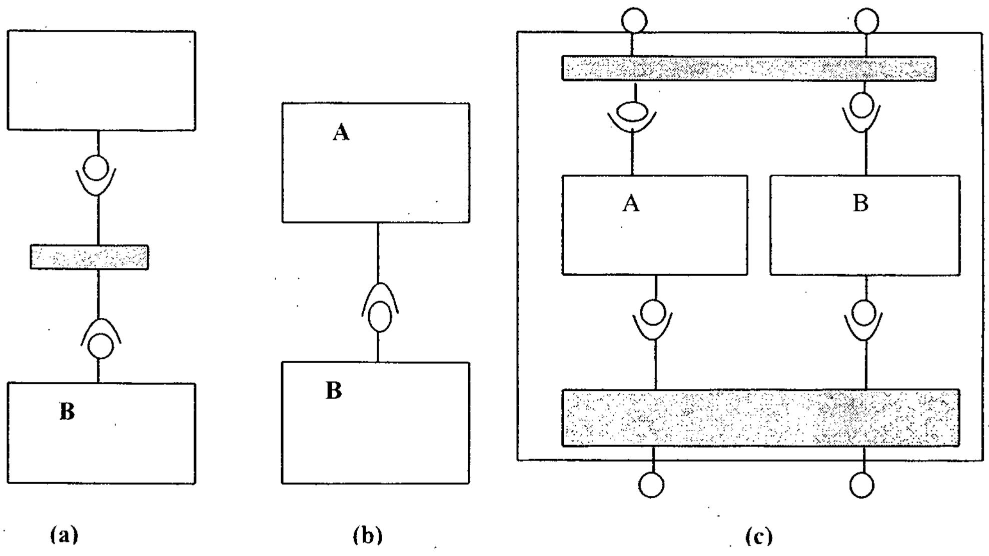


Fig. 1.4: Composition types

In the above diagram, figure (a) is of type sequential composition. In this type of composition, the composed components are executed in sequence. The shaded region is the interface needed for composition. In figure (b), hierarchical composition is shown, in which one component calls the services of another. The interface of one component is composed with required interfaces of another. In the figure (c) additive composition type is shown. In this type, interfaces of two components are put together to create a new component. So, the components should be secure in communication according to compositional design they belong to.

1.3 Certification

In a given set of components, it's really tough for an end-user to decide which component is secure and which is not. He can not check all components as he may not well equipped to do that. How he can trust that a component is secure? Solution for this is certification. The main aim of certification is to make components reliable. A certification can be defined as [2]

"A method to ensure that software components conform to well-defined standards; based on this certification trusted assemblies of components can be constructed."

This certification is also same as giving certificate to students after completion of course. A University will certify a student that he has that enough knowledge in that course. But in components, a third-party will certify it's security. Generally, third-party is a standard organization like NSA (National Security Agency), NIST (National Institute of Standards and Technology), and TCSEC (Trusted Computer Security Evaluation Criteria). Certification procedure is shown in below diagram [2].

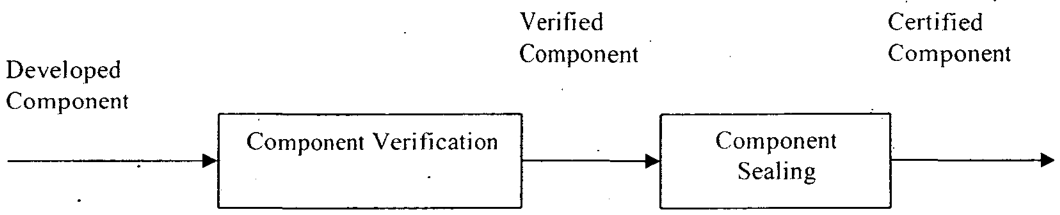


Fig. 1.5: General component certification procedure

The general component certification procedure is divided into component verification and component sealing. The verification step can be done through two approaches. One is Usage-based verification and the other one process-based verification. Usage-based verification deals with verifying final product. Process-based verification concentrates on the production process. The developed component is first verified by third-party organization and from that process, a verified component will be evolved which is later sealed by that organization. This sealing is like having their company's digital signature on that component to tell end-user that this component is secure enough in company's perspective. If user has trust in that company, he can use those components signed by them. We can divide the certification types in to three categories [3].

- Framework Certification
- Component Certification
- System Certification

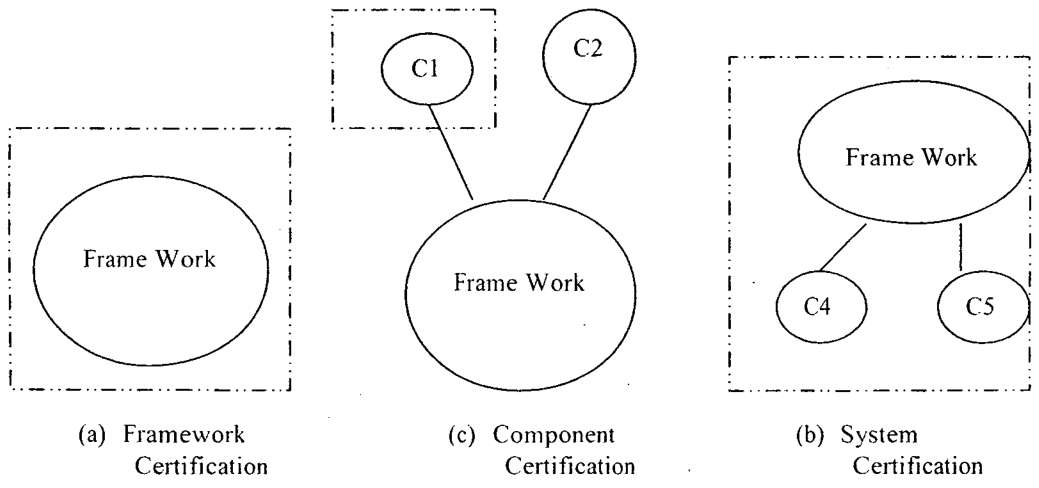


Fig. 1.6: Certification Types

Framework is like an operating system for the existing components [4]. It provides the underlying mechanism for the components so that they can communicate with each other. Examples of framework are .NET framework, EJB etc. For example, if we take the .NET framework, it contains three major components namely the common language runtime, class libraries and metadata. The common language runtime (CLR) is the execution engine for .NET framework-based applications. CLR is a Just in Time (JIT) compiler and MSIL

(Microsoft Intermediate Language) is the language of CLR. It provides security to .NET framework. Metadata is data relating to compiled code. It is stored with Microsoft intermediate language as a unit called assembly, where assembly is executable using one of the .NET framework's language compilers. Class libraries are reusable classes that provide functionality tasks such as user interface design, threading, network communications, authentication mechanisms and so on.

If there is any fault in that mechanism then the components assigned to that framework also works badly. So, framework's certification is necessary. Component certification deals with mainly internal functionality of component. This certification will deal whether the component is doing required task or not. These components and framework constitutes Component Based System. So, a certification is needed to know how the securities of individual components and frameworks are affecting whole system. In the above diagram dashed lines show for which stage the certification process is going on.

1.4 Motivation

Because of importance of certification, my work mainly focused on how to certify components through their functional and non-functional properties. This leads to studying different certification models and different certification processes. Although, there were different certification models, their main concentration is on certifying quality of the components. Quality itself includes security, availability, reliability, performance, efficiency, etc. Because of the importance of security in component based software engineering, it needs good focus. Though, there were also models on security, but few. So, my work is restricted to studying security aspects of component through their non-functional properties that means intra-component security certification.

An assessment scheme for certifying the whole system was given in [5]. If we look into that procedure, it is based on evaluation scheme which compares the system specific security requirements of the enclosing application with component specific security rating. An evaluation scheme is there which gives component security in terms of rank, based on this comparison. This method is based on CC (Common Criteria) evaluation. Analyzing component's security properties and comparing with system specific properties is the main objective of this method. But it has the drawback that it only tests the functionality of components as its main security measure. My proposed work is also similar to this method, which gives an algorithmic way of component certification. My model can be used as a component itself to work as an automatic component certifier. The main aspect of my model is to up gradation of a pre-issued certificate based on some properties. As COTS are composed dynamically, a separate mechanism is needed to dynamically update the existing certified components. This will save a lot of time which otherwise should be wasted by doing certification from the scrap.

1.5 Problem Specification

To develop an algorithm for giving a certificate to component or up-gradation of a pre-issued certificate by its non-functional security properties through that component version number.

1.6 Organization of work

The work is organized as follows:

In chapter 1, the basic concepts needed to understand the security aspects of component were given. Security properties like intra component security and inter component security are shown with example. Need for certification and certification types were also explained.

In chapter 2, a survey on different certification models is given. These certification models are divided into two parts and then they are sub-divided. Some mathematical models were explained along with reliability certification models.

In chapter 3, a security certification process is given through informal techniques. Some international standards in certification were briefly covered. This chapter provides the base for the proposed work.

In chapter 4, by using the informal techniques discussed in chapter 3, a brief overview of the proposed work was given. Algorithm with implementation details is also given by taking some examples.

In chapter 5, a brief overview on securities of different technologies likes JAVA, CORBA, .NET are given.

In chapter 6, Contributions and future work are given.

Chapter 2
CERTIFICATION MODELS

2.1 Classification of Certification Models

A certification model is a mathematical process of evaluating the component with respect to some conditions given by user or evaluator. Certification can be done at any time during development of component. Generally, it is done at third-party level to make sure no compromise with quality of end-product and to maintain trust. The main objective of the certification is to evaluate the quality of the component. As quality itself includes reliability, security, performance, etc, certification can be done at reliability, security, performance aspects of a component. If we look into the previous certification models we can divide these models into two categories [6].

- Mathematical Models
- Process-based Models

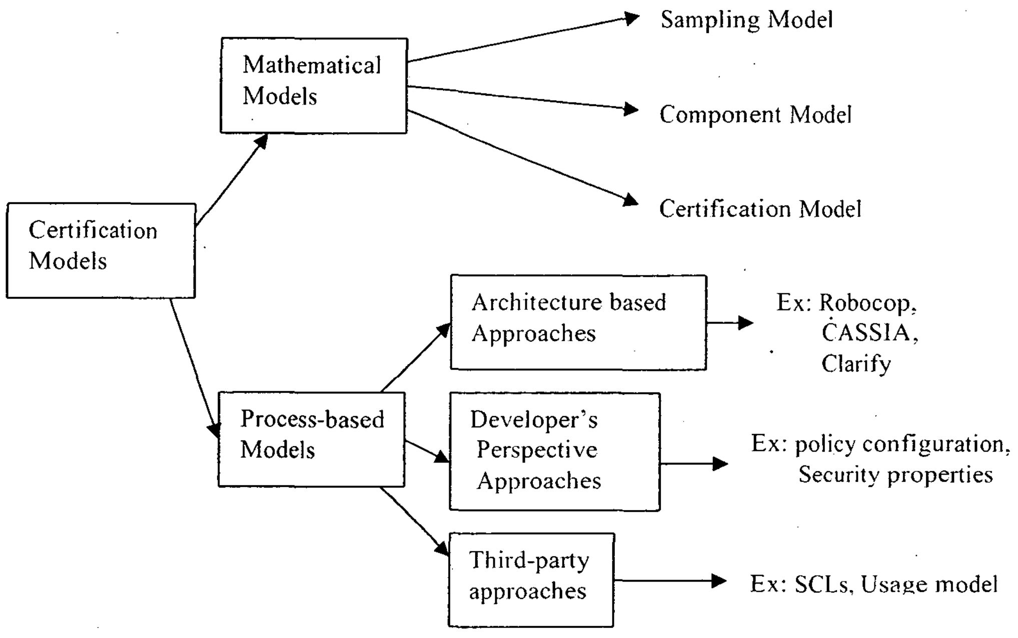


Fig 2.1: Classification of certification models.

2.2 Mathematical Models

Mathematical models take the help of mathematics to define metrics for evaluation of component. Three models were defined in this aspect based on which technique they will use to evaluate metrics of the candidate component. These models are based on the relationship of reliability and the mean time to failure [7]. We can define MTTF as the average number of uses between failures. Time is measured as the number of uses (or test cases) and the relation between reliability and MTTF is given by

$$\text{MTTF} = 1/(1-\text{reliability})$$

If time is represented by in any other way, the relationship is

$$\text{MTTF} = L/(1-\text{reliability})$$

These models were given for certifying reliability of a component. The main objective is that if a component has been verified by a mathematical proof of correctness, we may be able to attribute a high degree of reliability. The three models are given below [7].

- Sampling Model
- Component Model
- Certification Model

2.2.1 Sampling Model

This model is based on finding the number of test cases required to estimate the reliability percentage of the component. Through this, we can find number of test cases that should be run with out a failure. If 'C' is confidence in we want in experiment, then the equation is [7]

$$\text{Number of Test cases} = \lceil \log(1-c)/\log(r) \rceil$$

We can incorporate this model any one of these below methods.

(i) Zero-failures certification method

(ii) K- Failures method.

K-Failures method requires more test cases to show failures than of zero-failure certification method. It certifies software with known errors, if the errors are corrected; the certification is no longer valid because the test was conducted on the software before changes. Advantage of k-failures method will deny certification less often. The way the errors came in sampling model depends upon the certification method it uses. So, there are two ways of representing errors.

(i) It can certify the software that is, in fact, unreliable (zero-failures method).

(ii) It can deny certification to software that is, in fact, reliable (k-failures).

2.2.2 Component Model

The component model is useful for estimating how the individual reliabilities of components can estimate the whole system reliability [7]. This model can use existing component data to estimate system reliability with no additional testing. This model objective is to find out whether a component after being executed is able to transfer its control to some defined states. These defined states might be successful termination or to another component or unsuccessful termination. Consider a system composed of 'n' components, 1 to n, with component 1 is the single entry point to the system. Let r_i denote the probability that when component i is being executed, the system continues to another component without an error. Thus, $(1-r_i)$ is the probability of a failure during component i's execution.

2.2.3 Certification Model

This model assumed that MTTF grows exponentially over successive versions of a system . The certification model has three independent aspects [7].

1. The parametric form of AB^k, which is used to estimate the MTTF of version k
2. The corrected-log least-squares technique, which is used to compute A and B from the data points.
3. The technique for obtaining the data points.

Suppose $MTTF_K$ denotes the MTTF of version k of the system. For all k,

$MTTF_K = (B)(MTTF_{K-1})$ where B is a constant, then,

$MTTF_O = AB^k$ where $A=MTTF_0$

Taking logarithms on both sides, we get

$\text{Log}(MTTF_K) = \log A + k(\log B)$

$\text{Log}(MTTF_K) = a + kb$ where $a= \log A$, $b=\log B$

By solving above equation using standard linear regression, we can compute the estimates for a and b. Tentative estimates for A and B are e^{α} and e^{β}. The power of a model is the ratio of the released product's predicted MTTF to the number of tests. To estimate the power of the certification model, assume that A and B are exact and that data points lie exactly on the curve. Suppose that version n is released after versions 0 through n-1 have been tested. The estimated MTTF of version n is then AB^n ; the number of tests conducted is

$_{k=0}\sum^{n-1} AB^K = A(B^N-1)/(B-1)$

And the power ratio will be approximately (B-1). Thus if you need N tests under the certification model, you expect to do roughly 2N (B-1) tests to achieve the same level of MTTF certification under the sampling model.

2.3 Process-Based Models

These models based on particular process for certification of components rather than on mathematical models. In these, the certification can be done at various stages while developing a component. These stages can be design level, Development level, and End-product level or alternatively called as architecture-based approaches, developer's approaches, third-party approaches respectively. These models are less reliable than mathematical models. Therefore, the three categories of process-based approaches are

- Architecture-Based approaches
- Developer's perspective approaches
- Third-party approaches

2.3.1 Architecture-Based approaches

These models mainly based on the architecture of the whole system to certify component. In these models the factors which effect the certification will be only the system where the component is going to fit, that means the design of the system. Examples of these design-based certification models are Robocop model, CASSIA model and CLARIFI model. A brief description of two models was given in the below section.

2.3.1.1 Robocop Model

This model was given [8]. This model is based on Robocop component architecture and well suited to black-box components. It is proposed as middle-ware layer architecture of high volume embedded appliances such as PDAs, mobile phones, set top boxes and DVDs. In Robocop different aspects of a component are called models. Examples of these models are functional specification, the executable code, a resource model, documentation, source code and so on. A Robocop component is a set of such models and relations among them.

A Robocop component contains implicit claims [8]. A claim is expressed in terms of elements of component. The elements are models, relation and details about their representation. Certification in this type is of two step procedure. First one is the verification of a given set of claims (claim model) about a component resulting in a verification model. Second step is the subsequent sealing of the component and these two models. The verification step verifies the claims and results in a set of verification reports, which is a model. The sealing step produce a seal based on the component with the claims and the reports included, which is again a model.

2.3.1.2 CASSIA Model

This model was given in [9]. This model certifies scalability and performance of a component if the whole system is made of fortresses. Component Adaptive Scalable Secure Infrastructure Architecture is abbreviated as CASSIA model. The security precautions which are incorporated in to system may affect performance and scalability of the system badly. To remove that, this model was proposed. It also uses on protocol called SCOP (Secure Component Protocol). The components according to this model are built as fortresses. "*Software Fortress architecture is an enterprise architecture consisting of a series of self-contained, mutually spacious, marginally co-operating software fortresses interacting through carefully crafted and managed treaty relationships.*" If a component enters in a fortress, it gains access to all other components within that fortress. The walls of fortress are collectively called a security perimeter of the system. Treaties are formal agreements within the fortress. For example, consider a fortress has two components c1, c2 and one more fortress has a component c3. If c3 wants to communicate with c2 then both these fortresses enters in to a treaty so that communications of these components can cross the walls of these fortresses. Then CASSIA makes a decision to encrypt these messages between components and for that it uses SCOP. This model also proves that close to 80% of interactions between components and their clients in different commercial systems occur within protected boundaries, means with in fortresses.

2.3.2 Developer's perspective approaches

These approaches are based on developer's view to certify a component. This means there is interaction of developer while certifying a component. So, these models mainly based on some properties of component which a developer can only specify clearly and abstractly. Here we will discuss two approaches, policy configuration model and assessing security properties model.

2.3.2.1 Policy configuration Model

This model was proposed in [10]. A policy configuration model is to represent security policies in a format which can manifest conflicting properties across policy specification. Security certification criteria governing the integrated system can introduce conflicts with local component policies. Security policies and certification criteria lack a common representation. This model defines security policies according to fundamental attributes of property assertions, observable behaviors, mechanisms, constraints, communication and interaction expectations, dependencies on other policies, system configuration and component state. The common security concepts are trust, certificate, certificate sharing, certificate delegation, encryption, secure channel. Certificate sharing is useful when one component tries to allocate another component to do work on behalf of first one. For example, two components A and B can share a certificate then B can do work of A on behalf of A without any problem, but the problem comes whenever another component comes as proxy to A. Security policies addresses different overall concepts including goals, problems, systems, critical usage and domain of the application. Six common policy types on which this model was founded were authentication, authorization, data protection, audit, availability and non-repudiation.

2.3.2.2 Security Properties assessment Model

This model was proposed in [5]. This model is for whole system certification with the help of developer. If we look into that procedure, it is based on evaluation scheme which compares the system specific security requirements of the enclosing application with component specific security rating. An evaluation scheme is there which gives component security in terms of rank, based on this comparison. This method is based on CC (Common Criteria) evaluation. Analyzing component's security properties and comparing with system specific properties is the main objective of this method. This model treats a component as atomic one and this is not suitable for compositional components. It is also an engineer's perspective model since it is not possible to clearly distinguish security properties and their specific evaluation for a user, only a developer knows for what aspects he designed that component.

2.3.3 Third-Party approaches

These approaches were applicable after completion of component. After building the component a third-party certifies whether that component is maintaining its specified requirements or not. These models can be applicable to either white-box or black-box components. Here we will discuss about usage model and software component libraries.

2.3.3.1 Usage model

This model was given in [11]. This is for certifying reliability of component. This model certifies a component with usage testing by developing usage models, applying usage profiles and then applying a hypothesis certification model. During development for reuse, a usage model must be constructed in parallel with the development of the component. The usage model is a structural model of the external view of the component. The probabilities of different events are added to the model, creating a usage profile, which describes the actual probabilities of the events. The component is stored together with its characteristics, usage model and usage profile. The reliability measure stored should be connected to the usage profile, since another profile will probably give a different perceived reliability of the component altogether.

2.3.3.2 Software Component Laboratories

This model was given in [12]. This model aim is to have an independent stand with the development of a component to maintain a complete trust with consumers. These

SCLs would accept software from developers, gather information from user sites, use data gathered from several sites to generate statistics on use and performance in the field and provide limited warranties for the software based on these statistics. This model is based on test certificates that developers supply in a prescribed form so that purchasers can, in short order, determines the quality and suitability of purchased software. The test cases are designed in extensible markup language. XML is a widely adopted general-purpose markup language for representing hierarchical data items.

Along these models, there are other models like Bell la padula model, but here we looked at only some models. Though, we can clearly have a lot of certification models, the models based on security are not much. As security is an important aspect, there is a need of great focus in that area.

Chapter 3
SECURITY CERTIFICATION PROCESS

3.1 Standards

In this chapter we particularly deal with certification of a component with respect to it's security characteristics. There are some standards which specify the way of evaluating a component through some specific security objectives and provide security in terms of levels. Generally accepted standards are FIPS 140-2 for cryptographic models, ITSEC (CEC 91), TCSEC [DOD 85], Common Criteria (CC), NIST etc. Out of these models, CC, ITSEC, TCSEC are more popular and we will discuss about these three models briefly in below sections. My proposed model which will be discussed in the next chapter is based upon CC (Common Criteria) standard.

3.1.1 TCSEC (DoD 85)

This standard was developed by United States Government Department of Defense in 1985 for assessing security controls built inside computer. This standard was also referred as Orange book. It defines four divisions D, B, C, A where A has highest security. These division levels represent the differences between trusts levels desired in the target. These divisions can also have sub-divisions like C1, C2, C3, B1, B2, and A1 [13].

3.1.2 ITSEC (CEC 91)

A security standard was published by Commission of European Communities (CEC) in 1991 by the name ITSEC (Information Technology Security Evaluation Criteria) [14]. This standard is having a set of criteria useful for evaluating security of products or systems. The evaluated product is given as TOE (Target Of Evaluation) to this standard and the degree of evaluation depends upon the level of confidence desired in the target. For this, there are different levels of confidence, defined as Evaluation Levels (EL) in ITSEC. They are denoted as E0, E1 ... E6. The level of confidence increases as the level increases and hence higher levels require more extensive testing of the target. A given target's security features were documented in Security Target whose contents must be evaluated and approved before the target itself was evaluated

3.1.3 Common Criteria (CC)

It is an ISO/IEC standard for security. It provides a frame work in which users can specify their security requirements, developers can claim about security attributes of their products and third-parties or testing laboratories can evaluate the products to check their validity with respect to requirements [15]. The main concepts of CC are TOE- a product or system that is subject to evaluation, PP-protection profile, a document created by user and serve as template to security target, SFR-security functional requirements specify individual security functions of products, ST-Security Target, to identify the security properties of evaluated product, EAL-evaluation assurance level, which gives a numerical rating to target. These EALs are divided into 7 levels EAL1, EAL2.....EAL7.

3.2 Difficulties

Certification of component is a must in today's world where the same component can be downloaded through out the world and it's a herculean task to do. A lot of things need to be verified while certifying a component. Issues like architectural and design styles of component, performance of the component, security issues related to that component, efficiency of the component, reliability etc. needs to have a deeper look while giving certification. But, there are no agreed-upon standards for measuring component's quality. So, there is confusion while measuring which characteristics of a component should be highlighted and which to neglect. Applying certification standards to dynamically changing components is also a difficult thing. So, briefly we can say the below three are drawbacks of certification.

- No common set of standards for measuring a component's quality.
- Lack of experience in applying certification standards to existing component.
- No standard mechanism to derive metrics of certification

3.3 Informal Techniques

Informal techniques are those techniques which do not have strong mathematical base. They rely heavily on human reasoning and subjectivity. In software engineering, the general informal techniques are inspections, reviews, and walkthroughs. In security certification of components, these techniques have wider foot hold than formal techniques. Testing plays a vital role in these techniques. In the next subsequent sections we will discuss the types of testing and their importance in these techniques.

3.3.1 Role of Testing

Generally, certification of a component includes some test cases to evaluate required properties of that candidate component. These test cases depend on what area the certification is going on. It might be reliability certification or security certification or efficiency certification or on a whole system certification. But certification entirely based on testing is meaning less as in adaptive software like aircraft controller which "learn" after deployment. These systems behave differently after being run on field. The system tested in the lab is not a true reflection to the system after deployment. So, some more relevant techniques combined with testing are needed.

Certification of a component should be done at two levels. One is at component level and other at system level. System level certification can be obtained by evaluating composed component certificates. Based on whether a component is white box or black box, the testing techniques used can also be white or black box [16].

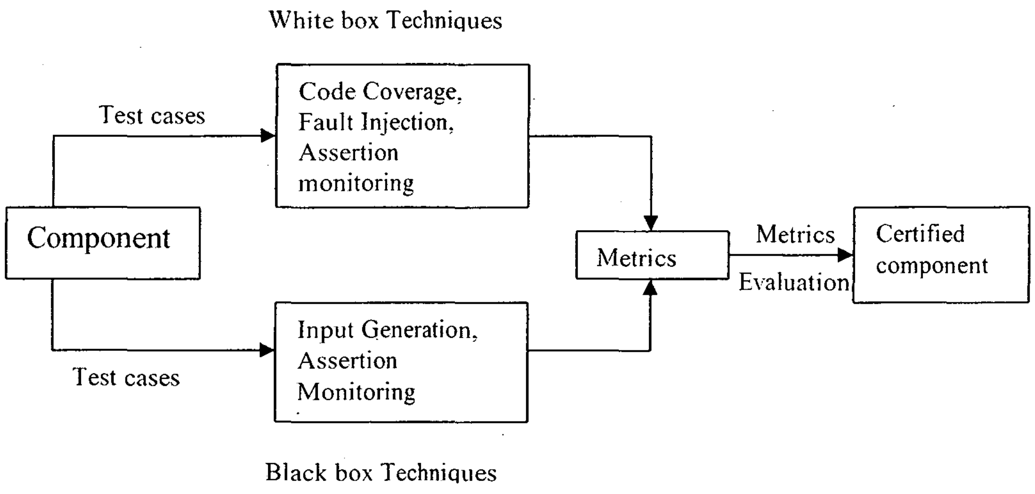


Fig 3.1: Component Certification Model

The above diagram represents a general certification model of a component. Test cases and testing techniques depend on the context. If the source code of candidate component is available, then we go for white box techniques otherwise black box techniques. These techniques will yield some metrics. By evaluating these metrics, the third party organization will give certification.

3.3.2 White box techniques

There are three types of white box techniques [16].

- Code Coverage
- Fault injection
- Assertion Monitoring

Code coverage analysis provides a measure by observing the codes of component which are not executed or not reached while testing. If this analysis identifies a section of code that has not been executed, for example it can be a function that was not called or a loop that was not followed, then that particular code is certified by including more test cases to the existing ones. The more that is tested, the higher confidence the analyst will have on the result of the certificate.

Fault injection technique attempts to corrupt inputs and outputs to software components. It tries to make the system fail in new and unique ways. Code coverage analysis is most useful when combined with security-oriented testing such as fault injection analysis and property-based testing. Both system inputs and component inputs can be corrupted through this technique. Property based testing is the process of analyzing the behaviour of a program to determine if it adheres or violates some property.

Observing security violations either through fault injection analysis or through property-based testing is made possible through the use of assertions. Assertions are conditional statements placed in program that satisfy security policy of the program. Combining assertion with fault injection analysis and coverage analysis, property based testing of software component can be performed. This should be done until a degree of confidence is reached. The degree of confidence necessary will be determined be the application in which the component will be deployed.

3.3.3 Black box techniques

The use of security assertions provides the ability to determine if a security policy violation has occurred as the result of input sampled from the expected user distribution [16]. The input stream can be generated as samples from the different input spaces of component .Most samples get input from the expected user input distribution or operational profile. If this is unknown, the input generation functions provided will support sampling from a wide range of potential user profiles. In cases where user profile data has been collected, the input generation module will support sampling from the customized user profile. Generally, the most dangerous vulnerabilities will be detected by sampling expected input distributions.

The test cases used for black box testing can be enhanced with malicious input using perturbation functions. These functions include the ability to truncate input streams, to overflow input buffers, to append garbage input and malicious commands, to append garbage input and malicious commands, to perturb numerical constants and to garble strings.

3.4 Formal Techniques

Formal techniques are those which can be expressed in restricted syntax language with defined semantics based on well-established mathematical concepts. In security certification of components these techniques were not as established as informal techniques. But, there is a need to focus on this way as these techniques are more valid than informal techniques.

Chapter 4

PROPOSED SECURITY CERTIFICATION PROCESS

4.1 Overview of Process

In this chapter we particularly deal with our proposed security certification process for a component. An algorithm regarding this will be given in next section. This model is mainly for intra component security in which only atomic components were considered. For simplicity reasons, atomic components were considered rather than hierarchical components or compositional components. In this section we will discuss about general overview of our process.

The main process is a goal-oriented approach. In this approach we divide every security requirement into a specific goal. There may be many sub requirements in a single security requirement. So, every security requirement is treated as a specific goal and every sub-security requirement is treated as a sub-goal. Every goal is evaluated by evaluating its sub-goals and in the end a security percentage as a out put will be given to show how much the component is competent as per the user specifications. The main approach can be shown in algorithmic way as below.

Main steps:

1) Identifying Security Requirements as Security Goals.
2) Dividing Security Goals into Sub-Goals using Security Profiles.
3) Evaluation of all Sub-goals.
4) Evaluation of main goal using those sub-goals.

5) Evaluation of whole component using all Security Goals.

4.2 Algorithm

Algorithm: **Certification Driver**

User Report	:	Required Security Goals (SG1, SG2, SG3…….SGn)
Input	:	A Component
Input Structure	:	Component specified with security goals from User Report and Version Number specified in X.Y format.
Output	:	Security evaluated Component
Output Structure	:	Component with security Percentage
Requirements	:	Test Case Database, Security Template.
Component Function	:	COMPONENT (name, version number, SG [], SL [])
Certification Driver	:	CERTDRIVER (component name)

COMPONENT having loaded with input parameters with the help of User-Report and Version number will be given as input to CERTDRIVER.

```
Step 1: CERTDRIVER reads Version Number which is in X.Y format, and reads
        the value of 'Y' and stores security goals in SG [n].
Step 2: If Y is zero, then go to step 5.
Step 3: From Read SG [0] to SG [n-1]
          3.1: call SECURITYTEMPLATE (SG [i])
          3.2: Identify the sub goals based on pre defined Security Template structure
          3.3: Read sub goals into SUB []
          3.4: From SUB [0] to SUB [n]
                3.4.1: call TESTCASEGEN (SUB [i])
                3.4.2: Call Test cases in Descending order Based on rank
                3.4.3: If there is failure of a sub-goal, increment rank of that test case.
                3.4.4: Otherwise return the percentage value tagged to that sub-goal
                And store it in SUB [i]
            3.4.5: Go to Step 3.4
        3.5: Sum up all the percentages in SUB []
        3.6: Return that percentage value to SG [i]
Step 4: Go to step 3
Step 5: Call SG [], and if available sum up all values in SG [] otherwise continue.
Step 6: Attribute this percentage to COMPONENT and put 'Y'=0 in Version.
Step 7: Check if there is any other component ready for certification
Step 8: If ready go to step1 otherwise Exit CERTDRIVER.
```

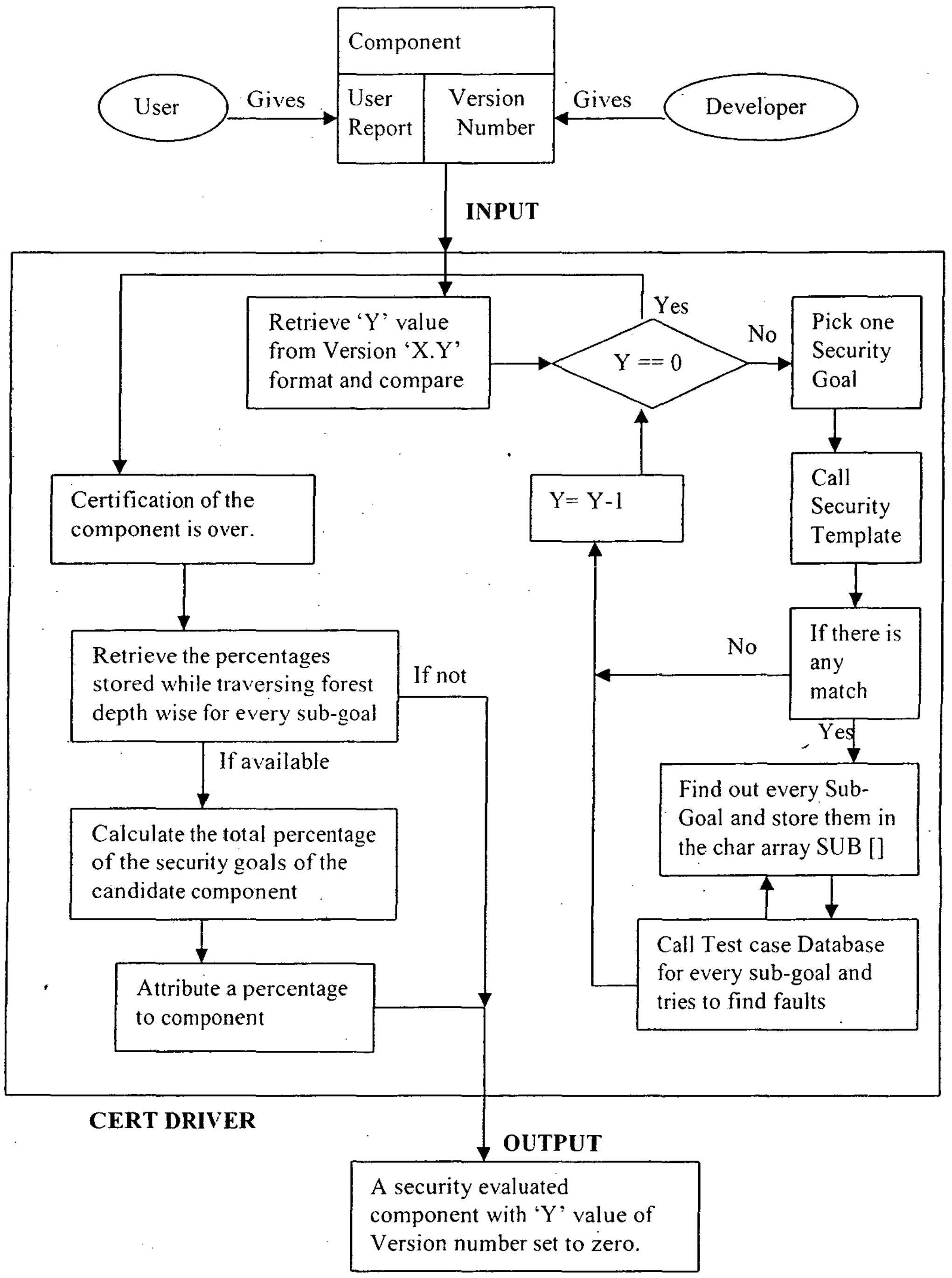
4.3 Implementation Details

The main principle concepts of this algorithm are User Report, Security Goals, Test Case Database, Version Number and Security Template. Security properties are inherently represented as security goals. These security goals are again represented as security sub-goals. The basic structure of CERTDRIVER will be shown in next page.

User Report generally given by user containing required security goals or its sub-goals directly from user's perspective. This is not a mandatory input as many times a user may not able to mention his required security aspects relating to particular domain. So, if a user having capability and want to particularly mention his security specifications then our model allow him in that regard. User report structure will be a set containing component's name and its required security goals.

Security Goals are nothing but the properties of that component represented in informal language. Users should specify them clearly to match their required security features. Ambiguity in specifying the security goals may lead a different direction of validation. If there is any ambiguity, then the CERTDRIVER tries to match this with the closest property mentioned in Security Template. If this is also not possible, then it again asks user to mention it properly.

User can have a look at the manual of CERTDRIVER before get used to the way of mentioning the security goals.



Component
User
Gives
User Report
Version Number
Gives
Developer
INPUT
Retrieve 'Y' value from Version 'X.Y' format and compare
Yes
Y == 0
No
Pick one Security Goal
Call Security Template
Certification of the component is over.
Y= Y-1
If there is any match
No
Yes
Retrieve the percentages stored while traversing forest depth wise for every sub-goal
If not
If available
Find out every Sub-Goal and store them in the char array SUB []
Calculate the total percentage of the security goals of the candidate component
Call Test case Database for every sub-goal and tries to find faults
Attribute a percentage to component
CERT DRIVER
OUTPUT
A security evaluated component with 'Y' value of Version number set to zero.

Fig 4.1: Flow chart of CERTDRIVER

Test case Database is a database consisting of pre-constructed test cases relating to different domains and different security aspects. These test cases can be represented in Extensible Markup Language (XML) since it is in general a widely accepted standard for mentioning hierarchical test cases. Our Test cases are hierarchical because they have to test security goals, as security goals are hierarchically arranged with security sub-goals, so the test cases be. By using XML, we can represent test cases in tree form like those of Security Template and Test cases can be retrieved based upon the required security sub-goal. These test cases in every hierarchy have one special property, rank of test case. Rank will be given to those test cases which are proved to be helpful in finding faults. Rank will be increased by one every time, when it results in failure of the component.

Version Number is a pre-defined syntax to know whether a component has been changed or not after giving at least one time certification. This concept plays a pivotal role in upgrading the certification. We follow some syntax while defining version of a particular component. Let us consider "Version X.Y" is a notation of a component, then 'X' denotes the number of security properties (Intra component security properties or NFS, as we particularly dealing with these) that component have and 'Y' denotes which properties of the components are going to effect after a change in the code of component. Here 'X' is an integer and 'Y' is a string of integers. 'Y' can be of [{a, c};f] where {a, c} means the properties from a, b and c are to be checked by third-party certifier and 'f' properties are also to be checked. To get a clear understanding, let us consider the following example notation, Version 7.[{1,4};7] denotes that there are 7 non-functional security properties in which the properties set {1,4}means 1, 2, 3, 4 properties should be checked because of a change in the code. One more property is also to be checked that is of 7th property. But for this implementation we need to have an order among security properties as we are mentioning them as numbers in version code and matching should be perfect. For this we can implement an alphabetic technique for mentioning these security properties or developer should issue a report specifying how he mentioned security properties.

The conditions that can be retrieved in Version Number through X and Y are as follows. Here, 'X' denotes the number of security goals prescribed in Version Number and 'Y' denotes the number of updated security goals which needs again certification. The pre-condition on 'X' was that it can not be zero at any time, only 'Y' can vary.

1. Given X, Y!=0 and X=Y, then the component is a new component
2. Given X! =0, Y=0 and X=Y, then that component is already certified one.
3. Given X, Y! =0 and X>Y, then some security goals needs to be re-certified.
4. Given X, Y! =0 and X<Y, not possible condition.

Security Template is an inbuilt pre-defined structure to match different security goals. This template main aim is to divide each security in to different security sub-goals. Goals and sub-goals are arranged in tree order, we can define it as forest data structure, where main goals is the root of forest, and the immediate sub-goals are arranged as immediate descendants in the forest. As there may be a chance that sub-goals can be further divided into sub-goals, so they can be arranged as next descendants of that sub-goal. Security goals can be arranged as a forest data structure as shown below.

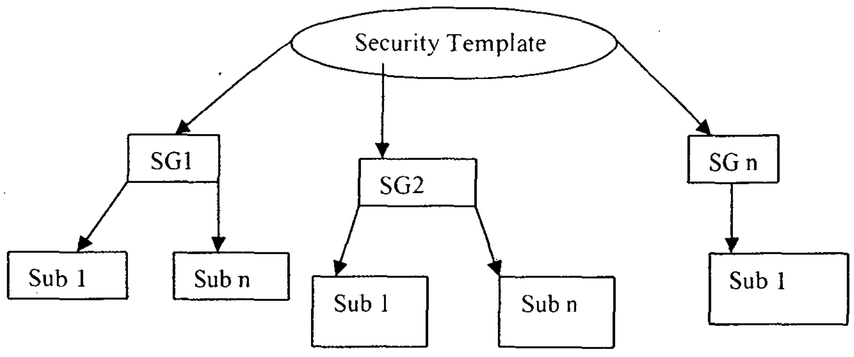


Fig 4.2: Structure of Security Template

In the above figure, we make a forest containing security goals as children. We choose forest as there is no limit on the number of security properties, a component can have. Each Security Goal is as important as another, so we will assign an equal value of percentage to each security goal. Suppose a component is having five security properties, then we will assign 20% importance to each property. That is if that component fails to satisfy two security properties then it is 60% (3* 20) secure to use. This structure will not suitable to specify priority among security properties. As security goals are not atomic, the sub goals are to be given equal weightage. So, if you consider the above example, every sub-goal of one security goal should be managed with in 20% only. Suppose if there are five sub-goals, then each will get 4%, only if three were satisfied, then whole security goal will be only 12%(3*4) instead of 20%. Like this all values are propagated from leaves of forest to the root to resulting in total evaluation of whole component. At the end, we will get some percentage values for each security goal. By summing these values we will get the total security position of component.

Let 'S' be the total security evaluation of component.

S = SG 1+SG 2+.........+SG n

= (Sub1+ Sub2+.........+Sub n) + (Sub1+ Sub2+.........+Sub n) +........+SG n

4.4 Benefits of this Model

The main benefit of this model is the facility to upgrade the certification. Up gradation in certification was done with the help of Version number and following some syntax while specifying that number by developers. The main theme is to divide every security requirement into some property, particularly Non-functional security properties in this model. As CERTDRIVER only checks for changed security properties, it saves a lot of time particularly in large–scale component based systems where each component is very large and it takes a lot of time to check that component through test cases, as there is a need of large number of test cases.

Giving rank to test cases is also an important aspect of this algorithm. This is also aimed at reducing time to find faults and having an effective testing. We increase the rank of a test case if that test case succeeds in finding a fault. Storing its domain of work is also a good move. We arrange these test cases in the decreasing order of their rank and first we will test with those cases later we will test with remaining. Ordering test cases in the tree order as of Security Template allows us to check the security goal needed to test quickly and by that produce the test cases for that. This is because we divide and organize test cases according to security properties, thus for retrieving them if we search them by security properties it will yields results more quickly than if we search sequentially. The out put of this model is in very simple metric which can be understood easily by a common user.

4.5 Case Study

We discussed in previous sections how to do up-gradation of certificate through component's version number. To get a clear understanding of the algorithm, take an example, a component by name Event History and apply this algorithm. As our model is based on Common Criteria, this example is also based on some specifications given in Common Criteria.

The component Event History, tries to store all the events generated by any user, whether he is administrator or guest. This component can be useful in many domains. The same component can be used in banking systems to retrieve the previous illegal operations of any user or it can be used in inter-networking of systems where any user tries to hack other computers. The main objective of this component is to secure other components through information security. Event History can be given with user report along its Version Number to CERTDRIVER.

The User Report should contain the required security goals of Event History. Here Version Number can be given depending on the state of the component. If that component is a new one, then it's version number should be represented as Version 3.3, assuming that there are three intra-security properties. Suppose it is already certified once and changes made to that component effect only some security goals of that component, then these can be represented as Version 3.2, if only two security goals should be again certified. So, the security goals can be represented as below.

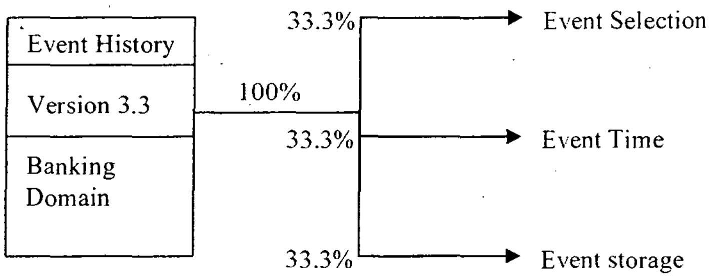


Fig 4.3: Finding Security Goals of Event History

The security template shown in the above diagram is stored in a database and it will be connected to CERTDRIVER. When CERTDRIVER calls the component Event History, the Event History send its required goals as Security Goals and Version Number in X.Y format. Here, the version was given as 3.3 which mean that either it's a new component or a component modified such that all its security goals might be affected. By reading one security goal at a time, CERTDRIVER stores them in SG []. For these every three goals, CERTDRIVER calls Security Template, to find out its sub-goals.

The division into sub-goals mainly depends on the process which these security goals would follow to achieve required level of security. First, consider the Event Selection security goal, the main objective is to selecting events individually, that means atomicity should be maintained while selecting events. In the same way, the Event Time goal can also divided into techniques used to get correct measuring of time at any situation, the Event storage goal can be divided into the efficient memory techniques for maintaining a list of operations what had happened and what effects they got on other events. These storage details can be maintained by some check points.

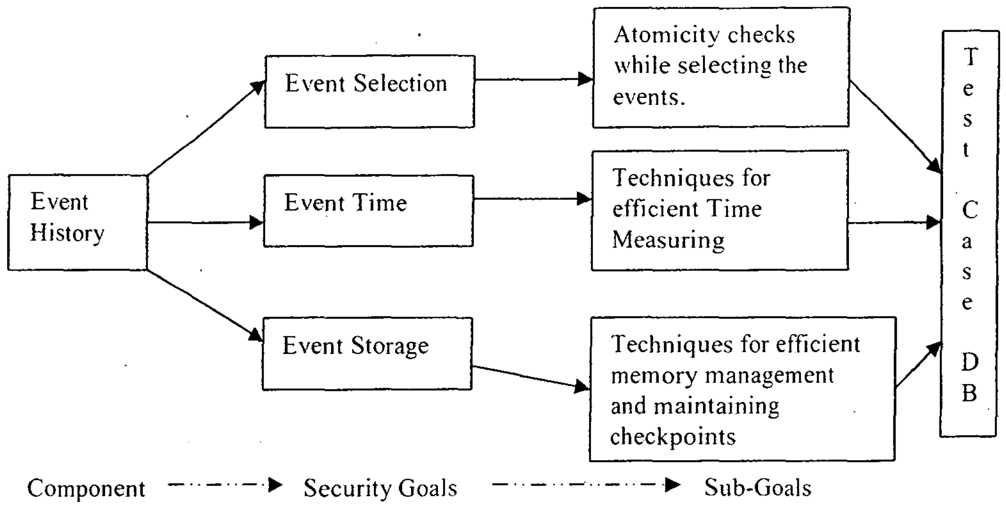

Event History
Event Selection
Event Time
Event Storage
Atomicity checks while selecting the events.
Techniques for efficient Time Measuring
Techniques for efficient memory management and maintaining checkpoints
Test Case DB
Component
Security Goals
Sub-Goals

Fig 4.4: Security Template for Event History

In the above figure, we observed that all the security sub-goals are connected to a Test Case Data Base. This is maintained as a hierarchical data base in which test cases are arranged in descending order of their rank. For maintaining hierarchical test data base we use XML. The hierarchy in test cases is dependent on the domain of the application and its sub-goals required. In our example, the first hierarchy will be domain and name of the component, second will be security goal of the component and third will be sub-goals of the component. Thus, by specifying in this order it will be very easy for retrieving test cases whenever required. The test cases depend on the nature of the component. For white-box components, code-coverage technique will be good and for black-box components, fault-injection technique test cases will yield good result.

Like the same way we did distribute percentages to different security goals, we will distribute percentages to each and every sub-goal. At the end, all these percentages which got success while testing are summed up and produced as whole component's security percentage.

Chapter 5
SECURITY IN TECHNOLOGIES

5.1 Security in CORBA

The security in CORBA deals with the following central four elements [4, 17].

- Security Policies
- Principal Authentication
- Access control
- Security Association

5.1.1 Security Policies

These policies aim at differentiating security administration to application development with the help of rules. These rules will be defined by security policies for authentication, secure invocation, privilege delegation, access control etc. There will be different security policies for different security domains. When a target is created, ORB will check which domain that belongs and adopts those policies. There may be sub-domains in a single domain, so, security policies are also designed specifically.

5.1.2 Principal Authentication

It generates credentials which carries information about the security attributes of a principal. Attributes can be identities and privileges of the principal or both. As these attributes are present in credentials, which can be expressed as system's policies, so, these attributes form the basis of system's policies. Which attributes are to be given will depend on principal authentication policy. Based on the provided information, there will be some underlying authentication mechanism which decides which privileges to put into a principal's credential object. These security attribute can be present at both client and target side.

5.1.3 Access Control

This feature will be normally at target side. This access control supports and controls requests coming in and out to the target side. In CORBA, there are standard sensitivity levels for each operations like g(get), s(set), u(use), m(manage). In this model, the policy compares the level required to access the operation with the level generated to the client. Access is allowed only if there is a correct matching of levels. This model can support several different policies, such as, access control lists, capability lists and role-based access control.

5.1.4 Security Association

This is mainly associating several credentials to several security contexts. These can be transferred to other networks i.e. to other remote CORBA security system, if communications among network are involved. These credentials provide necessary security information at both client and target sides. Generally, CORBA security services use standard network authentication mechanisms. This network authentication mechanism fails when both client and target reside on the same ORB, as the network does not involved in the communications, the mechanism will be ineffective.

5.2 Security in JAVA

Security in java provided through a model called sand-box model. This model keeps limitations on the down-loaded code. According to this model, local code will be given full access to system resources where as a downloaded code can have only access to limited resources. Overall security can be enforced by some mechanisms like type-safe and easy to use. Compilers and a byte-code verifier make sure that only java byte codes are executed. At run-time language safety will be guarantied by byte-code verifier and java virtual machine. A class loader defines a local name space. Security manager class restricts access to critical system resources. We can execute downloaded programs either in sand-box model or in JAR format. JAR format is applicable for signed applets only.

5.2.1 Security in JAVA2

The security architecture was introduced for the following purposes.

- Fine-grained access control.
- Easily configurable security policy.
- Extensible access control structure.
- Extension of security checks to not only remote code but also local code.

5.2.1.1 Protection domain

Protection can be done in two ways.

- System protection domain.
- Application protection domain.

All external resources such as the file system, the networking facility etc are belong to system domain. Applications like printing a message comes under second domain. Generally, a domain consists of a set of classes whose objects have same set of permissions. In Java, there will be mapping from code (classes and objects) to their protection domain. It may also possible that both application domain and system domain may involve in the same execution thread. Here, in this scenario, application domain will be restricted from gaining more permission by calling system domain. Suppose, if system domain calls application domain by invoking some method in application domain, the permissions of system domain will be maintained same to current enabled application domain. The rule used for calculating permissions is the following [4].

- The permission set of an execution thread is considered to be the intersection of the permissions of all protection domains traversed by the thread.

- When code calls the Do Privileged method, the permission thread of execution thread is considered to include permission if it is allowed by the code's protection domain and by all protection domains that are called or entered afterwards.

During execution, when access to a critical system resource is requested, the resource-handling code invokes a special *AccessController* class method that evaluates the request and decides if the request should be granted or denied. Such an evaluation follows the rule given above. The basic principle is to examine the call history and the permissions granted to the relevant protection domains, and to return silently if the request is granted or throw a security exception if the request is denied. Finally, each domain (system or application) may also implement additional protection of its internal resources within its own domain boundary.

5.2.1.2 Dynamic Class Loading

Dynamic class loading provides the java platform with the ability to install software components at run-time [4]. It has a number of important characteristics. First of all, classes are loaded on demand and at the last moment possible. Second, dynamic class loading, maintains the type safety of the java virtual machine by adding link-time checks, which replace certain run-time checks and are performed only once. Moreover, programmers can define their own class loaders that, for example, specify the remote location from which certain classes are loaded, or assign appropriate security attributes to them. Finally, class loaders can be used to provide separate name spaces for various software components. One can load mobile code from different servers using separate class loaders, thus maintaining a degree of isolation between those classes. This mobile code can contain classes of the same name, which are then treated as distinct types by the java virtual machine.

5.3 Security in .NET

The security features of framework can be classified as [4]

- Evidence-Based security
- Code access security
- Role-Based security
- Application domains
- The verification process

5.3.1 Evidence-Based security

Key elements of this security sub-system are policy, permissions and evidence. .Net framework security features are based on XML described security policy. This defines resource allocation by matching permissions to evidence. This can be a default security policy for a safe execution environment or customized by developers to address application specific security needs. Permissions maintain resources and associated rights implementation methods for demanding and asserting access. There are three types of permissions. Minimal, Optional, Refuse. The minimal criteria for granting an assembly, if the policy doesn't grant it, then the assembly will fail to load it. Using 'Refuse' request, developers can decline access to resources. By evaluating assembly's evidence, CLR determines which permissions can be assigned at run time. Sources of evidence include cryptographically sealed namespaces (strong names), software publisher identity (Authenticode), Code origin (URL, Site, internet explorer zone).

5.3.2 Code Access Security

This feature limits the assembly code not to exceed its permissions while executing. Generally, code assemblies are mapped with their corresponding set of permissions when they are loaded. An appropriate permission object is therefore needed whenever a method in an assembly needs permission to access a resource. A stack-walk is initiated to check whether each and every assembly in the chain has the required permission otherwise an exemption will be generated causing the operation to halt. By this stack-walking feature, we can stop untrustworthy code attempts. By two mechanisms, developers used to control these permissions. They are imperative and declarative. Imperative checks request a demand or override portions of the stack-walk operation. These occur at run-time. Declarative are also same but expressed as attributes that are evaluated at compile time. These checks occur at JIT also. Imperative checks limited to the place where they were written, where as declarative checks can cover whole class i.e. to every method, constructor and property in the class.

5.3.3 Role-based Security

Authentication and authorization will be given by Principal and Identity concepts. Principal represents context of running code whereas identity represents that particular user at that context. The running code queries the principal about identity roles, allowing or denying permissions according to role-membership.

5.3.4 Application Domains

Operating systems isolate application domains by running each application in different process with different address space. It is not good in terms of performance for loaded servers. Running different processes for each other is not possible. The type-safety of the code which restricts the code not to access arbitrary address provides a good isolation at process boundary. Code running in one domain can not directly effect other applications. All code is loaded into a single application domain and run according to that domain security policy.

5.3.5 The Verification Process

We already discussed that CLR provides memory safety which eliminates the unexpected actions during code execution. Verification process is the last step in ensuring the runtime safety of managed code. It prevents common errors from occurring such as buffer overflows, invalid references in the stack, using integer pointer to access memory locations. The unmanaged code that runs outside the CLR is outside of security checking and thus may cause unauthorized access to resources. Managed code wrappers will take care of verifying the permissions and parameters for unmanaged code.

Chapter 6
CONCLUSIONS

6.1 Contributions

- My work focused mainly on intra-component security, which was not given much focus in previous security models though the quality properties are given as a whole while giving certification.
- So, it leads to a specific non-functional security certification driver.
- There was no specific algorithm for non-functional security certification. My work focused mainly on algorithmic way giving a step towards implementation.
- Up-Gradation of certificate is an important aspect of this algorithm. By this feature, we can save a lot of time especially in large scale component-based systems where a component having large code needed to certify unnecessarily with out any change in the source code.
- By using Version numbers not only we can upgrade the certification and also we can eliminate redundant certification.
- Giving a rank to test cases is a good addition in algorithm to make the process of choosing test cases more effective. As rank will be given only to those test cases which can result a failure of component, choosing those cases in the beginning will certainly improve the speed of the whole process in large-scale Component- based applications.
- As it also considers previous test cases, we can say it as a self-learning certification driver as it learns from previous failures to make existing process more speed effective.
- We can also incorporate it as a component in component based applications, so that it can work automatically whenever there is a need, resulting in Automatic self-learning certification driver.

6.2 Limitations of the Proposed Model

- This model only deals with Non-functional security properties of the components represented as security goals.
- As Version number were specified by developer, if there is any mistake in representing it can damage whole certification process.
- Up-Gradation of certification totally depends on Version Number, which will be given by developer. This means that to get up-gradation we should depend on developer. This is against the basic concept as the user might not trust the developer.

6.3 Future Work

As our algorithm does not deal with security function properties and keeping the importance of inter-component security in mind in component based software engineering, implementing them will be a good step. Making whole certification process independent of developer will make user to have more trust on certificate. So, the further work be developing a developer-independent certification driver having able to do up-gradation in both intra and inter-component security areas.

REFERENCES

[1] Khaled Khan, Jun Han, and Yuliang Zheng Security Properties of Software Components: ISW'99, LNCS 1729, pp. 52-56, 1999.

[2] Jun Han and Yuliang Zheng, Security Characterisation and Integrity Assurance for Component-Based Software, pp. 61-66, 2000 IEEE.

[3] A material on certification by CMU/SEI-2000-TR-008.

[4] Marteen Rits, Karima Boudaoud, Component Adaptability and Security, a project internship submitted on June 29, 2003.

[5] Khaled M Khan, Jun Han, Assessing Security Properties of Software Components: A Software Engineer's Perspective, Proceedings of the 2006 Australian Software Engineering Conference (ASWEC '06), IEEE.

[6] Alvaro, Almeida, Meira, Software Component Certification: A Survey, Proceedings of the 2005 31st EUROMICRO Conference on Software Engineering and Advanced Applications, 2005 IEEE.

[7] J.H.Poore, Harlan D.Mills, David Mutchler, Planning And Certifying Software Reliability, IEEE 1993.

[8] Hailiang Mei, Johan Lukkien and Johan Muskens, A Compositional Claim-based Component Certification Procedure, Proceedings of the 30th EUROMICRO Conference (EUROMICRO '04), 2004 IEEE

[9] Mark Grechanik, Dewayne E. Perry, Don Batory, Proceedings of the fifth International Conference on Commercial-off-the-Shelf (COTS)- Based Software Systems (ICCBSS 2006), 2006 IEEE.

[10] M. Kelkar, R. Perry, R.Gamble, A.Walvekar The Impact of Certification Criteria on Integrated COTS-Based Systems, Sixth International Conference on Commercial-off-the-Shelf (COTS)- Based Software Systems (ICCBSS'07), 2007 IEEE.

[11] Claes Wohlin and Per Runeson, Concise Papers, 1994 IEEE.

[12] John Morris, Gareth Lee, Kris Parker, Gary A. Bundell and Chiou Peng Lam, Software Component Certification, 2001 IEEE.

[13] http://en.wikipedia.org/wiki/TCSEC

[14] http://en.wikipedia.org/wiki/ITSEC

[15] http://en.wikipedia.org/wiki/Common_Criteria

[16] Anup K. Ghosh & Gary McGraw, An Approach for Certifying Security in Software Components, produced at Reliable Software Technologies.

[17] http://www.ociweb.com/cnb/CORBANewsBrief-200205.html