USES OF PKI FOR PROCESS AUTHORIZATION

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

FEYZA TAŞKAZAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF COMPUTER ENGINEERING

DECEMBER 2003

Approval of the Graduate School of Natural and Applied Sciences.

_____

Prof. Dr. Canan Özgen

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Prof. Dr. Ayşe Kiper

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____	_____
Dr. Murat Erten	Dr. Attila Özgit
Co-Supervisor	Supervisor

Examining Committee Members

Prof. Dr. Semih Bilgen	_____
Dr. Murat Erten	_____
Dr. Attila Özgit	_____
Dr. Cevat Şener	_____
Dr. Onur Tolga Şehitoğlu	_____

ABSTRACT

USE OF PKI FOR PROCESS AUTHORIZATION

TAŞKAZAN, FEYZA

MSc, Department of Computer Engineering

Supervisor: Dr. Attila Özgit

Co-Supervisor: Dr. Y. Murat Erten

DECEMBER 2003, 66 pages

Enterprises require an information security solution that provides privacy, integrity, authentication and access controls for processes. License management systems are developed to be a solution for process authorization in different platforms. However, security threats on processes cannot be controlled with existing license management mechanisms. The need is a complete system that is independent from implementation, platform, and application. In this thesis, we design a complete system for process authorization based on Public Key Infrastructure (PKI) technology.

Keywords: Public Key Infrastructure (PKI), Process Authorization, Binary Patching, Process Creation in Linux.

ÖZ

İŞLEM YETKİLENDİRME İÇİN PKI KULLANIMI

TAŞKAZAN, FEYZA

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Dr. Attila Özgit

Tez Yöneticisi Yardımcısı: Dr. Y. Murat Erten

ARALIK 2003, 66 sayfa

Bilgisayarların günlük hayatta hızla artan kullanımı yazılımların gizliliğinin, bütünlüğünün, erişim ve yetkilendirmelerinin denetlenebileceği bir bilgi güvenlik sistemi oluşturulması gereksinimini artırmıştır. Erişim ve yetkilendirme denetimi için lisans mekanizmaları geliştirilmiştir. Bu mekanizmalar denetimlerinin zor olması, uygulama ve işletim sistemlerinden bağımsız olarak çalışamaması ve uygulamalar üzerindeki güvenlik açıklarını denetleyemedikleri için bilgi güvenliği olarak kullanılamamaktadır. Bu tezde işlem yetkilendirmede kullanılabilecek, gereksinimleri bütünüyle karşılayabilecek, sayısal sertifika teknolojisi kullanılarak bir işlem yetkilendirme sistemi tasarlanmaktadır.

Anahtar Kelimeler: Sayısal Sertifikalar, Açık Anahtarlama Altyapısı, İşlem Yetkilendirme Sistemleri, Çalıştırılabilir Kod Ekleme.

ACKNOWLEDGMENTS

I express sincere appreciation to Dr. Attila ÖZGİT for his guidance and insight throughout the research. To my husband, Gökhan, I offer sincere thanks for his unshakable faith in me and his willingness to endure with me. To my family, I offer thanks for their support in all steps of my education life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this century, data communication has become a fundamental part of computing. Worldwide network gather data about diverse subjects as atmospheric conditions, crop production and airline traffic. In addition, people communicate by electronic mailing lists, so they share information as source codes, executables for their personal computers. In the scientific world, also data networks are important to exchange scientific information.

The creation of networks depends on government agencies. They have realized importance and potential of open system interconnection technology for many years. This interconnection technology is called Internet [1]. Government agencies began to research on the open system interconnection technology and this research has been called Defense Advanced Research Projects Agency (DARPA). The DARPA technology includes a set of network standards that specify details of how computers communicate, as well as a set of conventions for interconnecting networks and routing traffic.

TCP/IP, officially named TCP/IP Internet Protocol Suite is used to communicate across any set of interconnected networks. As the technology and researches are being developed, DARPA, using TCP/IP, is being used by lots of groups for communication among geographically distant sites, known as the connected Internet, the DARPA/NFS Internet, the TCP/IP Internet, or just Internet [1].

After 1990, when the Internet was mostly used in academic environment, the interconnection of computers has been grown so rapidly that it has been available on desktop computers as well as offices.

Together with the incredible growth of the Internet, the number of security problems has been increasing too. One of the security problems is unexpected behaviour of program flows in a computer. Computing is just programs running on processors, including instructions integrated internal chips, firmware in read-only memory, the basic hardware-to-software interface, device drivers, operating systems, network implementations, data base management systems, utilities, applications and user programs. Thus, in one form or another, protection of programs is at the heart of security in computing [5].

If the system is known as a trusted system, the heart of security in the computing must be provided. The protection of programs is important for developers and programmers because of its license security and for system

administrators because of system security. Therefore, there is a need for a strong authorization mechanism, which can be used as license mechanism and also can be used to protect processes from security threats.

1.1 AUTHORIZATION MECHANISM

Authorization is the act of determining whether an identified entity has a right or authority to execute a specific service or access a specific piece of information [7]. In multi-user computing system, it is defined as the process of making decision whether to grant or deny access to a resource. The resource is a critical component of systems. Early time-sharing systems used accounts assigned to individuals and an associated secret password that was supposed to be known only by the account owner [6].

Such systems are still in use today but, as the number and variety of systems and resources has grown exponentially, and systems comprising of multiple entities that are inter-connected and resources that are controlled by these entities need more generalized and flexible authorization mechanisms.

Authorization mechanisms can be used for all applications, and also processes in a system according to security aspects. This brings several benefits for developers and system administrators, controlling process usage, having adequate protection for software publisher, creating a so-called trusted operational environment and also secure operational environment.

An authorization mechanism can be useful when it is application independent, implementation independent, operating system platform independent, and having standard authorization coding style. These goals bring ease in administration of process authorization mechanisms.

1.2 PROBLEM STATEMENT

Using resources without authorization is an important problem. Unauthorized processes cannot be controlled easily on today's operating systems. Therefore, there is a need for so-called "trusted" applications, and so-called "trusted" operating systems.

There are several mechanisms for blocking the unauthorized usage of processes. One and the most used method is using license mechanism. The license mechanism will be used for a wide area on the operating system environment, kernel, applications, etc. However, license mechanisms have several weaknesses; the security check of the mechanism cannot be followed by system administrators, the same license can be used on different computers, the corporation does not apply the license agreement.

The main purpose of this thesis is finding an approach for an appropriate process authorization system that is application and platform independent, and provides so-called "trusted" applications and so-called "trusted" operating systems.

1.3 THESIS OUTLINE

The rest of the thesis is structured as follows.

Chapter 2 - *Security Issues for Process Authorization.* In this chapter the security issues of process authorization in general are discussed, and some of today's most commonly used process authorization methods are described.

Chapter 3 – *PKI Based Process Authorization.* This chapter identifies the major needs for trusted process authorization mechanism and its requirements. Certified base process authorizations are described in detail.

Chapter 4 – *Design and Implementation of Process Authorization System.* This chapter starts with the specifications of the process authorization system. Then it presents details of issues related to the design of the system. Also implementation details are given.

Chapter 5 – *Test and Evaluation.* In this chapter, performance of the system that is implemented will be explained. Test results are interpreted.

Chapter 6 – *Summary and Conclusions.* This chapter gives a summary of the thesis study together with the further work that might be built upon this study and final conclusions.

CHAPTER 2

SECURITY ISSUES FOR PROCESS AUTHORIZATION

An authorization mechanism is needed for processes in a system after the growing of using Internet. Two of the most important reasons for the need of authorization system are insufficient license mechanisms and security attacks to systems.

Nowadays, everything is shared between users in the Internet. Almost every application that is needed by users can be easily found and downloaded from internet regardless the fact that its license is free or not. If an application is licensed under a free license mechanism, there is no problem to use it. However, an application may have license mechanism that has an authorization for whom can be used. This authorization mostly depends on commercial programs, needing some cost in order to use the program. Whereas, these authorization methods can be easily be breakable with the help of Internet -cracks, serial numbers, dongles etc. Because of the Internet and sharing, license mechanisms that hold copyrights become insufficient.

Therefore, there is a need for trusted authorization mechanisms that is application independent and can be used to control the program with the system that the program is executed on.

Furthermore, sharing everything in the Internet creates an attacker group in the world. These attackers share applications that are implemented for threading other systems security. These applications exploit vulnerabilities in computing systems and these are called as malicious programs. One type of malicious programs is trap door. This is a secret entry point into a program that allows someone that is aware of the trap door to gain access without going through the usual security access procedures. Programmers for debugging and testing programs have used trap doors for many years. However, it becomes a security threat when hidden intent programmers to gain unauthorized access use them. A logic bomb is another malicious program that is embedded in some applications in a system and that is set to explode when certain conditions are met. The other is Trojan horse that is a program or procedure that contains a hidden code for performing some unwanted or harmful function. The most used malicious programs are viruses. A virus is a program that can infect other programs in the system by modifying them [10]. An authorization mechanism is needed for protecting the system against execution of malicious programs by controlling all process executions in that system.

There are many process authorization mechanisms. In the following sections, the meaning of process authorization and its uses will be discussed.

2.1 PROCESS AUTHORIZATION

In today's Internet, there are a large and growing number of scenarios that require authorization decisions. Such scenarios include electronic commerce, execution of downloadable code, privacy protection, remote resource sharing, etc.

Computer security aims that ensuring the access to resources is restricted to and available to those parties with legitimate access permissions. Three mutually supportive mechanisms together provide the foundation for achieving this goal: authentication, access control, and audit. A system into entities that are interconnected and resources that are controlled by entities can be abstracted today's technology. Entities may include users, operating systems, processes, threads, objects, etc. Resources may include information, files, network connections, methods of objects, etc. When an entity wants to access a resource controlled by another entity, it sends a request to that entity. The entity that wants to access the resource is called the requester and the entity that controls the resource is called the authorizer.

Traditionally, when an authorizer receives a request, it first identifies the requester. This task of determining a requester's identity in a rigorous manner

is called authentication. In other words, authentication answers the question that made this request with an identity. Knowing the identity of the requester, the authorizer then decides whether this identity is allowed to access the requested resource. This step is called access control. The audit process gathers data about activities in the system and analyzes it to discover security violations and diagnose their cause. Analysis can occur off-line after the fact or it can occur on-line more or less in real time.

Using authorization mechanism for processes has several benefits in software industry both for authorizer and for requestor. For an authorizer, the key factors of using process authorization mechanism are process usage control and the lack of adequate protection for software publisher. In addition, requestors must deal with multiple products, from multiple software publishers, on multiple platforms, with multiple process authorization models. The exponential growth of using software, in complexity, there is a clear requirement for an overall framework for process authorization methods.

A process authorization mechanism must be extensible, flexible and comprehensive. In addition, it must be independent from software publisher, platform, and implementation. It must be easy to adapt to future technologies.

The primary goals of this specifications are to provide a consistent and standard means for the management of software licensing, to facilitate the availability of more flexible licensing terms, to enable cost-effective license

management and control, to provide access of licensing data in a heterogeneous and configuration independent environment, to prevent the system according to malicious codes.

2.2 USES OF PROCESS AUTHORIZATION

There are currently thousands of software products in use that rely upon technical process authorization mechanism to ensure compliance with license terms and conditions. Today, authorization systems used for protecting unknown usage of software, for protecting the network connected computer followed by the installation of trojaned binaries, root kits, worms and virus payloads [11], for protecting the system using by unknown user.

The usage of process authorization mechanism has different type of benefits: prevent systems, authenticate users, control the usage of software, control the number of users of distributed systems [12], etc.

Authorization in an Internet scenario is significantly different in centralized systems or even in distributed systems that are closed or relatively small. In these older settings, authorization of a request is divided into authentication and access control.

In Internet authorization scenarios, often there is no relationship between a requester and an authorizer prior to a request. Because the authorizer does not

know the requester directly, it has to use information from third parties who know the requester better; normally, the authorizer trusts these third parties only for certain things and only to certain degrees. This trust and delegation aspect makes distributed authorization different from traditional access control.

Access control of an application is based on license mechanism. In this mechanism authorizer creates a key for requester and trusts him for only use this key on a machine and also in one installation. The idea in a distributed system is same, but requester can abuse trust. Unlicensed use of processes has always been a major concern for the software industry. Lately, the piracy problem has been highlighted by the introduction of the Internet as a distribution channel [8].

Authentication and access control mechanisms are being combined in these years in order to increment the security of process authorization systems.

2.3 WORKS ON PROCESS AUTHORIZATION MECHANISMS

In the computing life, a license management system is mostly used as process authorization mechanisms. However, these systems do not have a support for protecting applications against malicious codes. After the development of digital certificates, a new approach, which is based on certificates, is being used for process authorization systems.

A new study on the license management systems that is based on smart-cards and digital certificates is developed in Helsinki University of Technology. In this study, protocols are based on signed delegation certificates that are mostly stored outside a smart-card. New hardware, a smart-card reader and cards, capable of public-key signatures, are needed to have a simple system for implementing and analyzing an easy one for users [9].

Using certificates provides the security on the network, and using smart-cards brings a solution for distributed systems in that new system. The main threats are that they address are multiple installations of software from a single-license distribution medium and the production of fake copies by professional pirates. These types of copying appear to have the greatest impact on the software publishers' revenues.

Another certificate based process authorization study has been developed for anti-Trojan and Trojan detecting by testing executables in the kernel. This method finds an attempt that has been made to execute a file that has been tampered with and that the affected computer system either has warned you or has denied the execution. In respect to special case systems such as sacrificial hosts or honey pots, this system has an obvious advantage to detect as quickly as possible that an attack is in progress [11]. In this system, certificates are used for processing the kernel modules to prevent the system from Trojans. In that point the process authorization mechanism is used in kernel modules of BSD systems.

CHAPTER 3

PKI BASED PROCESS AUTHORIZATION

The need for trustable process authorization mechanisms is being solved by digital certificate technology. In the following sections, Public Key Infrastructure (PKI) and certificate based process authorization mechanisms will be discussed.

3.1 PUBLIC KEY INFRASTRUCTURE (PKI)

We have talked about all communication over the Internet uses the TCP/IP in Chapter 1. The great flexibility of the TCP/IP has let its worldwide acceptance as the basic Internet and intranet communications protocol. On the other hand, the fact that TCP/IP allows information to pass thorough intermediate computers makes it possible for a third party to interfere with communications by listening the communication channel, changing the information that is transmitted, and spoofing the channel. However, many sensitive personal and business communications over the Internet require precautions that address

threats above. Fortunately, a set of well-established techniques and standards known as public-key cryptography make it relatively easy to take such precautions [13].

Public key cryptography facilitates encryption and decryption between two communicating parties to protect information they send to each other and allows the recipient to understand that the information has been changed or not. In the main structure of public key cryptography two key pairs, public and private, are used for encryption and decryption and for controlling data changes.

After the development of public key cryptography, another approach is being used for security threats in the communication channels: Digital Certificates or just Certificates. Digital certificate is simplifying the task of establishing whether a key truly belongs to its owner. A certificate is defined as "a document containing a certified statement, especially as to the truth of something." [5]. A certificate is a form of credential. Your passport, your social security card, or your birth certificate might be the examples of a certificate in the life. Each of these has some information on it identifying you and some authorization stating that someone else has confirmed your identity. Like these, a digital certificate is important for confirmation of your identity in the communication life.

The ITU X.509 international standard defined a format that is most widely accepted, for digital certificates. Every X.509 certificate consists of two sections; one of them is the data section that includes version number of X.509 standard that is supported by the certificate, certificate serial number, public key cryptography algorithm, public key, certificate issuer information, validity dates, certificate subject name, and optional extensions. The other part of them is the signature section that includes cryptographic algorithm of the issuer and issuer certificate [15].

Public Key Infrastructure (PKI) is being developed for creating and managing digital certificates. All PKIs have two basic operations. One of them is certification operation that is the process of binding a public key value to an individual, or an organization, to another distinguishing value describing an entity, or even to a piece of information, such as a permission or credential. The other is the validation operation that is the process of verifying that a certificate is indeed valid. Furthermore, these two operations are known as the basic characteristic of all PKIs.

Key registration for issuing a new certificate for a public key, key selection for obtaining a party's public key, trust evaluation for determining whether a certificate is valid, and certificate revocation for canceling a previously issued certificate services are performed by certification and validation operations of a PKI.

3.1.1 Components of a PKI

PKI needs some components for providing services that are described in the previous section: Certification Authority (CA) and a Registration Authority (RA).

Certification Authority is a trusted agency that signs certificates with its private key and lets other verify certificates by the usage of the corresponding public key. Issuing and revoking public key certificates are main functions of a CA.

Registration Authority is a trusted agency that provides communication between certificate holders and Certification Authority. Vouching for the binding between public keys and certificate holder identities and other attributes is the main function of a RA [3].

These two components provide a PKI to communicate with certificate holders and services of certification to them.

3.1.2 Digital Certificate Format

The general format of a X.509 certificate can be seen in Figure 3.1 that includes the following elements:

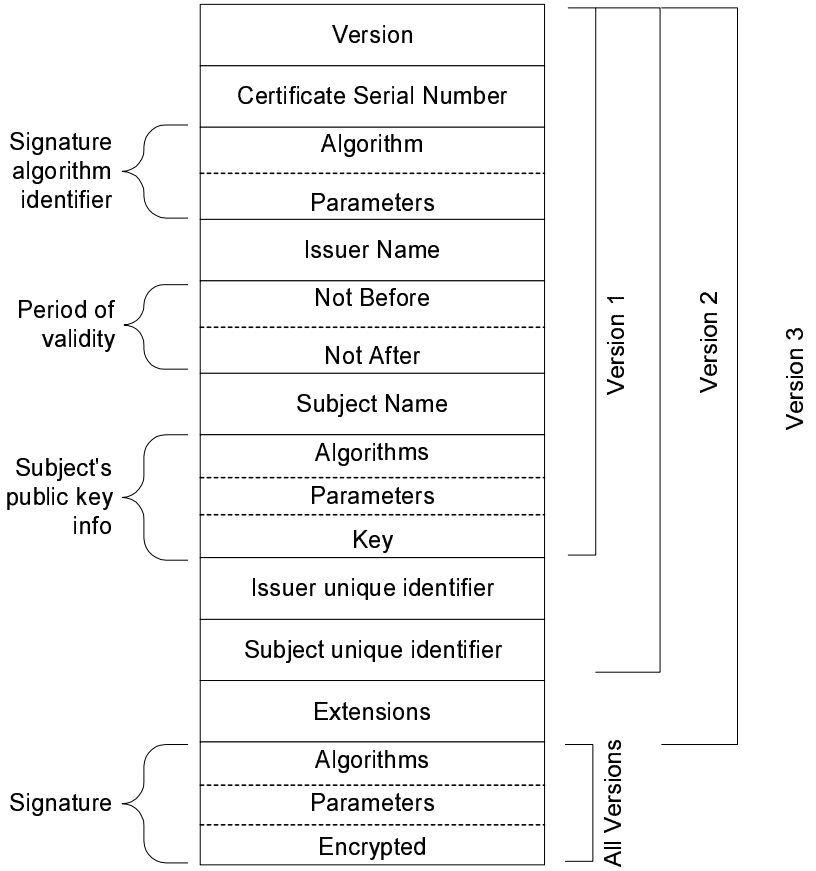

Figure 3.1: X.509 Certificate Format

- Version: A number that determines the version of the certificate format. The default value is version 1. If the issuer unique identifier and subject unique identifier are present, the value is version 2. If one or more extensions are present, then the value is version 3.
- Serial Number: An integer value that is unique with in the issuing CA.
- Signature Algorithm Identifier: The algorithm that is used to sign the certificate, together with any associated parameters.

- Issuer Name: The X.500 name of the CA by which this certificate is created and signed.
- Period of Validity: This consists of two dates: the first and last date on which the certificate is valid.
- Subject Name: The name of the user to whom this certificate refers. That is, this certificate certifies the public keys of the subject who holds the corresponding private key.
- Subject' s Public Key Information: The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.
- Issuer Unique Identifier: An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.
- Subject Unique Identifier: An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.
- Extensions: A set of one or more extension fields.
- Signature: This covers all of other fields of the certificate; it contains the hash code for other fields, encrypted with the CA private key. This field includes the signature algorithm identifier [10].

User certificates may be generated in different formats by the Certification Authority One of them is the text format. In this format, you can see and get

all certificate fields. You can see an example of a certificate, which is in text format, below:

```
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number: 5 (0x5)
      Signature Algorithm: sha1WithRSAEncryption
  Issuer: C=TR, O=Middle East Technical University, OU=Computer Center, CN=METU-
CA/Email=caadmin@metu.edu.tr
      Validity
         Not Before: Aug 12 13:12:30 2003 GMT
         Not After : Aug 11 13:12:30 2004 GMT
      Subject: C=TR, O=METU-CA, OU=Computer Center, CN=Registration Authority,
SN=5
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
         RSA Public Key: (1024 bit)
            Modulus (1024 bit):
               00:b9:8b:06:61:3b:4f:32:4f:e0:85:5f:7a:92:3d:
               d6:09:b9:75:eb:46:90:f0:4a:8b:6d:8c:3b:7a:0d:
               25:29:8a:19:b3:7b:d8:f5:5e:c3:03:48:c8:fa:36:
               d2:c1:55:55:86:35:fe:db:dd:4a:84:09:8b:8c:dc:
               68:ec:f9:d6:96:41:6f:6e:ab:cc:d0:24:74:3b:31:
               c3:86:2e:ad:8e:28:74:5a:e3:83:c9:1b:3d:5b:3c:
               e1:ae:b0:ae:f2:a9:ca:a4:88:c5:5a:f8:f6:6d:5c:
               c9:02:5a:da:13:2e:24:c3:9e:e8:a1:de:91:02:a8:
               b9:25:61:8d:a4:5a:35:de:db
            Exponent: 65537 (0x10001)
      X509v3 extensions:
         X509v3 Subject Key Identifier:
            FB:3E:01:F0:35:B8:91:8A:68:8C:9F:5C:D8:29:55:23:AF:C0:91:60
         X509v3 Authority Key Identifier:
            keyid:EF:79:CE:DF:C6:C3:2D:23:06:78:8F:95:60:84:13:33:31:2D:D7:06
            DirName:/C=TR/O=Middle East Technical University/OU=Computer
Center/CN=METU-CA/Email=caadmin@metu.edu.tr
            serial:00
         X509v3 Subject Alternative Name:
            email:caadmin@metu.edu.tr
         X509v3 Issuer Alternative Name:
            email:caadmin@metu.edu.tr
   Signature Algorithm: sha1WithRSAEncryption
      76:49:5e:5f:bf:ab:80:9f:7c:04:e6:91:6e:f1:fc:c7:96:ce:
      d6:69:4d:17:f1:8e:bb:20:52:e7:be:14:3e:cb:15:26:ef:ad:
      d7:7d:34:58:5a:75:19:ca:88:c4:67:86:26:cf:9a:ee:25:32:
      86:2b:0d:82:c6:a6:46:f3:78:42:dd:b2:a4:0d:68:5c:bb:b3:
      99:3c:92:fb:31:71:28:41:57:58:5b:1a:d1:06:f4:68:f4:17:
      a0:c9:e5:fa:c1:09:54:1c:0c:d9:26:1e:c6:e0:53:ef:33:87:
      c6:7f:03:86:9a:76:d6:75:eb:c1:10:27:af:0d:15:b9:3e:40:
      9c:4c:4f:c6:9b:45:84:e1:55:f6:e2:e4:08:f5:38:f6:81:be:
      38:d8:e7:5e:6f:c3:7e:f6:64:82:c1:74:b3:c6:d5:6e:61:66:
      ec:5a:c9:d7:50:89:38:fb:69:8b:99:bf:64:1c:16:14:3e:24:
      b3:41:c2:c6:be:f9:05:20:dd:ae:91:24:7a:f6:54:60:0e:7a:
```

d5:b8:e5:e5:8c:5c:f7:50:10:ef:0c:28:0e:09:76:6e:30:df:
d0:73:02:1e:e5:e8:6c:d4:0f:0c:cc:97:4f:7e:ac:62:46:35:
e2:3b:ba:63:23:db:e1:24:91:41:44:a8:49:55:0a:5f:eb:cd:
38:0b:32:17

In addition, a certificate can be created in DER format, which is a binary format to encode certificates. If this DER format has been encoded with Base64 and added a header and footer, PEM format, which is a text format that can be used to encode certificates, can be obtained. The PEM format of the above text-formatted certificate can be seen below:

-----BEGIN CERTIFICATE-----
MIIFSjCCBDKgAwIBAgIBBTANBgkqhkiG9w0BAQUFADCBiDELMAkGA1UEBhMCVFIx
KTAnBgNVBAoTIE1pZGRsZSBFYXN0IFRlY2huaWNhbCBVbml2ZXJzaXR5MRgwFgYD
VQQLEw9Db21wdXRlciBDZW50ZXIxEDAOBgNVBAMTB01FVFUtQ0ExIjAgBgkqhkiG
9w0BCQEWE2NhYWRtaW5AbWV0dS5lZHUudHIwHhcNMDMwODEyMTMxMjMwWhcNMDQw
ODExMTMxMjMwWjBmMQswCQYDVQQGEwJUUjEQMA4GA1UEChMHTUVUVS1DQTEYMBYG
A1UECxMPQ29tcHV0ZXIgQ2VudGVyMR8wHQYDVQQDExZSZWdpc3RyYXRpb24gQXV0
aG9yaXR5MQowCAYDVQQFEwE1MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC5
iwZhO08yT+CFX3qSPdYJuXXrRpDwSottjDt6DSUpihmze9j1XsMDSMj6NtLBVVWG
Nf7b3UqECYuM3Gjs+daWQW9uq8zQJHQ7McOGLq2OKHRa44PJGz1bPOGusK7yqcqk
iMVa+PZtXMkCWtoTLiTDnuih3pECqLklYY2kWjXe2wIDAQABo4ICYjCCAl4wCQYD
VR0TBAIwADARBglghkgBhvhCAQEEBAMCBLAwCwYDVR0PBAQDAgXgMCkGA1UdJQQi
MCAGCCsGAQUFBwMCBggrBgEFBQcDBAYKKwYBBAGCNxQCAjA5BglghkgBhvhCAQ0E
LBYqUmVnaXN0cmF0aW9uIEF1dGhvcml0eSBPcGVyYXRvciBvZiBNRVRVLUNBMB0G
A1UdDgQWBBT7PgHwNbiRimiMn1zYKVUjr8CRYDCBtQYDVR0jBIGtMIGqgBTvec7f
xsMtIwZ4j5VghBMzMS3XBqGBjqSBizCBiDELMAkGA1UEBhMCVFIxKTAnBgNVBAoT
IE1pZGRsZSBFYXN0IFRlY2huaWNhbCBVbml2ZXJzaXR5MRgwFgYDVQQLEw9Db21w
dXRlciBDZW50ZXIxEDAOBgNVBAMTB01FVFUtQ0ExIjAgBgkqhkiG9w0BCQEWE2Nh
YWRtaW5AbWV0dS5lZHUudHKCAQAwHgYDVR0RBBcwFYETY2FhZG1pbkBtZXR1LmVk
dS50cjAeBgNVHRIEFzAVgRNjYWFkbWluQG1ldHUuZWR1LnRyMDoGCWCGSAGG+EIB
BAQtFitodHRwczovL2NhLmNjLm1ldHUuZWR1LnRyL3B1Yi9jcmwvY2FjcmwuY3Js
MDoGCWCGSAGG+EIBAwQtFitodHRwczovL2NhLmNjLm1ldHUuZWR1LnRyL3B1Yi9j
cmwvY2FjcmwuY3JsMDwGA1UdHwQ1MDMwMaAvoC2GK2h0dHBzOi8vY2EuY2MubWV0
dS5lZHUudHIvcHViL2NybC9jYWNybC5jcmwwDQYJKoZIhvcNAQEFBQADggEBAHZJ
Xl+/q4CffATmkW7x/MeWztZpTRfxjrsgUue+FD7LFSbvrdd9NFhadRnKiMRnhibP
mu4lMoYrDYLGpkbzeELdsqQNaFy7s5k8kvsxcShBV1hbGtEG9Gj0F6DJ5frBCVQc
DNkmHsbgU+8zh8Z/A4aadtZ168EQJ68NFbk+QJxMT8abRYThVfbi5Aj1OPaBvjjY
515vw372ZILBdLPG1W5hZuxayddQiTj7aYuZv2QcFhQ+JLNBwsa++QUg3a6RJHr2
VGAOetW45eWMXPdQEO8MKA4Jdm4w39BzAh7l6GzUDwzMl09+rGJGNeI7umMj2+Ek
kUFEqElVCl/rzTgLMhc=
-----END CERTIFICATE-----

3.1.3 Revocations of Certificates

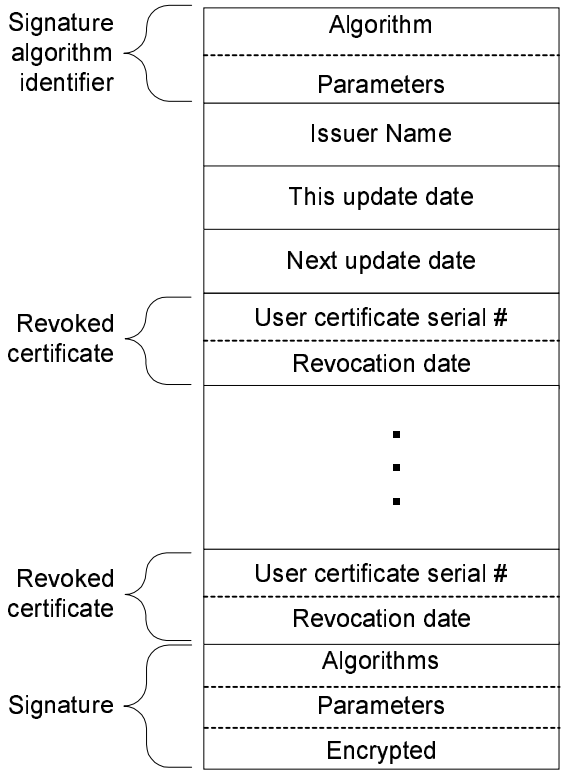

Figure 3.2: Certificate Revocation List

Recall from Figure 3.1 that each certificate includes a period of validity. Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires when the users' private key is assumed to compromise, or when certificates issuer no longer certifies the user, or when issuer certificate is assumed to be compromised.

Each Certification Authority must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists are known as Certificate Revocation Lists. Each Certificate Revocation List (CRL) includes (Figure 3.2) the issuers name, the date in which the list was created, the date in which the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of revoked certificate and revocation date of the certificate [10].

CRLs are important in the validity check operation of a certificate for controlling if the certificate has been revoked before its validity date.

A CRL can be in text or PEM format. The text format of a CRL and its PEM format can be seen below:

```
Certificate Revocation List (CRL):
        Version 1 (0x0)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: /C=TR/O=Middle East Technical University/OU=Computer Center/CN=METU-
CA/emailAddress=caadmin@metu.edu.tr
        Last Update: Aug 22 07:40:33 2003 GMT
        Next Update: Sep 21 07:40:33 2003 GMT
Revoked Certificates:
    Serial Number: 09
        Revocation Date: Aug 22 07:39:58 2003 GMT
    Serial Number: 0A
        Revocation Date: Aug 22 07:39:35 2003 GMT
    Serial Number: 0B
        Revocation Date: Aug 22 07:39:09 2003 GMT
    Serial Number: 0C
        Revocation Date: Aug 22 06:47:03 2003 GMT
    Serial Number: 0D
        Revocation Date: Aug 22 07:38:44 2003 GMT
    Serial Number: 0E
        Revocation Date: Aug 22 07:38:21 2003 GMT
    Serial Number: 0F
```

```
    Revocation Date: Aug 22 07:37:55 2003 GMT
Serial Number: 10
    Revocation Date: Aug 22 07:37:25 2003 GMT
Signature Algorithm: md5WithRSAEncryption
    28:4c:e1:ee:95:70:97:f9:52:df:6c:63:89:63:ad:72:4e:40:
    8e:dd:49:c8:d7:79:b3:c3:c0:ac:c9:fd:e3:4f:ef:75:2e:6b:
    92:0c:a0:3d:e4:03:16:7c:5c:da:d4:17:1a:06:4d:e2:14:18:
    7c:d6:cb:e5:d9:da:a7:76:ad:3c:c7:52:94:bc:5d:41:40:3b:
    f0:a2:5e:64:6b:33:ca:63:ef:1e:73:1a:d2:a7:ae:f4:c6:1e:
    c0:3a:56:5a:ee:27:6d:7a:bd:a0:d2:e5:dc:c3:6a:94:ef:15:
    a1:e7:d6:5b:d8:31:ed:99:c7:74:d7:87:3f:48:cf:90:4c:d9:
    c0:0c:63:f2:43:56:1f:61:d0:90:2e:11:fe:56:75:9e:6d:41:
    8c:32:34:b2:2d:14:0d:cb:3c:07:d4:48:ec:33:d6:52:21:21:
    96:98:68:cb:f7:88:8e:59:35:f1:f4:91:11:2a:4d:22:91:81:
    a8:c3:b4:61:0e:76:6e:28:6e:a9:97:f4:47:80:6f:24:78:b9:
    01:59:62:d5:2f:67:d8:8a:d0:65:21:d4:c6:d8:c4:42:7c:fc:
    ee:85:3e:8a:a2:9b:e2:3c:86:db:b0:e0:14:bc:1d:49:c9:ab:
    31:16:df:32:af:c5:93:76:04:99:24:9e:68:28:2a:96:05:44:
    c5:9d:27:18
```

```
-----BEGIN X509 CRL-----
MIICczCCAVswDQYJKoZIhvcNAQEEBQAwgYgxCzAJBgNVBAYTAlRSMSkwJwYDVQQK
EyBNaWRkbGUgRWFzdCBUZWNobmljYWwgVW5pdmVyc2l0eTEYMBYGA1UECxMPQ29t
cHV0ZXIgQ2VudGVyMRAwDgYDVQQDEwdNRVRVLUNBMSIwIAYJKoZIhvcNAQkBFhNj
YWFkbWluQG1ldHUuZWR1LnRyFw0wMzA4MjIwNzQwMzNaFw0wMzA5MjEwNzQwMzNa
MIGgMBICAQkXDTAzMDgyMjA3Mzk1OFowEgIBChcNMDMwODIyMDczOTM1WjASAgEL
Fw0wMzA4MjIwNzM5MDlaMBICAQwXDTAzMDgyMjA2NDcwM1owEgIBDRcNMDMwODIy
MDczODQ0WjASAgEOFw0wMzA4MjIwNzM4MjFaMBICAQ8XDTAzMDgyMjA3Mzc1NVow
EgIBEBcNMDMwODIyMDczNzI1WjANBgkqhkiG9w0BAQQFAAOCAQEAKEzh7pVwl/lS
32xjiWOtck5Ajt1JyNd5s8PArMn940/vdS5rkgygPeQDFnxc2tQXGgZN4hQYfNbL
5dnap3atPMdSlLxdQUA78KJeZGszymPvHnMa0qeu9MYewDpWWu4nbXq9oNLl3MNq
lO8VoefWW9gx7ZnHdNeHP0jPkEzZwAxj8kNWH2HQkC4R/lZ1nm1BjDI0si0UDcs8
B9RI7DPWUiEhlphoy/eIjlk18fSRESpNIpGBqMO0YQ52bihuqZf0R4BvJHi5AVli
1S9n2IrQZSHUxtjEQnz87oU+iqKb4jyG27DgFLwdScmrMRbfMq/Fk3YEmSSeaCgq
lgVExZ0nGA==
-----END X509 CRL-----
```

3.2 PKI BASED PROCESS AUTHORIZATION

In the previous section, PKI is discussed with its components, certificate and CRL formats. In these years, to gain a functional process authorization system PKI bases are used.

A PKI can be used to create and manage very reliable digital credentials for individual network users. Typically, these credentials are based on the X.509

standard and carry only a limited set of information about the holder. In practice, the question arises as how an application or resource can make appropriate authorization decisions upon receipt of such a credential.

At the basic level, the digital credential provides a trustworthy digital handle that is unique to the holder of the certificate. Validating that credential provides a starting point for deciding what entitlements that individual or entity might have, either by use of information within the credential or by discovering trustworthy information associated with that handle. This process of authorization can take many different forms depending on the nature of the certificate format.

Elements in an X.509 certificate might be used in some fashion to facilitate authorization. These include the issuer, the institution that stands behind the credential, the subject, an identifier associated with the holder, the public key associated with the holder's private key, the certificate serial number, which is unique for each credential from an issuer, and optional extensions.

The main structure of a certificate is suitable to be used for authorization according to its different decidable objects. Therefore, many scenarios may be designed for certificate based process authorization mechanisms.

CHAPTER 4

DESIGN AND IMPLEMENTATION OF PROCESS AUTHORIZATION MECHANISM

Software license management is mainly used as process authorization mechanisms. However, there are several weaknesses on them: They can be easily bypassed, different requesters on different machines can use their keys and they cannot control applications against security threats. Therefore, there must be a complete solution for an ideal process authorization mechanism. That may be used as a license management system and may prevent applications from security threats. In addition, this system should have authorization framework specifications.

In the last two years, after PKI's use was increased, methods have begun to use certificate based process authorization for authentication and access control. These methods are not operating system platform and implementation independent and these are not cost-effective solutions.

As a result, there is an open door for designing a new certificate based process authorization system that answers all the needs for authentication and access control.

In the previous chapters, basic requirements for an authorization system have been reviewed. This chapter is going to present our approach for an authorization system that uses PKI. The design and implementation of the system will be given in detail.

4.1 DESIGN ISSUES

The obvious design goal of our system is to allow only trusted processes in a system that has a valid process certificates and system certificates. This section explains issues that were taken into account during the design of the system in order to achieve this goal in detail.

4.1.1 Managing Digital Certificates

A PKI is based on digital certificate management. Operations, key-pair creation, certificate requests, signing certificates and creating certificate revocation list may be done by an open source PKI implementation in Internet X.509 standard, called PKIX [3]. PKIX describes basic certificate fields and extensions to be supported for certificates and certificate revocation lists, also it covers cryptographic algorithms. In addition, PKIX supports management

protocol that provides the data structure used for PKI management messages. PKIX entities can be seen in Figure 4.1.

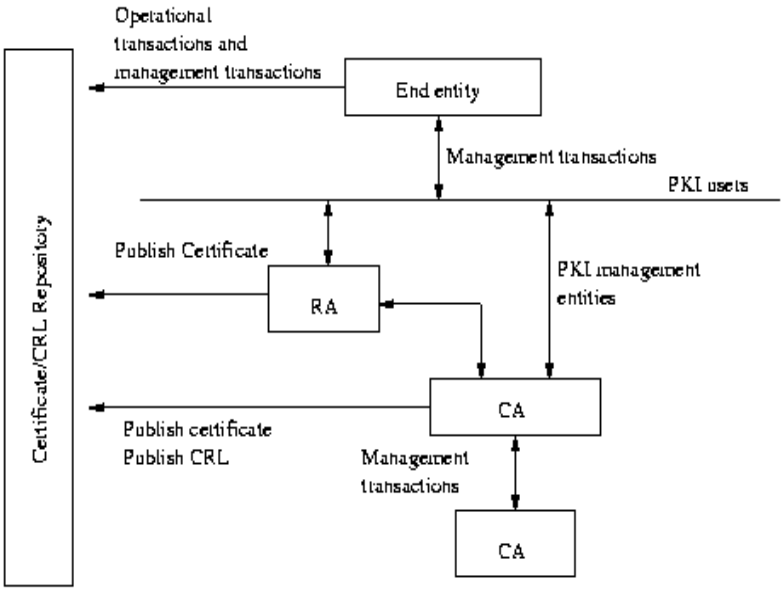

Figure 4.1: PKIX Entities

The most popular and useful open-source PKI implementation is OpenCA. OpenCA is a set of cgi scripts, which are written with perl, and it uses OpenSSL cryptographic libraries for key operations.

4.1.2 Logical Flow of Certificate Validity

In order to assert the validity of a certificate, the Certification Authority (CA) certificate and Certificate Revocation List (CRL) files have to be taken from the CA server. In the validation operation, the CA certificate validity is

controlled firstly by using the certificate validity date field. This validity date is compared by system date. If the CA certificate is valid, then the validity of the client certificate, which is created for processes or systems, is controlled by using CA certificate for checking the issuer name of the certificate and using the validity date of client certificate for checking the validity date interval with system date.

However, there is another issue for controlling validity of a client certificate, which controls the certificate revocation lists too. CRLs, as described before, are lists that consist of all revoked but not expired certificates issued by their CA. Therefore, any client certificate may be placed in the CRL, which is issued by its CA. The client certificate serial number should be searched in the valid CRL of its CA. The flow chart of validity operation can be seen in Figure 4.2.

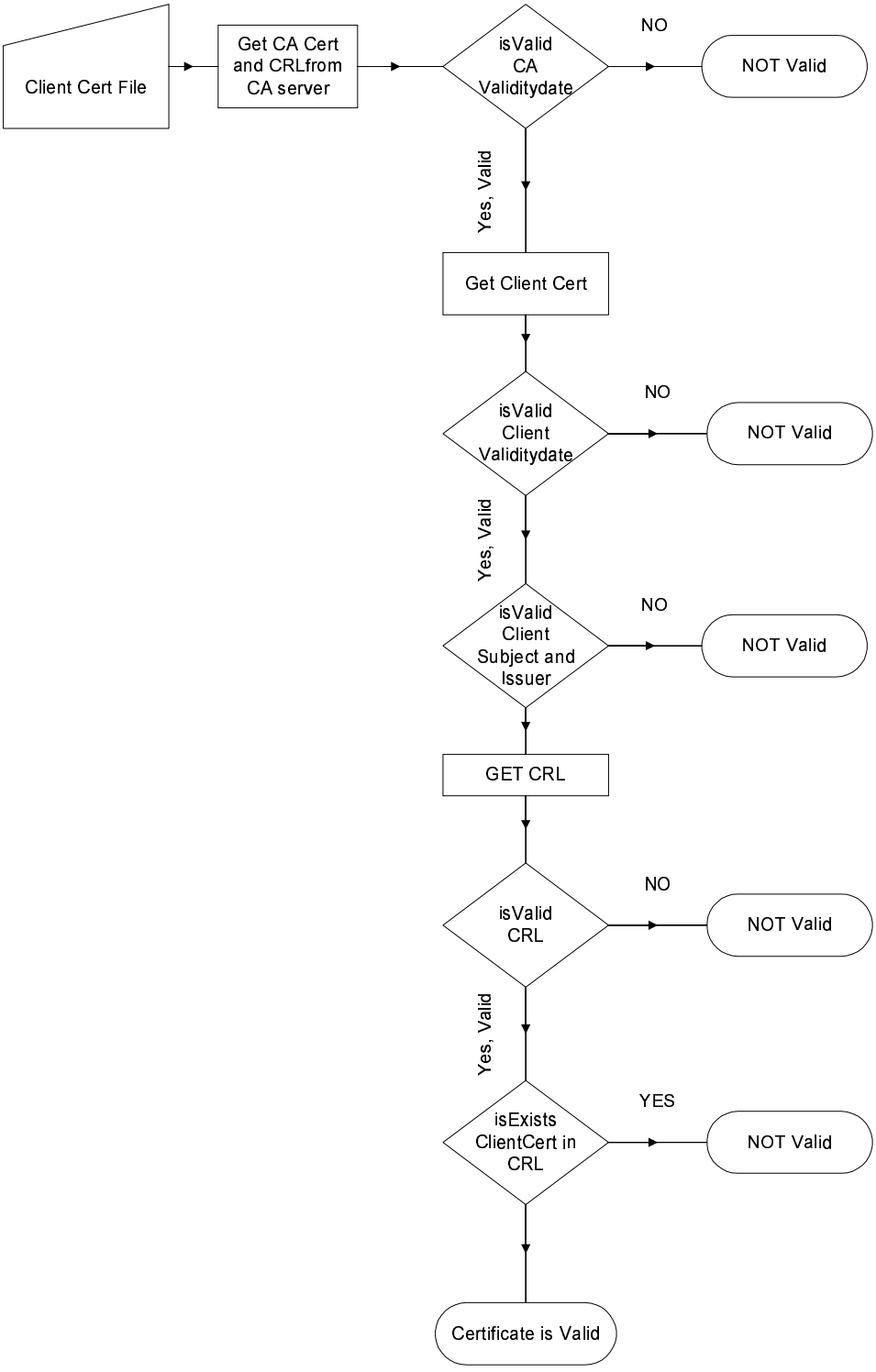

Figure 4.2: Flow Chart of Validity Operation

4.1.3 Logical Flow of Binary Patching Mechanism

Virus developers use binary patching mechanisms to provide a change in the execution of any executable in general. The base of the patching mechanism knows the exact size of the binary file called patch, which may be added to the beginning of an original executable binary. When a patch is prepared for appending any executable binary, it should have a code segment that provides the execution of the original binary after the patch execution. When a patch is ready for being added to the original executable binary, the original one is appended to the patch and the gained binary is copied on to the originals path. When this gained binary is processed in the system, the patch is firstly executed then the original one is executed.

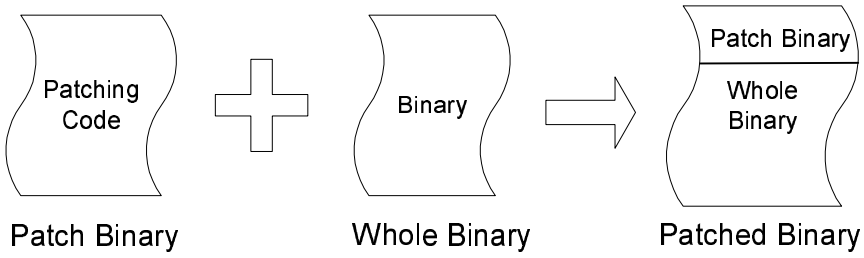

Figure 4.3: Binary Patching Mechanism

The code segment for execution of original binary that is placed in the patch first opens the executable binary that is patched, for reading itself in binary format. It calculates the code segment address of the original binary according to patch size. It seeks the file pointer to this address and copies the part from the address place of the file pointer to the end of binary file to a temporary

file. At the end of the execution of the patch, it executes the temporary file, which consists of only the original execution binary of the patched file. The flow chart of the execution of a patched binary will be seen in Figure 4.4.

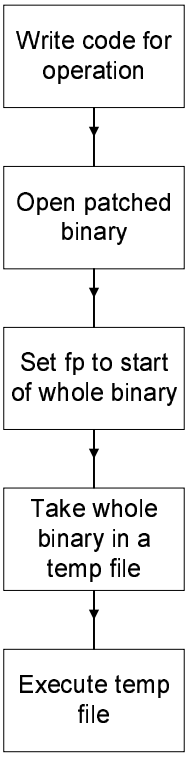

Figure 4.4: Flow Chart of Binary Patching Mechanism

4.1.4 Process Controlling in an Operating System

Every process in an operating system is controlled by a process creation function in the kernel. When a process is executed in a system, a new process is forked, and system create process function is called in the kernel level. This function finds an empty slot in the process table for the new process initializes process descriptor and reads the process binary file from the storage [17]. This

process creation functions structure is same in every operating system, although the name and the programming code of the function may be different according to operating system kernels.

4.1.5 The Whole System

In the previous chapters, the need for a process authorization system is described. Our design uses digital certificates, so PKI, for process authorization. In our process authorization model, we will have a PKI implementation, systems, and processes that are executed on these systems. The PKI implementation is used for management of processes, systems certificates and certificate revocation lists. In the model, every system must have a certificate to manage processes that should be executed in that system. In addition, every process must have its own certificate and the system certificate to be executable on the system.

The CA certificate and CRL file should be taken from the CA server in each validity operations of processes and systems certificates. In a PKI implementation, the CA certificate file and CRL file can be seen publicly. Therefore, any host may take these by using HTTP or FTP connection to the CA server of these files.

In the model of the system, each process binaries should have two digital certificates: a process certificate and a system certificate. The process

certificate of a process binary should be created independently from the model of the system by using PKI implementation. Furthermore, the created process certificate and system certificate should be added to the process binary by using binary patching mechanism. When this process binary is executed, a validation operation should be run for controlling the validity of two certificates to provide so-called "trusted" applications in the system. The validation operation should be used by every trusted process in a system. Therefore, this should be a shared library in systems. The details of this system will be described in section 4.3.

Adding certificates to process binaries provides the so-called "trusted" applications in a system. The model of authorization system goal is to have these so-called "trusted" operating systems. Therefore, all processes in a system should be examined in the kernel level before they have been executed. The process creation function of the kernel should be studied to prevent the system from untrusted process binaries.

Furthermore, our approach will be a complete PKI based process authorization system with so-called "trusted" applications and so-called "trusted" operating systems.

4.2 IMPLEMENTATION DECISIONS

In this section, some of decisions that have been made before the implementation of the system are going to be discussed.

4.2.1 PKI Implementation

PKI implementation is a system that gets certificate requests, creates certificates, issue certificates with CA certificate, gets revocation requests, and creates revocation lists. As described in section 4.1.1 there are a few open-source PKI implementations to be used. One of the most used implementation is OpenCA.

OpenCA functions may be described as follows:

- Initializing the whole system with its Certification Authority. Certificate Authority can generate CA certificate and CA private key. The certificate requests can be accepted, deleted, and pended by CA. If the certificate request is accepted, it is issued by CA and exported to users. In addition, certificate revocation list can be created and exported to users.
- Initializing a Registration Authority (RA). Registration Authority is used to be a communication channel between users and CA. The RA is a bridge that exports user requests, certificate revocation requests to the CA, and imports created certificates and CRLs from

the CA. It is public that RA can display CA certificate and valid certificates to users.

These functionalities of OpenCA provide all specifications of digital certificate management. The usage of OpenCA is user-friendly and easy to administrate, so we decided to use OpenCA in our design to manage digital certificates in the model.

4.2.2 Programming Language

There are a few programming languages that can be used in the implementation of the model. The most important factor in the decision of the programming language is the performance of the model. Therefore, the C programming language seemed to be the best choice.

4.2.3 Cryptographic Libraries

In a certificate structure, there is a need for cryptographic libraries to examine certificate fields, serial number, issuer name, subject name, public key, validity date, and others. The most used cryptographic library in secure layers is OpenSSL. OpenSSL is a cryptography toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1) network protocols and related cryptography standards required by them. OpenSSL can be used for creation of RSA, DH and DSA key parameters, creation of X.509

certificates and CRLs, calculation of message digests, encryption and decryption with ciphers, SSL/TLS client and server tests, and handling of S/MIME signed or encrypted mail.

In our authorization system, we need a cryptographic library that can examine the structure of a certificate. The best choice for this operation is OpenSSL. Therefore, we used OpenSSL libraries for validation operation of process certificates and system certificates.

4.2.4 Operating Systems

Linux and its distributions are become popular in these years because of its license mechanism type. We decided to create a prototype of our process authorization mechanism in a Linux operating system with a process creation mechanism in Linux kernel.

4.3 IMPLEMENTATION DETAILS

In this study, we have implemented a prototype of process authorization system that is based on X.509 digital certificates. We try to obtain an authorization framework that is application and implementation independent, may be used as license mechanism, and may be used to prevent process binaries from security threats. The diagrammatic view of the system can be seen in Figure 4.5.

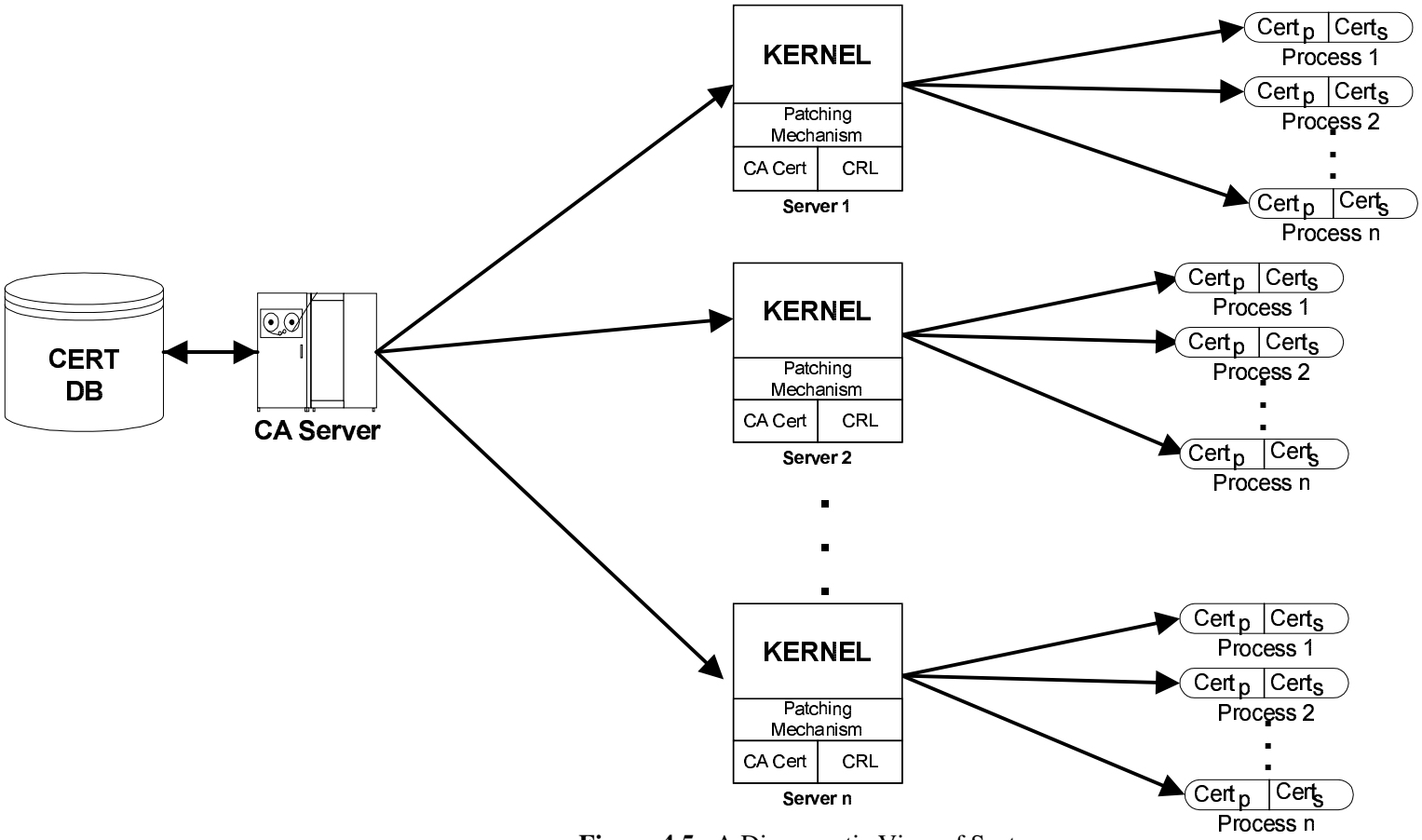

Figure 4.5 : A Diagramatic View of System

In the model, a CA server is used for getting file of CA certificate and file of CRL for validity operations and managing process and system certificates. Each server in the model can be connected to the CA server for getting file of CA certificate and file of CRL for validation. In addition, a certificate is created in the CA server for a process, and it is taken from the CA manually and patched to the process binary for obtaining a trusted process for the system. When this process is being executed, its process and system certificates are controlled by validation library, if these two certificates are valid then the process execution is continued, if one of these certificates is not valid then its execution is broken.

In the validity, operation only PEM formatted certificates are used, and data structure of the system is designed to be suitable for this certificate format.

Furthermore, the operating system kernel controls the creation of each process when the process binary is being executed. The control mechanism provides processes binaries to be checked, if they have certificates at the beginning of each binary or not. If process binaries do not have certificates, then the creation of processes is broken by the kernel create process function.

4.3.1 Data Structures

In the system, the most important data structure is X.509 certificate format for validation operation. Data structures that are defined in OpenSSL libraries are

used for X.509 certificate format. We used a structure called X509_st, which is defined in the x509_vfy.h, for system and process certificates, a structure called X509_store_cts_st, which is defined in x509_vfy.h too, for CA certificates, and a structure called X509_crl_st, which is defined in the x509.h, for CRLs. In addition, a library, asn1.h, is used for ASN1 types of these structures. ASN1 is a specification for how to encode structured data in binary form.

The CRL file will be loaded to x509_crl_st structure. The following fields are stored in this structure:

- x509_crl_info_st *crl: Information about version, signature algorithm, issuer name, last update time, next update time, revocated certificates serial number, revocation dates, extensions and extensions of CRL.
- x509_algor_st *sig_alg: Signature algorithm of the CRL and its parameters.
- ASN1_BIT_STRING *signature: Signature of CRL in binary form.
- int references: Reference number of the CRL.

The main structure of the x509_crl_st can be seen in Figure 4.6.

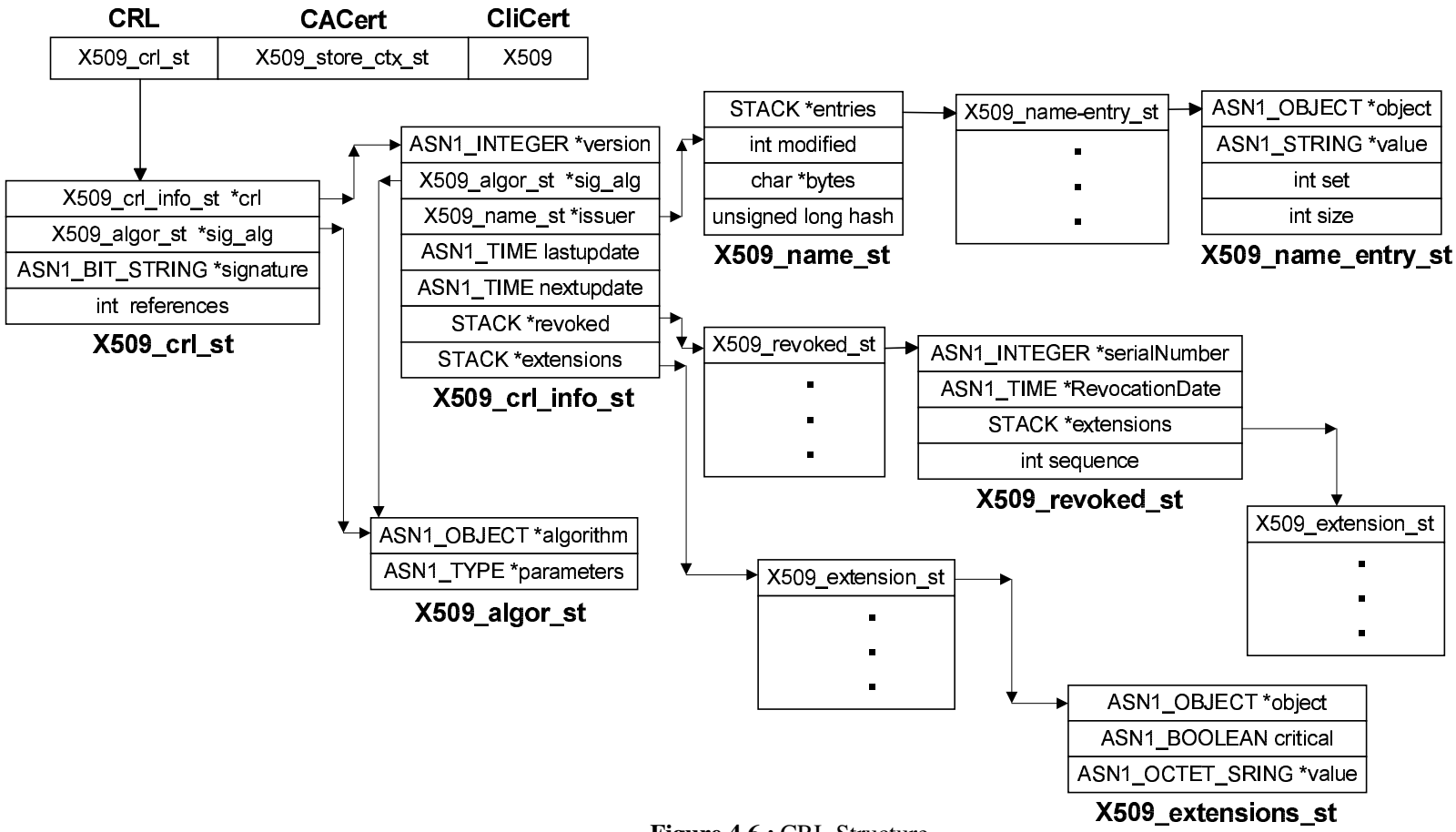

Figure 4.6 : CRL Structure

Certificate of a process or a system is loaded to x509_st structure. In this structure, following fields are stored:

- X509_CINF *cert_info: Information about certificate version, serial number, signature, issuer, subject, validity, public key, issuer identity, subject identity and extensions of certificate.
- X509_ALGOR *sig_alg: Signature algorithm of certificate and its extensions.
- ASN1_BIT_STRING *signature: Signature of certificate.
- int valid: Validity of certificate, this field is used while controlling validity of the certificate.
- ex_pathlen, ex_flags, ex_kusage, ex_xkusage, ex_nscert: Copies of various extension values.
- struct AUTHORITY_KEYID_st *akid: Give information about the key identity of issuer and its serial number.
- X509_CERT_AUX *aux: Private key identity of the certificate and its signature algorithm.

The certificate data structure X509_st can be seen in Figure 4.7.

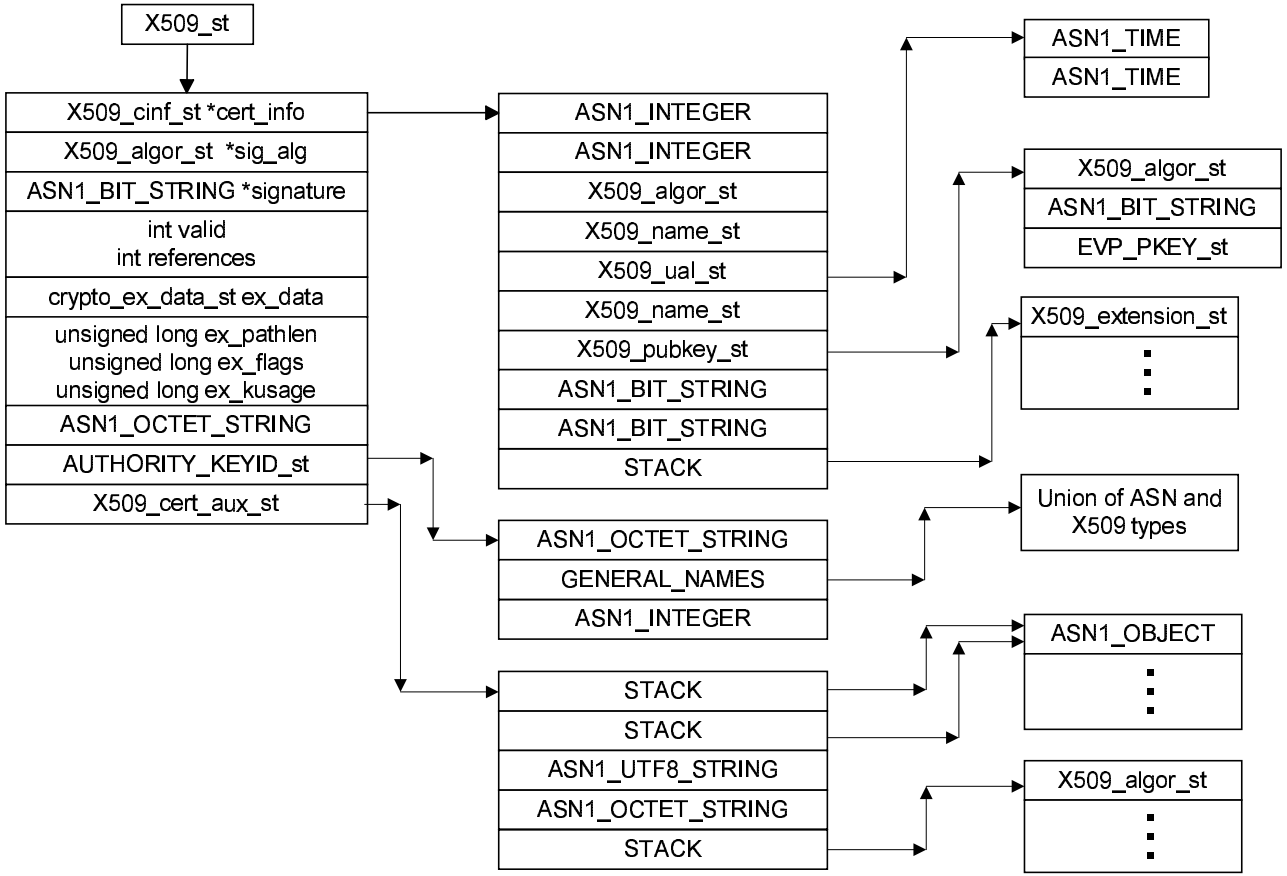

Figure 4.7 : Certificate Structure

The CA certificate is loaded to x509_store_ctx_st structure. In this structure, following fields are stored:

- x509_store_st *ctx: Gives information about objects which hold certificate, CRL and private keys. In addition, it gives information about lookup methods for validity operation, callbacks about verifying certificates and the CA chain of the certificate.
- int current_method: Holds looking up methods for verifying certificates.
- X509 *cert: Used when a validation control of a certificate is being done by CA certificate. The certificate is loaded to this structure.
- STACK_OF(X509) *untrusted: Used for validity operations and holds untrusted certificate list.
- X509 *current_cert: Currently used client certificate for validation operation.
- X509 *current_issuer: Currently being tested as valid issuer certificate.
- X509_CRL *current_crl: Currently used CRL for validation operation.

In Figure 4.8., CA certificate structure, x509_store_ctx_st, can be seen.

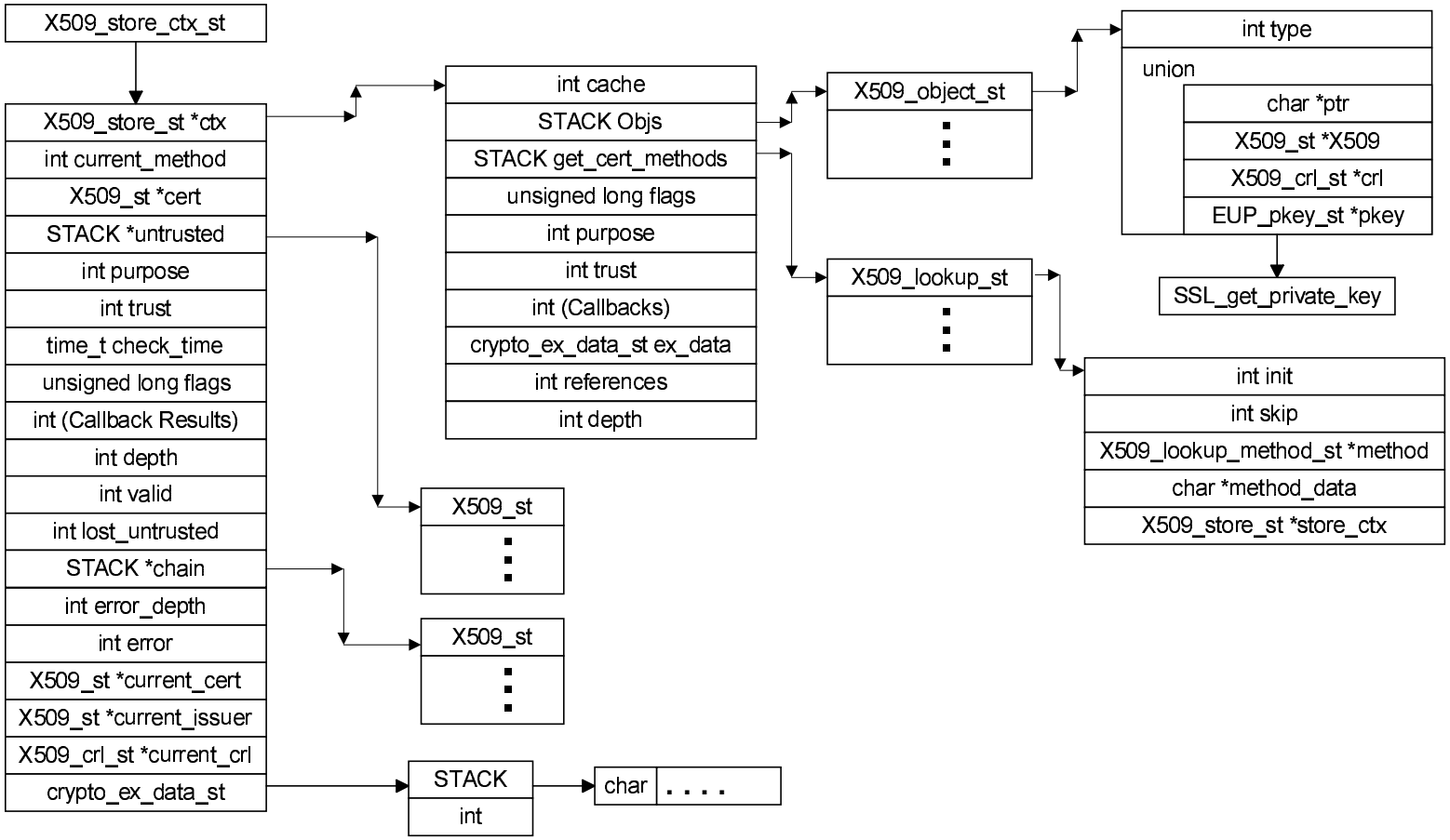

Figure 4.8 : CA Data Structure

4.3.2 Certificate Creation Process

Certificate creation operation is done by manually from web interfaces of OpenCA for processes and systems. These certificates are taken to systems by using web interfaces too. A certificate request is done by giving information about the certificate, when this request is accepted and issued by CA; the requested certificate is taken to the system by manually.

4.3.3 Certificate Validation Process

The certificate validation operation is implemented as a shared library in the system. This library checks the validity of a given client certificate with a CA certificate and CRL of that CA.

The validation operation is started with CA certificate validity check. CA certificate file is loaded to the CA structure and its validity is examined. If CA certificate is a valid certificate then client certificate is loaded. The client certificate may be a system certificate or a process certificate. The validity date of client certificate is checked with the system date. If this check returns a true value, then the issuer of the client certificate is controlled with the CA certificate. If the client certificate check returns a true value, again, the CRL is loaded to the CRL structure. If CRL is valid according to its validity date and issuer name, client certificate serial number searched in that revocation list. If the serial number is not in the revocation list, then the client certificate is said

to be a valid certificate. The validity operation flow chart can be seen in Figure 4.9.

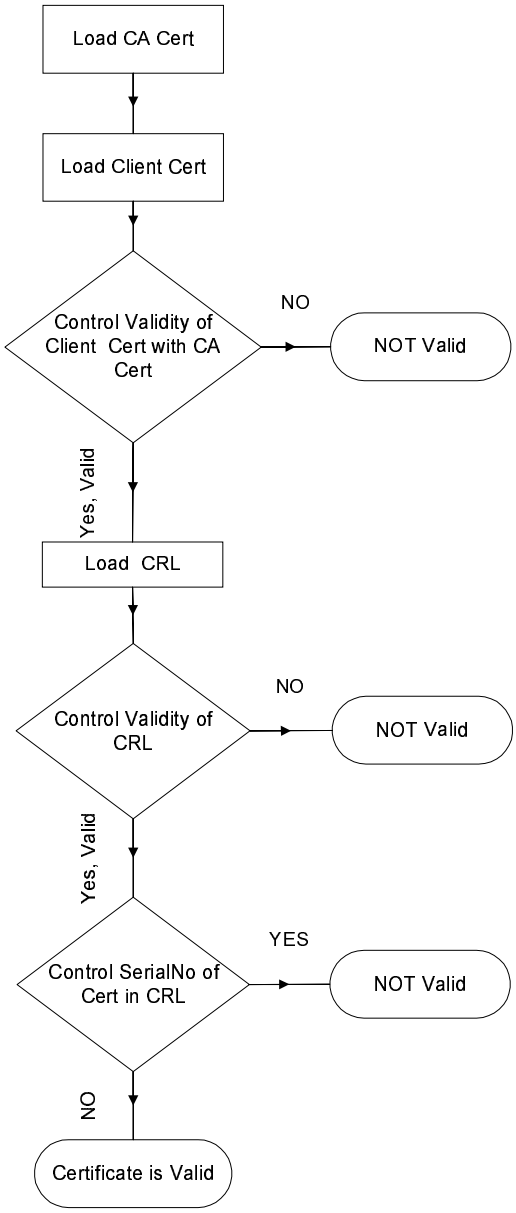

Figure 4.9: Validity Process

This validation operation is compiled as a shared library. This shared library is used by each patched binary in a server. Therefore, when this library was been compiled as a static library, it should be decrease the system performance.

The CertValidity function is designed to be called by any process binaries for validation of its process and system certificate. This function takes the client certificate file path and returns a true value when certificate is valid, a false value when certificate is not valid after the validation operation.

OpenSSL data structures and libraries are used for the validation operation. Validation operation functions for CRLs and CA certificates are defined separately in OpenSSL libraries. In our system, these functions are used. OpenSSL has not supported the validation control of a client certificate using both CA certificate file and CRL file yet. We have implemented a validation operation both using CA certificate file and CRL file.

4.3.4 Binary Creation Process

We need to patch binaries for our process authorization model to have an application independent process authorization system. We have implemented a patching code that stores the process certificate, the system certificate and CertValidity function that is defined in the validation library. The binary form of this code is added to the start of each process binaries.

When a patched binary is started its execution, it executes the patch that stores process certificate and system certificate in temporary files and calls CertValidity function. If CertValidity function returns true values for each certificate, the original process binary is executed. The flow chart of binary creation can be seen in Figure 4.10

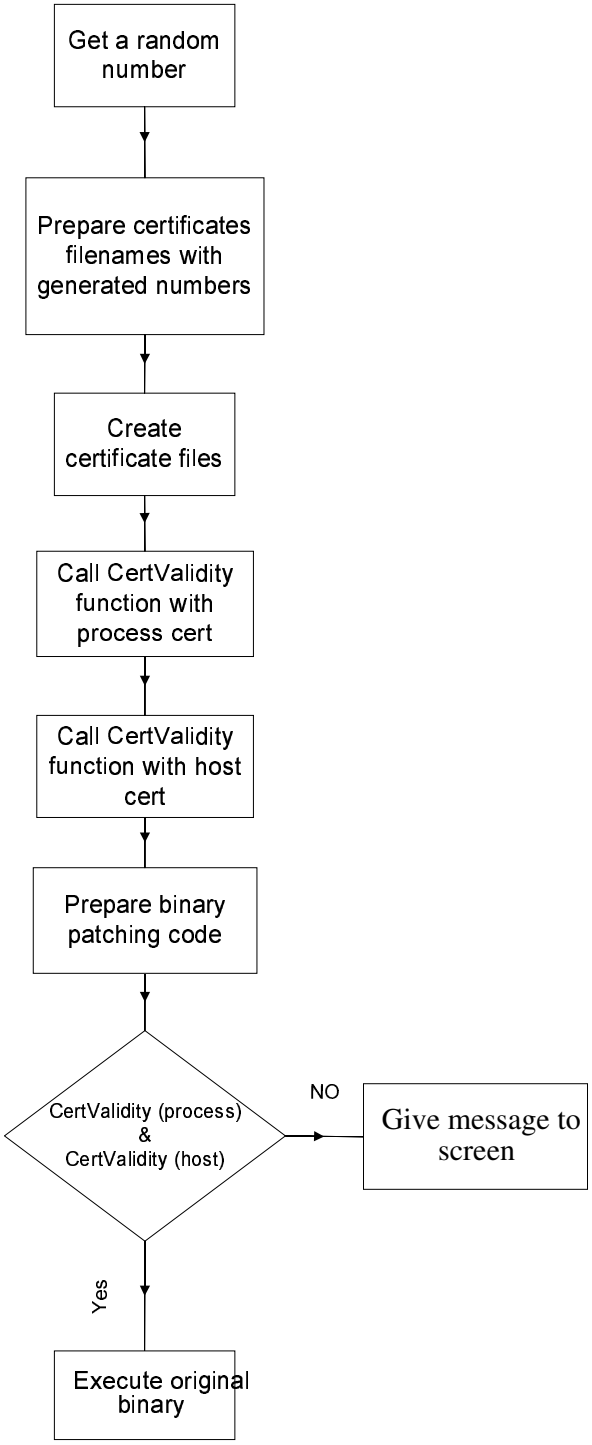

Figure 4.10: Binary Creation Process

4.3.5 Patching Process

The authorization system is required that every process in a system should have a unique process certificate and a system certificate. Therefore, these two certificates should be embedded to the patch code before the patch binary concatenate to the original binary. We have implemented a patching mechanism for patching certificates to the patch of the original binary.

Patching process takes the name of the system certificate file and path, name of the process certificate file and path, name of original binary, which would be patched, file and path, and OpenSSL libraries path in the system. Then the patching process controls files and their existence, prepares data structures of system and process certificates for patching the real patch. The process adds data structures to the real patch and compiles the real patch. After all, the process appends the original binary to the real patch. The flow chart of this operation can be seen in Figure 4.11.

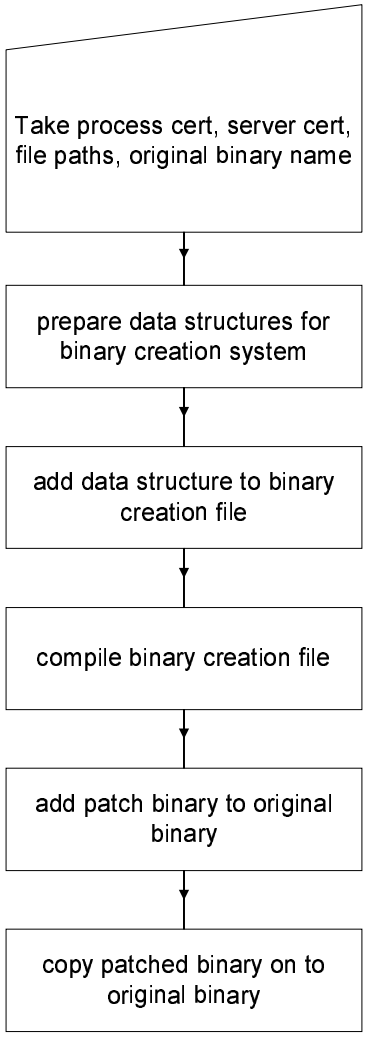

Figure 4.11: Flow Chart of Patching Process

4.3.6 Process Control Mechanism in Kernel

Linux kernel, when a new process is started to execute, uses an exec function for creating process descriptor in the system. There is not really a single function called exec, exec is often used as a generic term to refer to a family

of functions, all which do basically the same thing with different arguments [18].

Creating a new process in Linux takes two steps: forking a new process and executing the process binaries. Fork system call is used to fork a child process as same as itself and takes a process descriptor for new process. Then these process descriptors are used by exec function for initialization and execution of the process.

The do_execve function, which is defined in exec.c file in the fs (file system) directory of the kernel source, is used for all exec functions in Linux. This function reads some identifying information from the executable file into the memory, prepares executable arguments and environments, and then locates a binary handler that agrees to parse the executable [18].

In the process authorization model, we have added some control lines to the exec.c file for controlling executables certificates before taking a binary handler for process execution. Our control mechanism checks the binary of the process for determining if it is patched with certificates for validation. If the binary is patched before to be a "trusted" application, then kernel continues normal operation. If it is not patched, kernel stops the execution of the process. The flow chart of the control mechanism can bee seen in Figure 4.12.

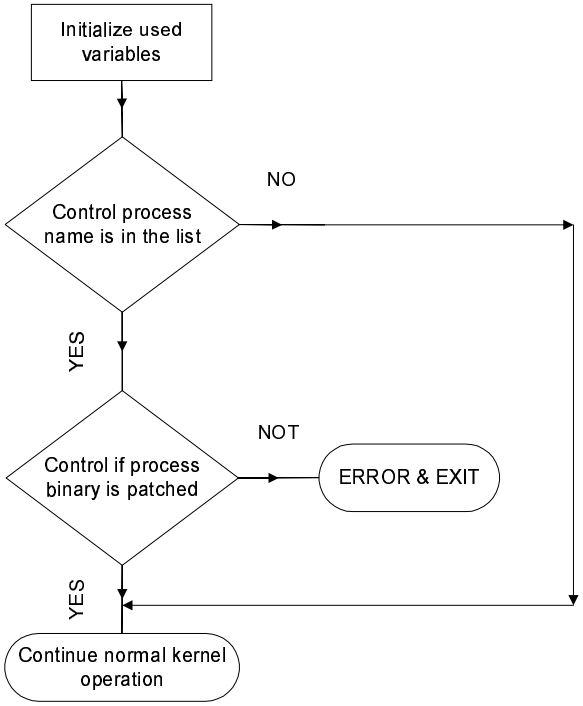

Figure 4.12: Flow Chart of Execution Control in Kernel

4.3.7 The Whole Process

The whole operation that consists of previous sections can be seen in Figure 4.13.

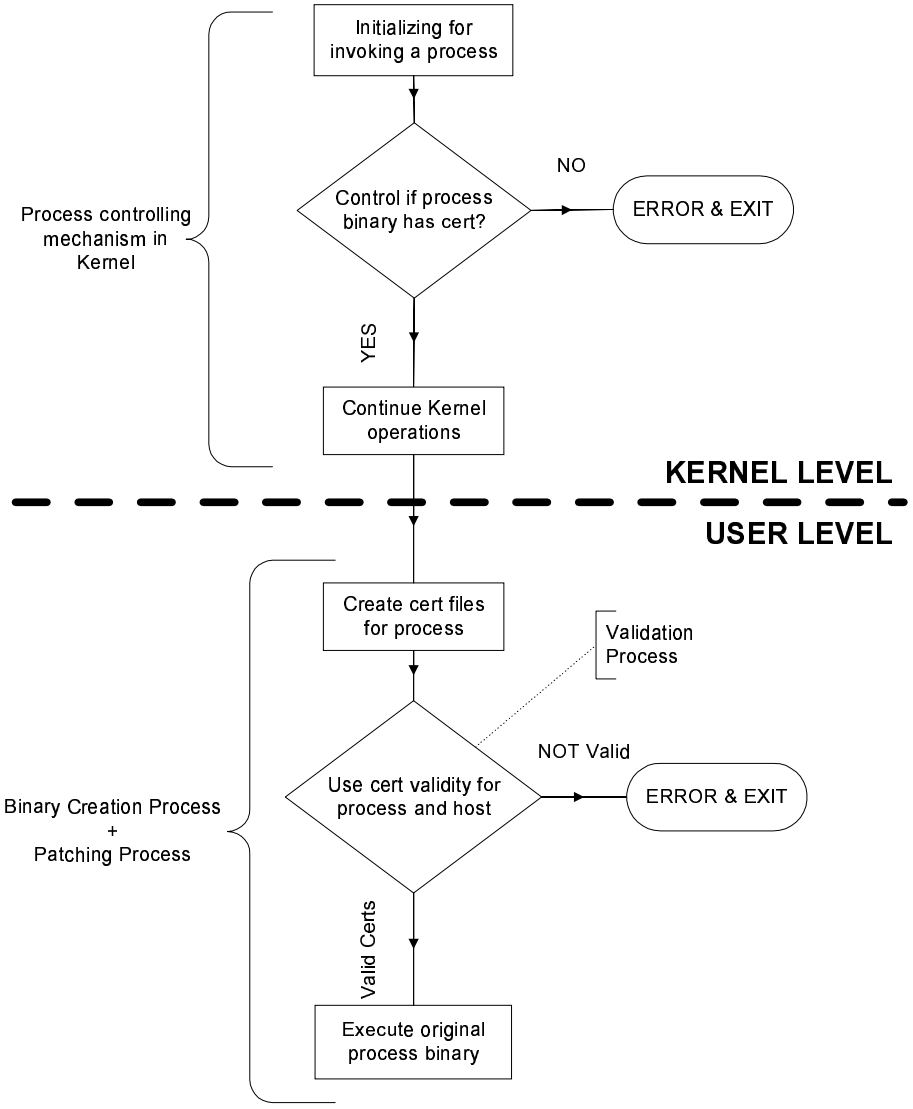

Figure 4.13: Flow Chart of the Whole Process

In Figure 4.13, we thought that we have patched some original binaries in the system and compile kernel according to control these binaries. When a process is being executed, in the kernel level, kernel controls if this binary has been patched before or not. If this binary is not patched, it stops its

new create process operation. If process binary has been patched, kernel continues to create new process operation. In the user level, the patched binary calls the validity process for process and system certificates. If these two certificates are valid then the original binary is executed. If one of two certificates is not valid, the patched binary prints error message to the standard output and stops the execution of the patched binary.

CHAPTER 5

TEST AND EVALUATION

In this chapter we will discuss the implementation performance and tests results.

5.1 PERFORMANCE

Our process authorization approach and implementation details of system design are discussed in the previous chapter. Patching process binaries in a system and adding additional codes to kernel exec.c file is needed by the design of the system.

Execution of the added code in the kernel for every process creation should bring an extra kernel execution load to the system. The added code appends 300 Bytes to the boot image, and uses user and kernel memory while controlling process binaries size.

Each process binary are patched with certificates and certificate validation function. This patch should bring an extra process execution load to the system. However, the patch that is used for adding to the original binaries increased binaries size with 850 Kbytes. This means that each binary in this "trusted" system, should be at least 850 Kbytes size.

5.2 DEVELOPMENT OF SYSTEM

The implemented process authorization system is a prototype of the whole process authorization mechanism. We have designed the implementation of this prototype for proving a certificate based process authorization system, which can form "trusted" applications and "trusted" operating systems, can be obtained. In addition, this system can protect the operating system and processes from security threats as well as can be used like a license mechanism. This prototype can be developed in the future.

In our implementation, operating system platform is Linux, kernel 2.4.21-99. However, other operating systems can be chosen and can be used as an operating system platform. The patch and validation code is written in C programming language, which is a generic developing platform. On the other hand, the added code to the kernel source should be changed according to operating system structure. In Windows operating systems, process creations are controlled by CreateProcess function and _exec functions. The added code should be changed according to windows programming rules.

In our prototype, we have controlled a number of system commands in the kernel that are already patched and we have tested the system with these commands. All processes, executables and applications on a system should be patched one by one and tests should be done in this system.

We used PEM formatted digital certificates in the prototype. This format is an encodable text format that has beginning and end section, which determines the certificate. Therefore, anyone can debug the patched binary and can take PEM formatted certificates of the binary in the system. Then he can use these certificates for his threats. An encryption mechanism can be developed to be used encrypted text certificates before patching mechanism.

Temporary files are used to store process and system certificates in the validation operation. These files are formed randomly by using a generic random number generator. The creation of these files can be changed to be more secure.

As a result, we can say that our implementation proves a PKI based process authorization system, however it can be developed to be platform independent and more secure process authorization system.

5.3 TEST RESULTS

In our implementation, we have changed two main parts of a standard operating system. One of them is the kernel of the operating system for controlling processes if they have certificates or not, the other is process binaries that are executed in this system. However, our model can bring an extra load to the system while patched processes are being executed with validation control.

We have tested our prototype by gnu time function. This function can measure resource usage of a process, which is given to the function as argument. System commands, cat, split, and sort, are tested with a test file which size is 5 Mbytes and is in text format. The test is repeated 20 times for each command and the elapsed real time of executions and CPU times, the kernel and the user mode, are examined. CPU times means that the job has spent in an active state in the CPU within the measured elapsed real time. The results can be seen in Table 5.1.

Table 5.1: Elapsed Real and CPU Times

Times in msec/ Commands	Elapsed Real Time	Kernel Mode CPU Time	User Mode CPU Time
cat Original Binary	16.99	0.56	0.01
cat Patched with Valid Certificates	21.41	0.59	0.34
cat Patched with Not Valid Certificates	1.30	0.05	0.33
sort Original Binary	18.88	0.73	1.80
sort Patched with Valid Certificates	20.91	0.77	2.14
sort Patched with Not Valid Certificates	1.27	0.05	0.33
split Original Binary	0.35	0.31	0.03
split Patched with Valid Certificates	1.83	0.36	0.37
split Patched with Not Valid Certificates	1.27	0.05	0.33

In Table 5.1, we can see the elapsed real time and CPU times of process binaries. In the tests, we have tested all binaries with its original binaries, binaries that are patched with valid certificates, and binaries that are patched with not valid certificates. In addition, we have tested these binaries with an original kernel image and with changed kernel image.

In these tests, we have obtained that there is not significant difference between the changed kernel image and original kernel image according to binaries kernel mode CPU times.

In Table 5.1, rows of commands, which are patched with not valid certificates, show us the validation operation time. The elapsed real time of validation operation is nearly 1.30 msec, the user mode CPU time that is spent in this 1.30 msec is 0.33 msec, and the kernel mode CPU time is 0.05 msec. In addition, these values could be found from difference between patched binaries with valid certificates elapsed time values and original binaries elapsed time values.

CHAPTER 6

CONCLUSION

In this chapter, we give the summary of our process authorization system with its positive and negative points.

6.1 SUMMARY

A process authorization mechanism is needed for controlling all processes, which refer applications, executables, and operating systems. In these years, certificates are being used by security teams in different areas for defining the identity of each item in computing. This brings a solution for "trusted" process authorization mechanisms to gain "trusted" systems.

We combined X.509 certificate creation, binary patching mechanism, certificate validation operation by using OpenSSL libraries, and kernels process execution function to obtain a trusted, application, implementation, independent process authorization mechanism.

Our goal was implemented a certificate based process authorization mechanism to prevent our systems according to security threats and illegal usage of our applications. At the end of the implementation, we obtain a so-called "trusted" process authorization mechanism, which is application independent. In addition, this process authorization implementation provides goals in an authorization framework, which is discussed in Chapter 2.

Finally, having "trusted" applications and "trusted" operating systems is possible by the design and implementation of PKI based process authorization system.

6.2 FUTURE WORK

A PKI based process authorization mechanism is implemented in this thesis. It provides goals of an authorization framework, however the mechanism can be developed to be more secure and to be used in different platforms easily.

In the previous chapter, we discussed the development of the process authorization system. The system will be developed by using different operating system platforms for controlling process executions in the kernel level. In addition, the public key infrastructure implementation can be changed to be more administrative in the future.

In addition, used certificate types can be changed to obtain a more secure process authorization system. Also, process and system certificates can be encrypted before added to process binaries.

As a result, our system provides the need of a trusted process authorization system, but it can be developed in the future to be more effective and more secure.

REFERENCES

[1] Corner, D. E., Internetworking with TCP/IP Vol: 1 Principles, Protocols and Architecture, Prentice Hall, Second Edition, 1990

[2] Landwehr, C. E., Golschlag D. M., Security Issues in Networks with Internet Access, Proceedings of IEEE, Vol.85, No.12, 1997

[3] Xenitellis, S., The Open-source PKI Book, 2.4.6 Edition, 2000

[4] OpenCA Web Page, http://www.openca.org, September, 2003

[5] Pfleeger, C. P., Security in Computing, Prentice Hall, Second Edition, 1997

[6] Wasley, D. L., PKI Based Authorization, Draft, UCOP, Rev. 1.0, 2002

[7] Grek, C., Directory Enabled Security, SIMC, Authorization, Dascom Inc., 1998

[8] Intellectual Property Protection in Cyberspace: Towards a New Consensus, Information Technology Association of America, ITAA Discussion Paper, 1998

[9] Auro, T., Software License Management With Smart Cards, USENIX Workshop on Smartcard Technology Chicago, Illinois, USA, May, 1999

[10] Stallings, W., Network Security Essentials: Applications and Standards, Prentice Hall, 2000

[11] Williams, M. A., Anti-Trojan and Trojan Detection with In-Kernel Digital Signature Testing of Executables, Security Software Engineering, NetXSecure NZ Limited, April, 2002

[12] Auro, T., Distributed Access Rights Management with Delegation Certificates, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Springer, 1999

[13] Introduction to Public-Key Cryptography, Netscape Developers Web Page, April, 2003
[14] Edwards, A., Jaegar, T., Zhang, X., Verifying Authorization Hook Placement for the Linux Security Modules Framework, IBM T.J. Watson Research Center, November, 2001

[15] Internet X.509 Public Key Infrastructure, RFC 3280, IETF, Network Working Group, April, 2002

[16] Atreya, M., Introduction to PKCS Standards, RSA Security Inc., http://www.rsasecurity.com/solutions/developers/whitepapers.html, October, 2003

[17] Crowley, C., Operating Systems, A Design-Oriented Approach, Irwin Book Team, 1997

[18] Maxwell, S., Linux Core Kernel Commentary, Coriolis Open Press, Part II, 1999

[19] Hunt, G., Brubacher, D., Detours: Binary Interception of Win32 Functions, Proceedings of the 3rd USENIX Windows NT Symposium, Seattle, WA, July, 1999

[20] Marchesini, J., Zhao, M., Keyjacking 101: Why Storing Keys in Your Browser Isn' t Safe, Department of Computer Science, Dartmouth College, March, 2002

[21] Internet X.509 Public Key Infrastructure, Certificate and CRL Profile, RFC, 2459, IETF, Network Working Group, January, 1999

[22] Wright, C., Cowan, C., Smalley, S., Morris, J., Hartman, G., Linux Security Modules: General Security Support for the Linux Kernel, Proceedings of the 11th USENIX Security Symposium, August, 2002

[23] Thompson, M. R., Essiari, A., Mudumbai, S., Certificate-Based Authorization Policy in a PKI Environment, ACM 1073-0516/01/0300-0034, 2001