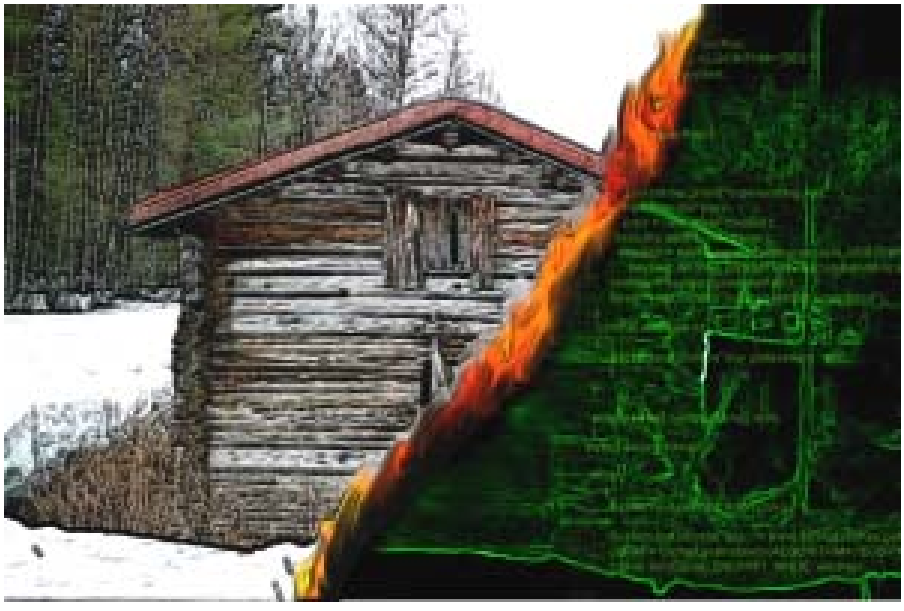
Jarkko Holappa

Security threats and requirements for Java-based applications in the networked home environment



TECHNICAL RESEARCH CENTRE OF FINLAND ESPOO 2001

VTT PUBLICATIONS 444

Security threats and requirements for Java-based applications in the networked home environment

Jarkko Holappa

VTT Electronics

TECHNICAL RESEARCH CENTRE OF FINLAND
ESPOO 2001

ISBN 951–38–5865–0 (soft back ed.)
ISSN 1235–0621 (soft back ed.)

ISBN 951–38–5866–9 (URL:http://www.inf.vtt.fi/pdf/)
ISSN 1455–0857 (URL:http://www.inf.vtt.fi/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

Valtion teknillinen tutkimuskeskus (VTT), Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde (09) 4561, faksi (09) 456 4374

Statens tekniska forskningscentral (VTT), Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel (09) 4561, fax (09) 456 4374

Technical Research Centre of Finland (VTT), Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
phone internat. + 358 9 4561, fax + 358 9 456 4374

VTT Elektroniikka, Sulautetut ohjelmistot, Kaitoväylä 1, PL 1100, 90571 OULU
puh. vaihde (08) 551 2111, faksi (08) 551 2320

VTT Elektronik, Inbyggd programvara, Kaitoväylä 1, PB 1100, 90571 ULEÅBORG
tel. växel (08) 551 2111, fax (08) 551 2320

VTT Electronics, Embedded Software, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland
phone internat. + 358 8 551 2111, fax + 358 8 551 2320

Technical editing Maini Manninen

Otamedia Oy, Espoo 2001

Holappa, Jarkko. Security threats and requirements for Java-based applications in the networked home environment. Espoo 2001. Technical Research Centre of Finland, VTT Publications 444. 116 p.

Keywords public key infrastructure, security policy, Java, distributed software, protection profile

Abstract

This work presents the networked home environment from the security point of view. Threats, technologies and the special characteristics of the users are examined. 'Common Criteria' is used in this thesis as a security evaluation criterion to construct a protection profile for the software distribution platform of a networked home environment. 'Protection profile' describes the target of the evaluation - the networked home environment and its security environment, along with access control and information flow policies. This environment sets the context for the security requirements that are established as a result of this thesis to counter the threats that are also identified in the protection profile as a part of the security environment.

Java is a relatively promising platform for the networked software because of its security model, which has evolved since the first versions of Java. Java's application programming interfaces provide support for widely used cryptographic techniques and public key infrastructure frameworks, including the X.509 authentication framework. Java's security features are applied to the software distribution platform developed at VTT Electronics. The security framework for the platform is developed and presented in this work.

'Home', as a distributed computing environment, presents many new issues when compared to typical corporate office networks. Users are very heterogeneous and their needs differ from one to another. The requirements specification must be done with care, and by using knowledge of the system and existing security techniques to develop a system that provides adequate confidentiality, integrity and availability for its users.

Preface

The work done on this thesis was carried out in the ITEA VHE project, which is part of a large EU programme called EUREKA. The estimated budget for ITEA (Information Technology for European Advancement) is 3.2 billion euros - 20,000 person years - and it focuses on strengthening European software technology competence.

I would like to express my deepest gratitude to Mr. Hannu Rytilä and Research professor Eila Niemelä, from VTT Electronics, for their guidance, not only during the thesis but also throughout the time I have worked at VTT Electronics, and also to Professor Juha Röning from the University of Oulu, who worked as the supervisor, for guiding me through the writing process. My second supervisor, Professor Tino Pyssysalo, also deserves my appreciation. I am grateful to reviewers of this publication, Lic. Tech. Kimmo Takanen from LM Ericsson and Professor Veikko Seppänen from University of Oulu, who provided me with valuable advices to the world of scientific writing.

Special thanks to Mr. Rauli Kaksonen from VTT Electronics for his security-aware comments and pointers.

Last, but not least, I would like to thank my friends, near and far, and my family for filling my student days with all the good things.

Oulu, 29th August 2001

Jarkko Holappa

Contents

Abstract 3

Preface 4

List of symbols 7

1. Introduction 9

2. Security Technologies 12
   2.1 Introduction and terminology 12
   2.2 Security threats in a networked environment 14
      2.2.1 Impersonating a user or system 15
      2.2.2 Eavesdropping 16
      2.2.3 Denial of Service 17
      2.2.4 Packet replay 19
      2.2.5 Packet modification 19
   2.3 General guidelines for designing trusted computer systems 21
      2.3.1 Department of Defense Trusted Computer System Evaluation Criteria 21
      2.3.2 Trusted Network Interpretation of the TCSEC 23
      2.3.3 Common Criteria for Information Technology Security Evaluation 23
   2.4 Cryptographic protocols and algorithms 27
   2.5 Data security: Three areas of concern 32
      2.5.1 Confidentiality 33
      2.5.2 Integrity 34
      2.5.3 Availability 35
   2.6 Authentication and authorization of a user 36
      2.6.1 X.509 Authentication Service 36
      2.6.2 Secure socket layer (SSL) and transport layer security (TLS) ... 47
   2.7 Auditing 49

3. Java Technology and security 50
   3.1 Introduction 50
   3.2 The Java virtual machine 51
      3.2.1 Life cycle of the Java virtual machine 51
      3.2.2 The architecture of the Java virtual machine 52
   3.3 Java's built-in security model 54

3.3.1 Evolution of the sandbox model 55
3.3.2 Secure class loading and verification 58
3.3.3. JVM's responsibility in Java security 63
3.3.4 The security manager 64
3.3.5 The protection domain and access control mechanism 64
3.4 Security management in Java 66
3.4.1 Code signing and authentication 66
3.4.2 JDK's security-related tools 67
4. Middleware protection profile for the networked home environment 69
4.1. Protection Profile (PP) Overview 69
4.2. Target of evaluation (TOE) description 69
4.3. TOE Security Environment 71
4.3.1. Assumptions 72
4.3.2. Threats 72
4.3.3 Security policies 74
4.4. Security Objectives 75
4.5. Security Requirements 77
Auditable event 79
4.6. Rationale 88
4.6.1 Security objective rationale 88
4.7.2 Security functional requirement rationale 89
5. LONTONEXTG Distribution platform 97
5.1 Introduction to LONTONEXTG environment 97
5.2 Example service 100
5.3 Security framework of the system 101
6. Discussion 105
6.1 User roles 106
6.2 Functional requirements and security policy definition 107
6.3 Java as the implementation platform 109
6.4 Characteristics of PKI-based security services 110
7. Conclusions 112
References 114

List of symbols

3GPP™	3rd Generation Partnership Project
AES	Advanced Encryption Standard
API	Application Programming Interface
CA	Certification Authority
CC	Common Criteria
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSRC	Computer Security Resource Center
CRL	Certificate Revocation List
CTCPEC	Canadian Trusted Computer Product Evaluation Criteria
DES	Digital Encryption Standard
DoD	Department of Defense
DSA	Digital Signature Algorithm
EAL	Evaluation Assurance Level
GSM	Global System for Mobile Communications
IDEA	International Data Encryption Algorithm
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	The International Organization for Standardization
ITSEC	European Information Technology Security Evaluation Criteria
ITU	International Telecommunications Union
JAR	Java ARchive
JDK	Java Development Kit
JVM	Java Virtual Machine
LON	Local Operating Network
MAC	Message Authentication Code
MT	Mobile Terminal

NCSC	National Computer Security Center
NIST	National Institute of Standards and Technology
NSA	National Security Agency
NTCB	Network Trusted Computing Base
OS	Operating System
OSI	Open Systems Interconnection
PDA	Personal Digital Assistant
PP	Protection Profile
RC4, RC5	Rivest Cipher (encryption algorithms)
RMI	Remote Method Invocation
RSA	Rivest, Shamir and Adleman (encryption algorithm)
S/MIME	Secure/Multipurpose Internet Mail Extension
SET	Secure Electronic Transaction
SHA	Secure Hash Algorithm
SPI	Service Provider Interface
SSL	Secure Socket Layer
ST	Security Target
TCB	Trusted Computing Base
TCP	Transmission Control Protocol
TCSEC	Trusted Computer System Evaluation Criteria
TNI	The Trusted Network Interpretation
TOE	Target Of Evaluation
TSC	TSF Scope of Control
TSF	TOE Security Function
TSL	Transport Layer Security
TSP	TOE security policy
UI	User Interface
UML	Unified Modelling Language
VHE	Virtual Home Environment
X.509	Authentication Framework, ITU-T recommendation

1. Introduction

The evolution of information technology impacts not only on our working habits but also on our everyday life at home. Many daily routines can be done remotely by using, for example, a computer with a network connection to make on-line payments and purchases from a web store. In addition to these, many visions of intelligent homes of the future have been presented. Such a home has household appliances (e.g. sauna stove, refrigerator and heating system) that are connected so that the homeowner can operate them remotely, using, for example, a mobile terminal as a user interface.

The home environment consists of those appliances that are networked to enable flexible use. The diversity of the underlying network technologies is substantial, including many wireless and wired networks. The home also presents a fairly novel environment for the software developer because the diversity of users and their capabilities, including children, adults and elderly people, is broad. Taking this into account, ease of use is an essential requirement.

Ease, location and transparent use of household appliances, along with multiple user interfaces, terminals and network connections, raises many scenarios with regard to serious security threats that must be investigated and solved in order to devise a reliable and secure home environment without sacrificing the convenience of the home. The diversity of services leads to different emphases in security requirements because the characteristics of transferred information varies from one service to another. Confidentiality and integrity of information is crucial when the service is, for example, an electronic commerce application or on-line payment system. Using appliances (i.e. services that appliances provide) with a mobile terminal from a car or somewhere else outside the network appoints, for its part, new requirements for information availability and quality, including security requirements.

A great amount of software is being developed using pre-made components, and those components must be able to communicate with each other in a secure way without exceeding their authority - i.e. following access control policies enforced by the system. In a networked environment these components are also mobile: they move within the network carrying out the tasks they were aimed at. Here, the integrity and authenticity of the mobile code is essential. In other

words, it is vital that the code is not tampered with while being transferred and also that the code is coming from a benevolent source.

Use of the service must be restricted to authorized users ('user' denotes both software components and humans) only and the authenticity of the user must be verified in a reliable way before granting user access to services located in the home network. Without ensuring the user's authenticity (to a service or to the network) many threatening scenarios can occur - for example, an ignorant neighbour living just behind the wall could register to the wrong wireless network and turn the wrong sauna stove on. In addition to this, numerous intentional attacks must be prevented.

This thesis concentrates on the security requirements brought by a networked home environment. Some of the most commonly used techniques and protocols are presented to give an overview of networked security and the threats they are meant to combat. The security requirements for a networked home environment are constructed by examining the threats and objectives of such an environment in more detail. The Protection Profile for the networked home environment constructed in this work describes the networked home environment, its user roles and security domains, as well as security threats and objectives.

Before gathering these subjects, four questions can be raised to shape the research problem behind this work:

1. What is the security environment that the home network presents?

2. Is it possible to form a shared security policy or policies for all types of services?

3. Who are users of the system and what are their roles?

4. What are the most important security requirements that most services of the distribution platform require and, thus, are able to share?

Functional requirements are then derived from these system characteristics to find answers to the above questions. In addition, the protection profile defines the access control and information flow policies developed in this work. In

Chapter 5, the protection profile is applied to Java implementation of the distribution platform developed at VTT Electronics for the use of home middleware research.

2. Security Technologies

2.1 Introduction and terminology

The Merriam-Webster dictionary [1, p. 2053] defines security as "the quality or state of being secure" or "freedom from danger". While being the most general definitions of security, they also apply to computer system security. Security can be roughly divided into three areas of concern, which must be satisfied in order to consider a computer system as safe:

1. Confidentiality,
2. Integrity,
3. Availability.

This holds true for simple systems as well as more complex, distributed and networked systems. This chapter will focus on the requirements of a secure computer system, especially in a networked environment. The special characteristics of a networked environment will be examined and, equally, general guidelines for designing trusted computer systems will be presented. *Common Criteria* for information technology security evaluation [2] will be presented in more detail. Finally, three security areas of concern will be studied, reviewing the most common cryptographic techniques and algorithms used in the implementation of an adequate level of security.

The Glossary of Computer Security Terms [3] gives exact definitions for most terms used in this document. These terms are described in Table 1.

Table 1. Definition of terms.

Term	**Definition**
Access Control	"The process of limiting access to the resources of a system only to authorized programs, processes, or other systems (in a network. Synonymous with controlled access and limited access."
Attack	"The act of trying to bypass security controls on a system. An attack may be active, resulting in the alteration of data; or passive, resulting in the release of data. Note: The fact that an attack is made does not necessarily mean that it will succeed. The degree of success depends on the vulnerability of the system or activity and the effectiveness of existing countermeasures."
Authenticate	"(1) To verify the identity of a user, device, or other entity in a computer system, often as a prerequisite to allowing access to resources in a system. (2) To verify the integrity of data that have been stored, transmitted, or otherwise exposed to possible unauthorized modification."
Authorization	"The granting of access rights to a user, program, or process."
Availability of data	"The state when data are in the place needed by the user, at the time the user needs them, and in the form needed by the user."
Confidentiality	"The concept of holding sensitive data in confidence, limited to an appropriate set of individuals or organizations."
Cryptography	"The principles, means and methods for rendering information unintelligible, and for restoring encrypted information to intelligible form."
Data integrity	"The property that data meet an a priori expectation of quality."
Data security	"The protection of data from unauthorized (accidental or intentional) modification, destruction, or disclosure."

Denial of Service	"Any action or series of actions that prevent any part of a system from functioning in accordance with its intended purpose. This includes any action that causes unauthorized destruction, modification, or delay of service. Synonymous with interdiction."
Protocol	"A set of rules and formats, semantic and syntactic, that permits entities to exchange information."
Spoofing	"An attempt to gain access to a system by posing as an authorized user. Synonymous with impersonating, masquerading or mimicking."
Tampering	"An unauthorized modification that alters the proper functioning of an equipment or system in a manner that degrades the security or functionality it provides."
Threat	"Any circumstance or event with the potential to cause harm to a system in the form of destruction, disclosure, modification of data, and/or denial of service."

2.2 Security threats in a networked environment

Maintaining security is relatively easy with a standalone system, but when it is connected to a public network, many kinds of security threats will come up. In this case, 'public network' can generally be understood as a network with many users, possibly containing untrusted parts. The term 'untrusted part' refers to untrusted (unknown) users and unreliable transfer media. This discussion is valid in many kinds of networks, including the Internet, public telephone network and many kinds of wireless networks. Transmission errors, error detection and error correction fall outside the scope of this discussion because they are not considered a deliberate attempt to cause defects in data transfer. Figure 1 depicts a successful data transmission situation where no defects, active or passive, occur. The five most common threats can be pointed out [5, pp. 79–83] in a networked environment as follows:

1. Impersonating a user or system
2. Eavesdropping
3. Denial of service

4. Packet replay
5. Packet modification.

What is common to these threats is that they are all active attempts to harm security in the system and get, modify or destroy classified information. Each of them is now examined in more detail.

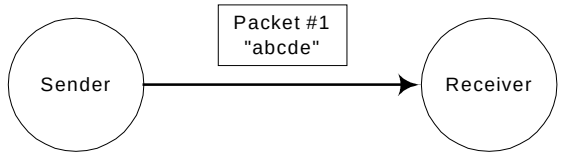


Figure 1. Successful data transmission.

2.2.1 Impersonating a user or system

The most common way of identifying a user is to use account names and passwords, or, for example, biometric checks [5]. To a malicious user, security holes in these identification practises offer ways of impersonating a legally authorized user of the system. The most reliable identification methods use biometric checks, or, for example, physical keys, and these are less likely to become misused. Networked systems enable many new possibilities for impersonating, compared to standalone systems [5]:

- A malicious user can have access to a wide range of systems over a large geographic area.

- Numerous methods for guessing passwords are available, from "through trial and error" to monitoring activity of the system and electronic eavesdropping. Monitoring activity of the system and impersonating a user is used when such attack is less likely to be detected.

'Impersonating' is described in Figure 2, where the malicious user acts as a legal receiver for the packet and steals it. The destined receiver does not get a copy of this packet.

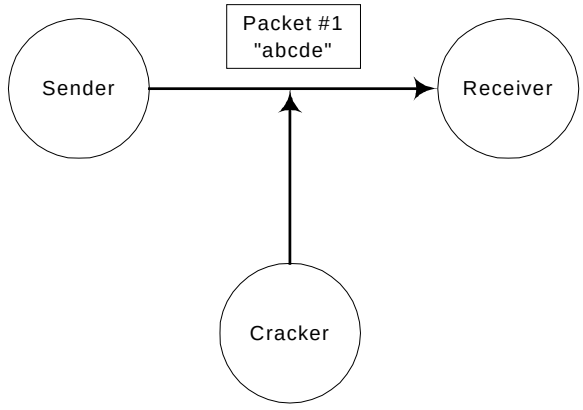


Figure 2. Impersonating a user or system.

2.2.2 Eavesdropping

The motive for eavesdropping is to gain sensitive information (user accounts, passwords, and data). Eavesdropping is depicted in Figure 3. Eavesdropping can be carried out [5] using wiretapping, by radio (especially wireless networks) and via auxiliary ports on terminals. Eavesdropping is also possible using network-monitoring software that keeps track of the packets sent over the network. When network traffic is not encrypted, eavesdropping offers a very powerful means for a malicious user to gather the information necessary to get access to the desired system. It is also quite difficult to detect a malicious user in the act of eavesdropping. Very often, eavesdropping offers an easy route to system resources and, thereby, leads to other critical security violations, such as the denial of service attacks.

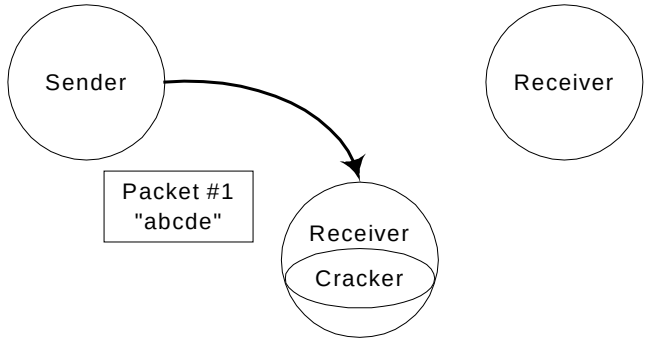


Figure 3. Eavesdropping.

2.2.3 Denial of Service

Networked, multi-user and multi-tasking systems are exposed to denial of service attacks [5]. 'Denial of service' can be considered as both intentional and unintentional attacks on a system's availability [6, pp. 159–170]. The denial of service attack, which is always intentional, is carried out by taking up shared resources to the extent that other users become unable to use the system, or degrading a resource so that it is less valuable to users. Shared resources are comprised of other processes, shared files, disk space, percentage of CPU, modems, etc. The Denial of service attacks can be divided into five categories, as follows [6]:

1. Destruction
2. Process degration
3. Storage degration
4. Process shutdown
5. System shutdown.

These types are described in Figure 4, which also shows the general procedure of the denial service attack. The malicious user starts the denial of service attack

by exploiting vulnerabilities and then either obtains unauthorized access to system resources (processes, etc) or uses processes in an unauthorized way. The attack is completed by using some means of destroying files or degrading system resources to cause the shutdown of a process or a system.

Systems must be protected against denial of service attacks without denying the access of legitimate users. However, this condition is very often hard to satisfy. Restricting access to critical system resources will cut down the possibility of denial of service attacks [6].

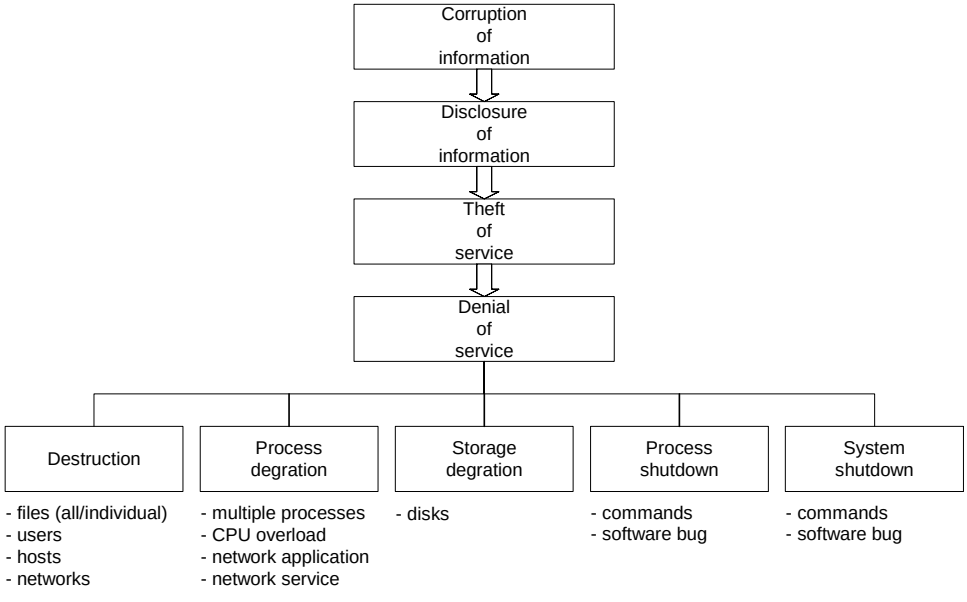


Figure 4. The general procedure of denial of service attack.

2.2.4 Packet replay

If a malicious user needs to obtain authentication sequences, he can use packet replay to record and re-transmit the packet to the network, described in Figure 5. With this procedure, the intruder is able to replay an authentication sequence to gain access to the system. Packet replay can be detected using packet time stamping and packet sequenc counting [5], but it is relatively hard to prevent.

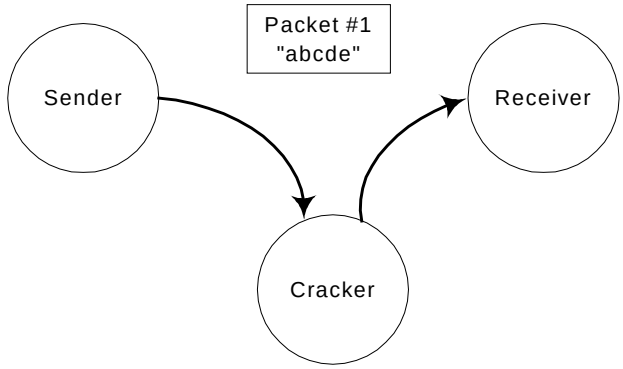


Figure 5. Packet replay.

2.2.5 Packet modification

In addition to the most obvious definition of the term (depicted in Figure 6a), packet modification can also be considered as destruction of information (Figure 6b). Packet modification always requires interception and represents a significant integrity threat for data transmission [5]. However, this threat can be detected using encryption and secure hash codes (message digest) to ensure the validity of information. These issues are examined in more detail in Chapter 2.4.

One special case of packet modification is fabrication, in which the malicious user creates counterfeit packets for the receiver (Figure 6c.). From the receiver's point of view, this can also be considered as impersonation due to the malicious user acting as a legitimate sender of the packet.

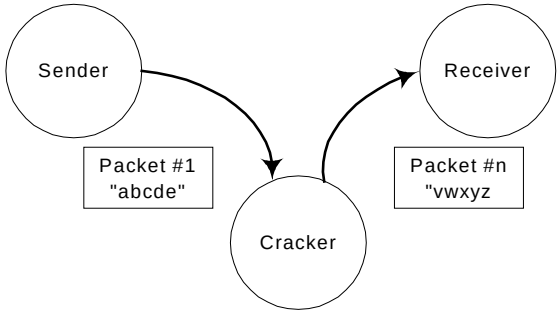


Figure 6a. Packet modification.

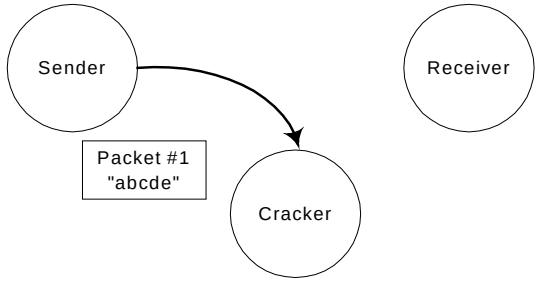


Figure 6b. Packet destruction.

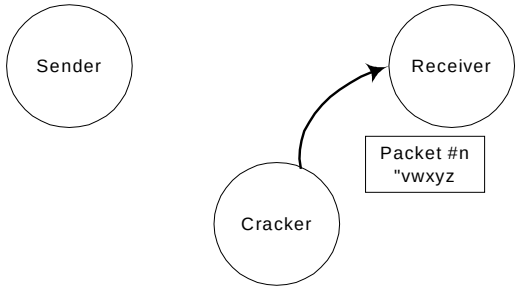


Figure 6c. Packet fabrication.

2.3 General guidelines for designing trusted computer systems

This Chapter deals with security evaluation criteria and gives an overview of the three major criteria: The Orange Book [7], The Red Book [8] and The Common Criteria [2].

2.3.1 Department of Defense Trusted Computer System Evaluation Criteria

The history of Trusted Computer Security Evaluation Criteria (TCSEC, The Orange Book) goes back to 1967, when work towards security evaluation guidelines started. The Orange Book was the first widely accepted evaluation criteria. This guideline was directed towards governmental - i.e. national - security, but the authors also intended to create a more general document for security evaluation. The Orange Book is meant to provide [9, pp. 79–83]:

- A measurement for a user to evaluate the degree of the trust that can be placed in a computer security system,
- Guidance for manufacturers of computer security systems and a basis for specifying security requirements in acquisition specifications.

Security evaluation focuses on the security-relevant part of the system, which TCSEC refers to as the Trusted Computing Base (TCB). The access control policies of TCSEC are taken from the Bell-LaPadula model - i.e. discretionary access control and mandatory access control based on a lattice of security labels, which represents the security level of an object. The Bell-LaPadula model is presented in Chapter 2.5.1. From a basic definition of security, the Orange Book derives the following six fundamental security requirements [7]:

1. *Security Policy*: access control policies expressed in terms of subjects and objects.
2. *Marking of objects*: access control labels associated with objects specify the sensitivity of objects.

3. *Identification of subjects*: individual subjects must be identified and authenticated
4. *Accountability*: Audit information must be selectively kept and protected.
5. *Assurance*: Operational assurance (security architecture issues) and lifecycle assurance (design methodology, testing and configuration management).
6. *Continuous protection*: Security mechanisms must be continuously protected against tampering and/or unauthorized changes.

The Orange book uses these requirements to define four security divisions and seven security classes. The four divisions and their classes are:

1. **D** Minimal protection
2. **C** Discretionary protection (’need to know’)
    - **C1** *Discretionary security protection:* Co-operating users process data at the same level of integrity.
    - **C2** *Controlled access protection:* Users are individually accountable for their actions via discretionary access control at the granularity of a single user.
3. **B** Mandatory protection (based on labels)
    - **B1** *Labelled security protection:* Each subject and object has labels, constructed from hierarchical classification levels.
    - **B2** *Structured protection:* Increased assurance mainly by adding requirements to the design of the system.
    - **B3** *Security domains:* A security administrator is supported; trusted recovery after a failure must be facilitated.
4. **A** Verified protection
    - **A1** *Verified design:* Functionally equal to B3. Achieves the highest assurance level through the use of formal specification of policy and system. Requires consistency proofs between model and *formal top level specification.*

2.3.2 Trusted Network Interpretation of the TCSEC

The Trusted Network Interpretation (TNI, The Red Book) addresses network security with the concepts and terminology introduced in the Orange Book. The Red Book has to address issues that are not present in the Orange Book - for example, new security problems that arise due to:

- The vulnerability of the communication paths.
- Concurrent and asynchronous operation of the network components.

There are some considerable limitations on the TNI - for example, only centralised networks (*single trusted systems*) with single accreditation authority, policy and *network trusted computing base* (NTCB) are considered by the Red Book [9]. Keeping this in mind, the Red Book should be treated as a link between the Orange Book and new criteria, like Common Criteria, which have been proposed in later years.

2.3.3 Common Criteria for Information Technology Security Evaluation

The Common Criteria (CC) is result of efforts to develop criteria for evaluating IT security that are widely used within the international community. It is an alignment and development of a number of source criteria: the European, US and Canadian criteria (ITSEC, TCSEC and CTCPEC respectively) [2]. The CC is intended to resolve the conceptual and technical differences between the source criteria and is a contribution to the development of an international standard. The security framework of Common Criteria uses the hierarchical framework of security concepts and terminology depicted in Figure 7.

Common Criteria, the current version of which is version 2.1, has three parts. Each part, and its purpose and three interested parties, are briefly presented in Table 2. To fully understand Table 2, some key concepts of CC must be defined:

- *The Target of Evaluation (TOE)*

  TOE presents that part of the system which is subject to evaluation.

- *Security Target (ST)*

  ST contains the IT security objectives and requirements of a specific identified TOE - i.e. the IT product - and defines the functional and assurance measures to meet the stated requirements.

- *Protection Profile (PP)*

  PP defines a set of requirements and objectives in an implementation-independent way for a category of products or systems which meet similar consumer needs for security (for example, Firewall-PP). A PP is intended to be reusable and has been developed to support the definition of functional standards and as an aid to formulating acquisition specifications. Chapter 4 presents the functional protection profile for a networked home environment distribution platform, applying the Protection Profile defined in CC version 2.1.

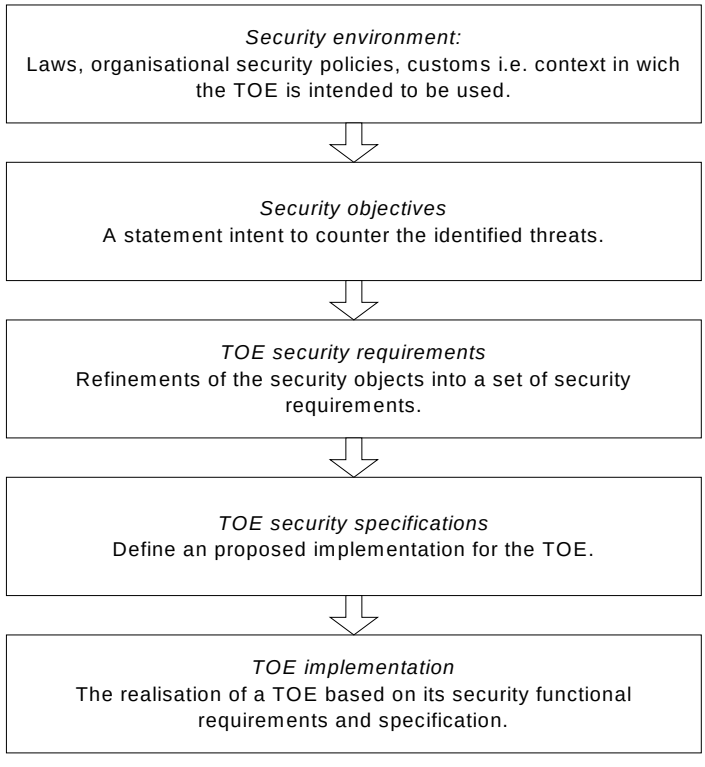


Figure 7. The hierarchical framework of Common Criteria.

Table 2. Three parts of the Common Criteria.

	Consumers	**Developers**	**Evaluators**
Part 1: Introduction and General Model	For background information and reference purposes	For background information, and reference for the development of requirements and formulating security specifications of TOEs	For background information and reference purposes. Guidance structure for PPs and STs
Part 2:Security Functional Requirements	For guidance and reference when formulating statements of requirements for security functions	For reference when interpreting statements of functional requirements and formulating functional specifications of TOEs	Mandatory statement of evaluation criteria when determining whether TOE effectively meets claimed security functions
Part 3: **Security Assurance Requirements**	For guidance when determining required levels of assurance	For reference when interpreting statements of assurance requirements and determining assurance approaches of TOEs	Mandatory statement of evaluation criteria when determining the assurance of TOEs and when evaluating PPs and STs

Functional and assurance requirements

CC describes the functional component classes for expressing the security requirements within PPs and STs. These requirements describe the desired security behaviour expected of a TOE and are intended to meet the security objectives stated in a PP or ST, as well as counter threats in the assumed operating environment of a TOE and cover any organisational security policies. CC Version 2.1 (part 2) describes following classes [2]:

- Class FAU: Security Audit
- Class FCO: Communication
- Class FCS: Cryptographic support
- Class FDP: User data protection
- Class FIA: Identification and authentication
- Class FMT: Security management
- Class FPR: Privacy
- Class FPT: Protection of TSF
- Class FRU: Resource utilisation
- Class FTA: TOE Access
- Class FTP: Trusted path/channels

Each of these classes is further divided into functional families. For example, the family *Cryptographic key operation* can be found in Class FCS. This is further divided into various functional requirements, such as *Cryptographic key generation* and *Cryptographic key destruction*.

Assurance requirements are described in CC part 3 [2], and they are similarly divided into classes and families, like functional requirements. From these families, CC constructs a set of *Evaluation assurance levels (EAL)*. These assurance levels provide backward compatibility to earlier source criteria - e.g. to TCSEC. The CC describes seven assurance levels. Their comparability to assurance levels of TCSEC is presented in Table 3. The additional assurance level EAL0 is added to present comparability to TCSEC's level D.

Table 3. Correlation between CC's and TCSEC's assurance levels.

Common Criteria	**TCSEC**
EAL0:	D: Minimal protection
EAL1: functionally tested	
EAL2: structurally tested	C1 Discretionary security protection
EAL3: methodically tested and checked	C2 Controlled access protection
EAL4: methodically designed, tested and reviewed	B1 Labelled security protection
EAL5: semi-formally designed and tested	B2 Structured protection
EAL6: semi-formally verified design and tested	B3 Security domains
EAL7: formally verified design and tested	A1 Verified design

2.4 Cryptographic protocols and algorithms

The mathematics underlying present cryptographic techniques can be very complex, and in this text is only described when it is necessary for understanding the algorithm and cryptography protocol. More exact proofs of the mathematical basis can be found from references [10], [12].

A protocol is defined to have a sequence of steps, from start to finish. Every participant must know the protocol and agree to follow it. Each step of the protocol must be well defined and there must be a specified action for every possible situation. A cryptographic protocol is simply a protocol that uses cryptography. Cryptographic protocols make it possible to transfer data via untrusted public networks. The protocol can be *arbitrated, adjudicated* or *self-enforcing*. The difference between an arbitrated and an adjudicated protocol is that in an adjudicated protocol a neutral third party is used only when there is a dispute between the participants. This can be considered a special circumstance where the arbitrator and adjudicator are both disinterested and trusted third parties. 'Disinterested' means that the third party has no malicious interest in the protocol and has no allegiance to any of the parties involved. A self-enforcing

protocol requires no arbitration and there cannot be any disagreements between the parties. All ill-defined intents are detected [10, pp. 21–31].

Cryptographic protocols provide mechanisms to identify and authenticate data transmission participants and make sure that only these legitimate participants can share confident information. Before sending, data is encrypted in order to achieve privacy between the participants.

Cryptographic algorithms are mathematical functions used for encryption and decryption. The security of *restricted algorithm* is based on keeping the way that the algorithm works a secret. Restricted algorithms are not adequate for large groups of users because every time a user leaves the group the algorithm must be changed to maintain privacy within the group. This is solved by using *keys* for encryption and decryption. A key is chosen from a large number of possible values. This range of values is called *keyspace*. Keys used for encryption and decryption can be the same or different, depending on the algorithm. All security in key-based algorithms is based in the keys, not the algorithm. Knowing the algorithm is of no use in decrypting the secret unless the malicious user knows the right key.

Symmetric algorithms (also called *secret-key algorithms, single-key algorithms* or *one-key algorithms*) are algorithms where the encryption key can be calculated from the decryption key and vice versa. This requires that the sender and receiver agree on a key before establishing a secure communication link. When the key is revealed, security is lost and the participants must agree on a new key. Two categories of symmetric algorithms can be defined: *stream algorithms (or stream ciphers)*, which operate on the plain text a single bit at a time, and *block algorithms (or block ciphers)* which operate on the plain text in a group of bits (block size can be, for example, 64 bits, large enough to prevent analysis and small enough to be workable). Although fast, symmetrical algorithms have problems that must be taken into consideration:

1. Key distribution must be done in secret. If a key is revealed, security is compromised.

2. If every pair of users in a network has a separate key, the total number of keys increases rapidly as the number of users increases. A network of *n* users requires *n(n-1)/2* keys.

The most-used and best-known symmetrical cryptography algorithm is *Data Encryption Standard (DES)*. It has been the world-wide standard for over 20 years. DES encrypts data in 64-bit blocks. Key length is often expressed as a 64-bit number, but key length is 56 bits because every eighth bit is used for parity checking. All security is based on keys and in DES there are numbers that are considered weak keys, but they can be easily avoided [10, pp. 265-283]. Weak keys usually introduce some kind of symmetry (key is entirely 0s or one half is 1s and the other 0s, for example). A list of weak keys consists of 64 keys and the odds on picking a weak key from a keyspace size of 72 057 594 037 927 936 possible keys is negligible. In the near future, DES is to be replaced with a new standard, *Advanced Encryption Standard* (AES), which specifies the algorithm which must implement block cipher symmetric key cryptography and must support block sizes of 128-bits and key sizes of 128-, 192-, and 256-bits. NIST (National Institute of Standards and Technology) has selected Rijndael as the proposed AES algorithm. Other symmetrical algorithms are IDEA, RC4, RC5 and Blowfish [11, pp. 24–26]

Public key algorithms (also called *asymmetric algorithms*) use different keys for encryption and decryption and the decryption key cannot be calculated from the encryption key (at least, in a reasonable time). The encryption key can be made *public*. An untrusted third party can use the encryption key to encrypt the message, but only a specific person with the right decryption key can decrypt the message. The encryption key is often called a public key and the decryption key is called a private key (or secret key). In digital signatures, the message is encrypted using a private key and decrypted with a public key.

RSA algorithm's mathematical basis is presented here on a general level, without proof, to gain more understanding of public key algorithms. RSA was developed by three people - Rivest, Shamir and Adleman - after whom it is named. RSA is suitable for both encryption and digital signatures. RSA is based on the use of two keys, public and private. Security lies in the difficulty involved in factoring large numbers. Both of the keys are the function of a pair of large prime

numbers. Two large random primes, p and q, must be chosen to generate two keys. The product of these primes is then computed:

$$n = pq \tag{1}$$

The encryption key, e, is then chosen randomly so that e and $(p-1)(q-1)$ are relatively prime. The decryption key, d, is obtained from the following equation:

$$\begin{aligned} & ed = 1 \bmod (p-1)(q-1) \\ & \Leftrightarrow d = e^{-1} \bmod [(p-1)(q-1)] \end{aligned} \tag{2}$$

The numbers d and n are also relatively prime. The numbers e and n are the public key and the number d is private key. Two primes, p and q, are no longer needed after key generation and they should be discarded, but never revealed [10, pp. 466–474].

For message encryption, the message is divided into numerical blocks smaller than n. That means if both p and q are four-digit primes, n will have just under eight digits and each message block m_i should be just under eight-digits long. The encrypted message is composed of a similarly sized message block c_i of about the same length. The encryption formula is:

$$c_i = m_i^e \bmod n \tag{3}$$

Decryption is computed for each encrypted block c_i:

$$m_i = c_i^d \bmod n \tag{4}$$

Message Authentication and digital signature

Message authentication is a procedure for ensuring that the received message came from the claimed source and has not been altered en route. Verification of sequencing and timeliness is also possible. A digital signature is a technique for countering repudiation by source or destination. The authentication mechanism

must provide a means of producing an authenticator, usually a numerical value - i.e. function - of the message and a protocol to verify authenticity.

The authenticator can be produced with the following functions [12, pp. 237–249]:

- *Message encryption*
  The entire message is encrypted and the resulting cipher text is used as the authenticator.
- *Message authentication code (MAC)*
  A secret key and a public function of the message are used to procedure fixed-length value that serves as the authenticator.
- *Hash function*
  A public function that maps a message of arbitrary length to a fixed-length hash value that is used as the authenticator.

Message authentication code (MAC, cryptographic checksum) is a fixed-size block of data that is appended to a message. Two communicating parties, A and B, share a common secret key and when A sends a message to B, it calculates the MAC as a function of the message M and the key C_K:

$$MAC = C_K(M) \tag{5}$$

The message and MAC are then transmitted to B and B is able to perform the same calculation using the same secret key. This newly generated MAC is compared to the received MAC. If the secrecy of the key is not compromised and the received MAC is equal to the calculated MAC, the message's authenticity is ensured [12].

The hash function, also called message digest, is public, there is no secrecy as to how it is generated. The security of the hash function lies in its *one-way* nature: it is easy to generate a hash code from the message but it is very hard to generate a message that hashes to the desired value. It is also *collision free*, which means that it is hard to generate two messages with the same hash value [10, pp 30–31]. Well-known one-way hash functions are MD5, with a hash length of 128 bytes (MD stands for message digest), and SHA, with a hash length of 160 bytes (secure hash algorithm). Because both of them are based on MD4, SHA can be

considered more secure. In addition to that, SHA is not known to be vulnerable to cryptoanalytic attacks.

Digital signature is a secure hash code that the signer has calculated. It is usually calculated using some asymmetric algorithm [11, p. 35]. The requirements for a good signature are [10, pp 34–37]:

1. The signature is authentic and deliberately signed by the alleged signer.
2. The signature is non-forgeable, i.e. no one else is able to make it or change it.
3. The signature is not reusable and is part of the document. Therefore, it is not possible to move the signature to another document
4. The signed document is unchangeable. After the document has been signed, it cannot be changed.
5. The signature cannot be repudiated, i.e. the signer cannot deny that it has been signed by him.

Generally speaking, asymmetric algorithms meet these requirements. Examples of public key algorithms are RSA and DSA (digital signature algorithm). In RSA either public key or private key can be used for encryption. If a message is encrypted using the private key, the output of the calculation is digital signature. In DSA there are separate algorithms for digital signatures and they cannot be used for encryption.

2.5 Data security: Three areas of concern

As discussed in Chapter 2.1, the computer system must be reliable, integrated and available for users. How these areas are emphasised depends on the system. An air traffic control system, for example, does not need high confidentiality but availability of service and integrity can be crucial in order to avoid serious accidents [13, pp 19–31]. Most of the security techniques are based on existing standards and cryptography. There is no strict grouping of technologies into specific areas of concern, such as confidentiality, but, in most cases, the technologies are presented under the heading they are most attached to.

2.5.1 Confidentiality

Confidentiality denotes protection of data from unauthorized access [13]. Cryptographic protocols, encryption and authorization mechanisms are ways of avoiding the most common confidentiality threats that were discussed in Chapter 2.2:

- Impersonation
- Eavesdropping
- Packet modification.

Confidentiality models describe the actions that must be taken to ensure the confidentiality of information. The most widely used model is the *Bell-LaPadula* model, which defines the relationships between objects (files, programs and systems) and subjects (users and processes that cause information to flow between objects). Thereby, the relationship denotes "the subject's assigned level of access or privilege and the object's level of sensitivity" [13]. Subjects can access objects to read, or read and write information. The lattice principle of the *Bell LaPadula* model specifies that subjects are allowed to:

- write access to objects at the same or higher level as the subject
- read access to objects at the same or lower level as the subject
- read and write access to objects only at the same level as the subject.

This model ensures that the subject is not able to write information at a higher level into a lower classified object and prevents the subject from giving higher classified information to a lower classified subject. Consequently, in this kind of *flow model* information at a given security level flows only to an equal or higher level.

Another widely used model is the access control model, which organises a system into objects (target of actions), subjects (actors, i.e. persons or programs doing the action) and operations (process of the interaction). A set of rules is used to define which operations can be performed on an object by which subject. In addition to the flow model, this model ensures not only confidentiality but also the integrity of information as well [13].

Confidentiality models can be implemented with trusted computer system evaluation criteria (Orange and Red book) and Common Criteria, which are discussed in more detail in Chapter 2.3.

2.5.2 Integrity

Integrity signifies protection of data from unauthorized access and modification, as well as maintenance of data in the state that users expect [13]. Access controlling is a very essential part of maintaining integrity. Thereby, users should be able to access only those resources needed to perform their tasks (*Need-to-Know access*). Other principles for establishing integrity policies are *separation of duties* (no single user has control of a transaction from beginning to end) and *rotation of duties* (a periodic changing of job assignments to avoid the possibility of the user controlling the complete transaction) [13].

Integrity models help to describe what has to be done in order to accomplish the integrity policy. Different models address different ways of achieving three goals of integrity [13]:

1. Preventing unauthorized users from making modifications to data or programs
2. Preventing authorized users from making improper or unauthorized modifications
3. Maintaining internal and external consistency of data and programs.

The National Computer Security Center Report [14] describes five integrity models, viz.:

1. Biba
2. Goguen-Meseguer
3. Sutherland
4. Clark-Wilson
5. Brewer-Nash.

Biba's model is similar to the Bell LaPadula model for confidentiality; the Sutherland model focuses on the problem of interference; and the Clark-Wilson on the well-formed transactions and separation of duties. The Brewer-Nash model, for its part, concentrates on the implementation of the dynamically changing access authorizations. More of these can be found in Reference [14].

The Goguen-Meseguer model provides an approach to secure systems that is based on automaton theory and domain separation. Goguen-Meseguer provides a strict distinction between the security policy and security models. Thereby, the security policy denotes the security requirements on a given system and the security model is abstraction of the system itself [14]. In this model security policy is based on the concept of non-interference, where "one group of users, using a certain set of commands, is non-interfering with another group of users if what the first group does with those commands has no effect on what the second group of users can see." Non-interference is achieved by separating users into different domains. Domain is defined as "the set of objects that a user has the ability to access" [14].

2.5.3 Availability

Availability of the computer system means that it is accessible to legitimate users when they need it. Degration of availability is usually caused by denial of service attacks or loss of physical computer data processing capabilities caused by natural disaster, human actions and hardware or software failures - which are probably more common [13]. Maintaining availability consists mostly of physical, technical and administrative issues. Physical issues include access control (in the physical sense of the term – locked doors, etc.) and environmental issues (temperature control and fire and water control mechanisms). Technical issues, for their part, consist of fault-tolerance computing (hardware redundancy and disk mirroring). Administrative issues are access control policies and user training, etc. [13].

2.6 Authentication and authorization of a user

The authentication of a user answers the question "who are you?". In the case of public key cryptography, this question is answered by providing the questioner with an appropriate key or *certificate* as proof of the legitimacy of a user. Once the user is properly and reliably identified, the next question is what the user is allowed to do and what resources the user is allowed to access. Authorization answers these questions and grants access rights to the user, as defined in 2.1. Authorization can be implemented in many ways. The Java platform's protection-domain-based solution is presented in Chapter 3. As an example of authentication, the X.509 authentication service is presented next in more detail.

2.6.1 X.509 Authentication Service

X.509 is an ITU-T recommendation and is part of the X.500 series of recommendations that define a directory service [12, pp. 341–350]. Directory, in this case, is defined as a server or distributed set of servers that maintains a database of information about users. This information consists of mapping from user name to network address, and other attributes and information about users.

X.509 prescribes a framework for taking care of authentication services by the X.500 directory to its users. The directory can work as a repository of public key certificates. The certificate consists of the public key of a user and is signed with the private key of a trusted *certification authority* (CA). The X.509 certificate format is used, for example, in S/MIME (Secure/Multipurpose Internet Mail Extension), IP Security and SSL/TLS (Secure Socket Layer/Transport Layer Security), and SET (Secure Electronic Transaction). X.509 is based on the use of public-key cryptography and digital signatures and does not mandate use of any specific algorithm. However, it does recommend the use of RSA. Neither does the standard dictate a specific hash algorithm in a digital signature scheme, although use of the hash function is assumed.

Certificates

The most important feature of the X.509 scheme is the public-key certificate associated with each user. Certificates are assumed to be created by some trusted

CA and placed in the directory by the CA or by the user. The directory server provides an easily accessible location for users to obtain certificates, thus it is not responsible for the creation of public keys or for the certification function. Figure 8 depicts the general format of a certificate. It includes the following parts:

1. *Version*: Differentiates among successive versions of the certificate formats, the default version being version 1. The value of the version must be version 2, if the Initiator Unique Identifier or Subject Unique Identifier is present. The version must be version 3 if one or more extensions are present.

2. *Serial Number*: An integer value that is unambiguously associated with this certificate and is unique within the issuing CA.

3. *Signature algorithm identifier*: Algorithm used to sign this certificate, together with associated parameters. This field has little utility because the same information is repeated in the *Signature* field.

4. *Issuer Name*: X.500 name of the CA that has created and signed the certificate.

5. *Period of validity*: This field includes the first and last date on which the certificate is valid.

6. *Subject name*: The name of the user to whom this certificate refers. In other words, this certificate certifies the public key of the subject who holds the corresponding private key.

7. *Subject's public-key information*: Public key of the subject and identifier of the algorithm for which this key is to be used, plus associated parameters.

8. *Issuer unique identifier*: If the X.500 name of the CA has been reused for different entities, this optional bit string field is used to uniquely identify the issuing CA.

9. *Subject unique identifier*: If the X.500 name of the subject has been reused for different entities, this optional bit string field is used to uniquely identify the subject.

10. *Extensions*: A set of one or more extension fields.

11. *Signature*: Covers all the other fields of the certificate, including the hash code of other fields encrypted with the CA's private key. This field also includes the signature algorithm identifier.

The X.509 standard [15] uses the following notation to define a certificate:

$CA\langle\langle A\rangle\rangle = CA\{V, SN, AI, CA, T_A, A, Ap\}$,

where

$Y\langle\langle X\rangle\rangle$ = the certificate of user X issued by CA Y

$Y\{I\}$ = the signing of I by Y, which consists of I with an encrypted hash code appended.

The CA signs the certificate with its secret key. If a user knows the corresponding public key, the user can verify that the certificate signed by the CA is valid.

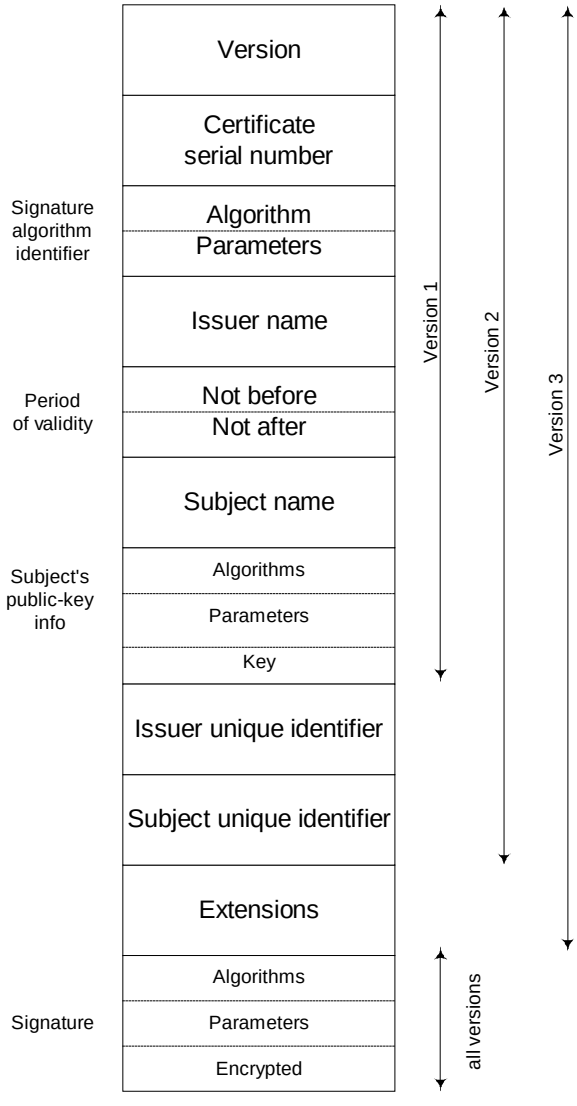


Figure 8. X.509 Certificate format.

Figure 9 shows an example of a real certificate from a trusted certification authority. It includes all the mandatory fields defined above that are valid for version 1.

```
Version: V1
 Subject: OU=VeriSign Trust Network, OU="(c) 1998 VeriSign, Inc. - For authorized use only",
          OU=Class 2 Public Primary Certification Authority - G2, O="VeriSign, Inc.", C=US
 Signature Algorithm: SHA1withRSA, OID = 1.2.840.113549.1.1.5

 Key:  com.sun.rsajca.JSA_RSAPublicKey@34a74b

 Validity:    From: Mon May 18 03:00:00 GMT+03:00 1998,
              To: Sat May 19 02:59:59 GMT+03:00 2018

 Issuer: OU=  VeriSign Trust Network, OU="(c) 1998 VeriSign, Inc. - For authorized use only", OU=Class 2
              Public Primary Certification Authority - G2, O="VeriSign, Inc.", C=US

 SerialNumber: 1f42285f 3c880f8e 3c89b384 b3ab1f1c

 Algorithm: SHA1withRSA

 Signature:
              0000: 11 45 A8 A4 7F F1 E3 73   20 CA BD EE DF F5 87 23  .E.....s ......#
              0010: 91 3D 8D AC 47 45 1A DE   6D DB 54 21 CE 0E 83 0E  .=..GE..m.T!....
              0020: F8 DC E5 43 D5 EB 2E 61   91 23 E2 72 00 34 55 F7  ...C...a.#.r.4U.
              0030: C4 CF 11 33 DD C1 E4 22   23 5C 50 53 19 F8 64 E7  ...3..."#\PS..d.
              0040: F7 09 0F 45 51 A0 57 2B   DF BC 22 66 FE 31 70 7B  ...EQ.W+.."f.1p.
              0050: 25 3A 0F C5 8A 7E C3 BB   72 01 CC F0 BD 4D 52 81  %:.....r....MR.
              0060: A4 1B 58 58 53 D5 53 3A   F5 0E 6A DA E9 AF C4 E1  ..XXS.S:..j.....
              0070: 58 F3 42 6F 54 62 47 AC   31 94 D1 0D CE EF 1D 31  X.BoTbG.1......1
```

Figure 9. X.509 Certificate.

Obtaining a User's certificate

A certificate generated by a CA has the following characteristics:

- Any user that knows the public key of the CA can recover the certified user's public key.
- Only CA can modify the certificate without detection, so it can be considered tamper proof.

Being non-forgeable, certificates can be placed in a directory without the need for special protection. There is a common trust in the CA if all users subscribe to the same CA and all user certificates can be placed in the directory that is accessible to all users. In addition to that, users can send their certificates directly to other users. When user B has A's certificate, B can be confident that the message it encrypts with A's public key will be safe from eavesdropping and that the message signed with A's private key cannot be modified without detection.

If there is a large community of users (e.g. the Internet), it is not possible for all users to subscribe to the same CA. Each participating user must have a copy of the CA's own public key to verify signatures, because it is the CA that signs the certificates. The CA's public key must be given to each user in such way that the integrity and authenticity of the key is not compromised. It is more practical to have a number of CAs, each of which provides its public key to some smaller group of users.

Let us assume that user A has obtained the certificate from CA X_1 and B has obtained the certificate from CA X_2. If A does not unambiguously know the public key of X_2, B's certificate is useless to user A because it cannot verify and, therefore, trust it, even though B's certificate is readable to A. If the two CAs have securely exchanged their own public keys, A is able to obtain B's public key using the following procedure:

A obtains the certificate of X_2 signed by X_1. Knowing X_1's public key, A is able to get X_2's public key from its certificate and verify it using X_1's signature on the certificate. Next, A obtains the certificate of B signed by X_2 and is able to verify it and securely obtain B's public key because A has a trusted copy of X_2's public key.

In the notation of X.509, this chain of certificates which A has used to obtain B's public key is expressed as:

$X_1\langle\langle X_2\rangle\rangle X_2\langle\langle B\rangle\rangle$

Similarly, B can obtain A's public key with the reverse chain:

$X_2\langle\langle X_1\rangle\rangle X_1\langle\langle A\rangle\rangle$

An arbitrarily long path of CAs can be followed to produce a chain. A chain with n elements is expressed as:

$X_1\langle\langle X_2\rangle\rangle X_2\langle\langle X_3\rangle\rangle \ldots X_N\langle\langle B\rangle\rangle$

It is assumed that each pair of CAs in the chain (X_i, X_{i+1}) must have created certificates for each other. Every certificate of CAs by CAs needs to be in the

directory and the users must know how they are linked in order to follow a path to another user's public key certificate. In X.509's suggestion, CAs are arranged in a hierarchy to make straightforward navigation possible. Figure 10 depicts a CA hierarchy where associated boxes indicate certificates that are maintained in the directory for each CA entry. For each CA there are two types of certificates included in its directory entry:

Forward certificates: Certificates of X generated by other CAs.

Reverse certificates: Certificates generated by X that are the certificates of other CAs.

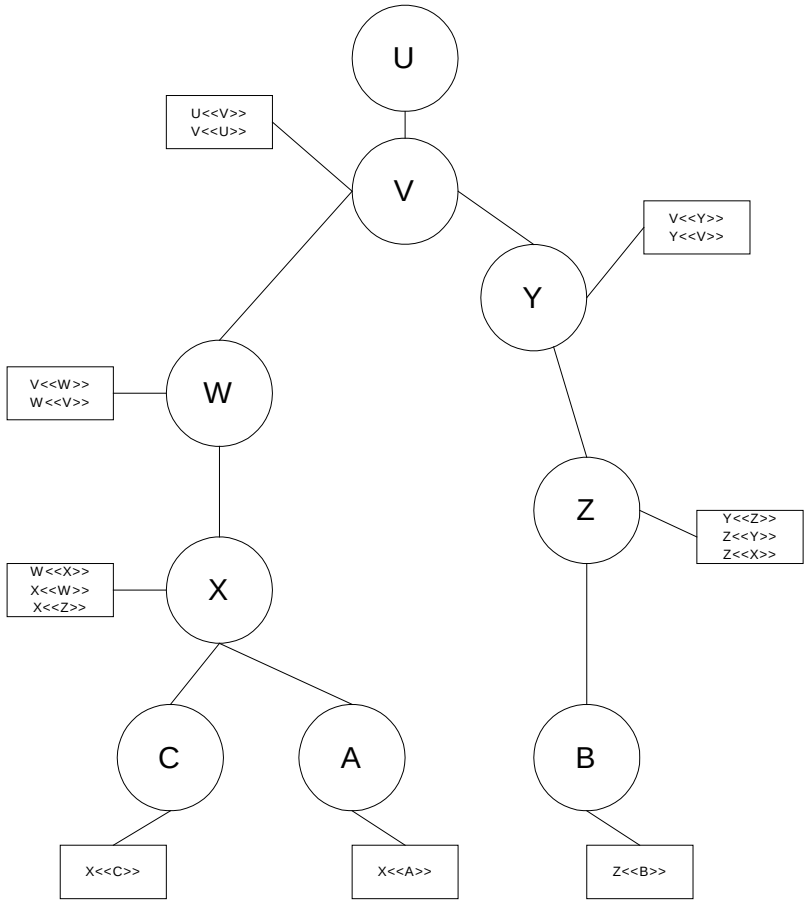


Figure 10. Example of X.509 hierarchy.

In this case, user A can obtain the following certificates from the directory to establish a certification path to B:

X<<W>>W<<V>>V<<Y>>Y<<Z>>Z<<B>>

When A has acquired all these certificates, it can unwrap the certification path in sequence in order to recover a trusted copy of B's public key.

Certificate revocation

One of the certificate's mandatory fields is its period of validity. In a normal situation a new certificate is issued just before the expiration of the old one. There are also situations when it is desirable to revoke a certificate before it expires:

1. The user's private key is assumed to be compromised.
2. The user is no longer certified by this CA.
3. The CA's certificate is assumed to be compromised.

Every CA must have a list of all revoked, but not expired, certificates issued by the corresponding CA, which must be posted on the directory. These certificates include both those issued to users and those issued to other CAs. Figure 11 shows the general format X.509 of a certificate revocation list (CRL).

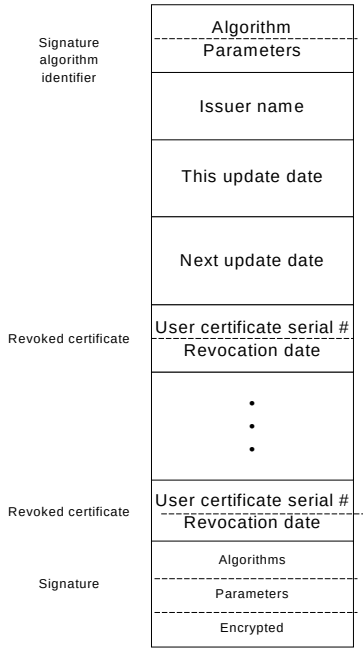


Figure 11. X.509 Certificate revocation list format.

The issuer signs each CRL and, among the other fields depicted in Figure 11, includes an entry for each revoked certificate. One entry consists of the serial number of the certificate and its revocation date. The serial number is sufficient information for identifying the certificate because it is unique within the CA.

The user must determine whether or not the certificate has been revoked. One possible way is to check the directory every time a certificate is received, but, because this is time consuming (and possibly expensive), it is more practical for the user to maintain a local copy - i.e. cache - of certificates and lists of revoked certificates.

Strong Authentication

X.509 describes three authentication procedures - which take advantage of the approach to authentication just presented - that make use of public key signatures. It is assumed that the two participating users know each other's public key. The public key can be obtained either from the directory or directly from the initial message from each side. In Figure 12 three authentication procedures are presented in a way X.509 describes them.

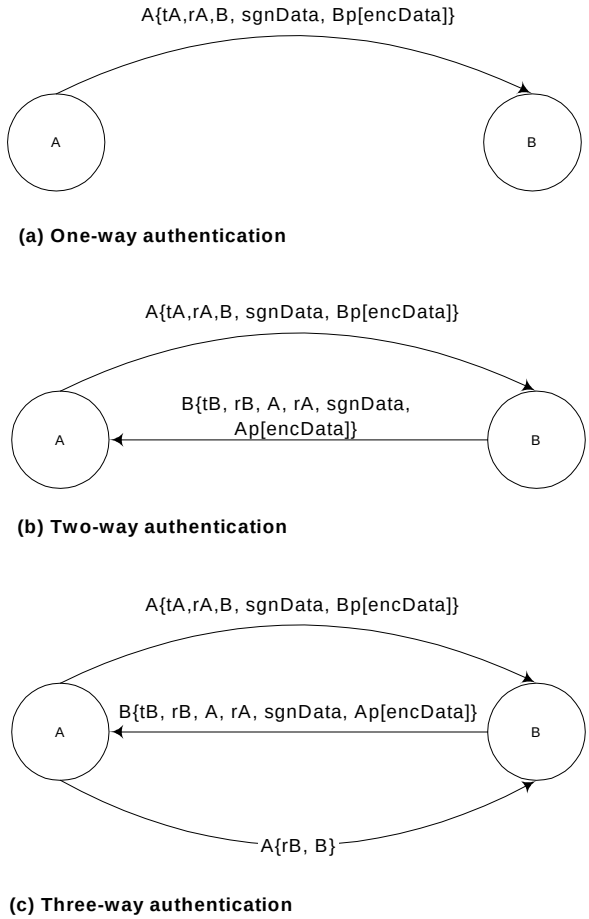


Figure 12. X.509 Authentication procedures.

One-way authentication involves a single transfer of information from one user (A) to another (B) and establishes the following:

1. The identity of A and that the message was actually generated by A.
2. The identity of B and that the message was intended to be sent to B.
3. The integrity and originality (not having been sent two or more times) of the authentication message.

A sends the following message to B:

$B \rightarrow A$, $A\{t^A, r^A, B, sgnData, Bp[encData]\}$, where

t^A	= timestamp
r^A	= non-repeating number
sgnData	= digital signature (optional)
Bp[encData]	= session key (encDate) for B, encrypted with B's public key (Bp)

The timestamp consists of one or two dates: the generation date of the token (optional) and the expiry date. This is used to prevent delayed delivery of messages. The nonce r^A is used to detect attacks that threaten the integrity (replay attacks and forgery). This value must be unique within the expiration date of the message so B can store the nonce until it expires and reject any new messages with the same nonce. This message is signed with A's public key. For authentication purposes only, the message is used simply to present credentials to B. The message can also convey information (sgnData) which is within the scope of the signature to guarantee its authenticity and integrity. In addition to this, the message can used to transfer a session key to B, which is encrypted with B's public key.

Two-way authentication establishes the following three elements in addition to the three defined in one-way authentication:

1. The authentication message was generated by B and was intended to be sent to A
2. The integrity and originality of the reply
3. The mutual secrecy part of the messages (optional).

In the two-way authentication scheme both parties are able to verify each other. The reply message includes the nonce from A to validate the reply. In addition to that, it includes the timestamp and nonce generated by B. As in one-way authentication, the message may include signed additional information and a session key encrypted with A's public key.

Three-way authentication includes a final message from A to B which contains a signed copy of the nonce r^B. Both nonces are echoed by the other side. Therefore, each side can check the returned nonce to detect replay attacks. This approach was chosen to avoid the need of synchronised clocks.

2.6.2 Secure socket layer (SSL) and transport layer security (TLS)

SSL was originally developed by Netscape and is widely accepted as the authentication and encryption mechanism for communication between client and server. The Internet Engineering Task Force (IETF) standard called Transport Layer Security [16] is based on SSL and is very close to SSL version 3.0 [17]. This discussion is based on SSL version 3.0. SSL is two layers of protocols, as depicted in Figure 13.

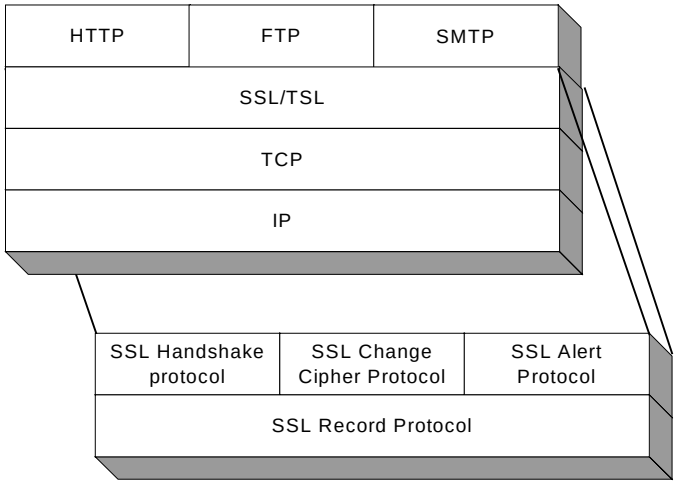


Figure 13. SSL protocol stack.

SSL record protocol provides basic security services to higher layer protocols: *the handshake protocol, the change cipher spec protocol* and *the alert protocol,* of which the handshake protocol is examined in more detail. These protocols are mainly used in the management of SSL [12, pp. 444–457]. Two important terms, the SSL *Connection* and the SSL *Session*, are defined as follows:

1. *Connection* is a transport that provides a suitable type of service. In SSL's case, connections are peer-to-peer relationships. Every connection is associated with one session.

2. *Session* is an association between a client and a server and is created with handshake protocol. Sessions define a set of cryptographic parameters that can be shared among multiple connections to avoid expensive negotiation of new security parameters for each connection.

Between two participants there can be multiple secure connections and, practically, one session, although multiple simultaneous sessions are possible.

The most complex part of SSL is the handshake protocol, which allows the server and client to authenticate each other and negotiate security parameters (encryption and message authentication code (MAC) algorithm and cryptographic keys). This protocol is used before any application data is transmitted [12]. The handshake protocol action is depicted in Figure 14, where optional messages are presented in lines of dots and dashes.

Client_hello and server_hello establish security capabilities, including protocol version, session ID, cipher suite, compression method and initial random numbers. Next, the server may send an optional certificate, mandatory key exchange and, optionally, request a client certificate. If the server sends a request, the client sends its certificate. The client then sends key exchange and optional certificate verification. After this, cipher suites are changed and finish messages sent to end the handshake protocol. At this point, the handshake is done and client and server are ready to change application layer data.

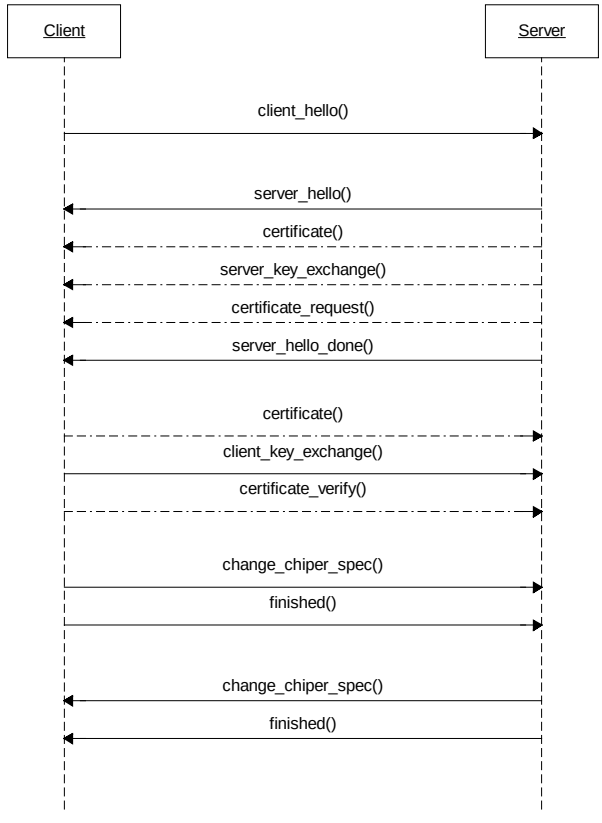


Figure 14. Handshake protocol action.

2.7 Auditing

Audition and audit records are foundational tools for intrusion detection. Audit records can be gathered by using two different schemes [12, pp. 492–501]:

1. *Native audit records*: Audit collection is done by the operating system and no additional software is needed.
2. *Detection-specific audit records*: External collection facility that collects information needed by the intrusion detection system. The advantage over native audit records is that the audit trail only contains the necessary information and in a more convenient form, but there is overlap if the operating system itself has auditing capabilities.

3. Java Technology and security

3.1 Introduction

Java was originally known as "Oak" and was developed by Sun Microsystems to meet the needs of embedded consumer electronic applications at the beginning of 1990. Java is a general-purpose object-oriented programming language and, from the start, has been designed to address the special characteristics of networked software, including multiple host architectures and secure delivery of software components [18, pp. 61–91], [19]. These issues are mainly solved by means of Java *Bytecode* and the *Java virtual machine*, later referred to as JVM. Therefore, Java is not only programming language but also a complete software platform, including *application programming interface* (API) and virtual machine. Under the Java platform lies an operating system and hardware, thus the Java platform is not a replacement for an operating system but hides the operating system from the application. The Java platform (runtime environment) and compilation environment is described in Figure 15. Different parts of the Figure are described in following chapters.

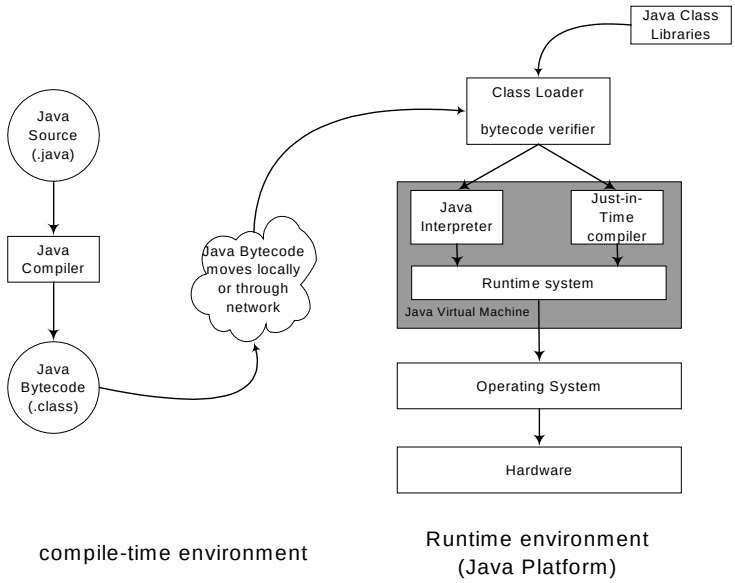


Figure 15. Java's runtime environment and compilation environment.

3.2 The Java virtual machine

Generally speaking, the Java virtual machine can denote three different things [20, pp. 134–190]:

1. The abstract specification
2. A concrete implementation
3. A runtime instance.

The abstract specification is a concept of the Java virtual machine, and is defined in Reference [18]. Concrete implementation exists on many platforms and can be either complete software implementation of the abstract specification or a combination of software and hardware. A runtime instance of concrete implementation hosts a single running Java application. The Java virtual machine is an essential part of the Java platform because, for its part, it provides solutions to many features of Java that are considered to be Java's advantage over previous programming languages. JVM is responsible not only for Java's hardware- and operating system independence - but also for protecting users from running malicious code. These security issues include code-checking mechanisms at many levels. JVM is an abstract computing machine and what is similar to real computing machines is that it has an instruction set and is able to manipulate various memory areas at runtime. JVM does not make any presumptions of the underlying hardware, operating systems or even implementation technology. This makes possible, for example, implementation of JVM directly into a silicon chip. JVM does not deal directly with Java language but rather with a particular binary format - class file format - which contains JVM instructions (i.e. bytecode), symbol table and other ancillary information. To endure a certain level of security, JVM emphasizes strong format and structural constraints, both in a code and in a bytecode.

3.2.1 Life cycle of the Java virtual machine

The runtime instance of the JVM runs a Java application. Therefore, when a Java application starts, a new runtime instance is born and when the application completes, the runtime instance dies. In other words, each application runs

inside its own virtual machine. The Java version of the 'Hello World' application, a typical first programming example, is presented next. Virtual machine starts running its application by invoking *main()* method, which is defined as *public, static* and must return *void* and accept one string array as a parameter.

```
class HelloWorldApp
{
  public static void main(String[] args)
    {
        System.out.println("Hello World!");
     }
  }
```

After the Java source is saved to file (correct form of file name is *ClassName.java*) and compiled in bytecode, it can be executed in the Java virtual machine. In Sun Microsystem's Java Development Kit (JDK) compiling is done at command line by typing: *javac HelloWorldApp.java* and the virtual machine is executed by typing: *java HelloWorldApp* (optional arguments are typed after the class name).

There are two kind of threads inside JVM: daemon and non-daemon. The daemon thread is usually a thread used by JVM itself - for example, thread that performs garbage collection is a daemon thread. However, the application can mark any of the threads it creates as a daemon thread. The initial thread of the application - in other words, the main()-thread - is a non-daemon thread. A runtime instance of virtual machine continues its execution as long as any non-daemon thread is running. When all non-daemon threads are terminated, the Java application stops its execution and, at the same time, the runtime instance of virtual machine is terminated. In HelloWorldApp, main-method does not generate any other threads, which means that when main-method has done its work and exits, the application's only non-daemon thread is terminated.

3.2.2 The architecture of the Java virtual machine

The Java virtual machine specification [18] describes the abstract inner architecture of abstract JVM in terms of subsystems, memory areas, data types and instructions. These components prescribe less inner architecture of concrete implementations than define the strictly external behaviour of implementations.

Figure 16 depicts a block diagram of JVM architecture with the major subsystems and memory areas defined in the specification.

Each JVM has a class loader - which is responsible for loading types (classes and interfaces) - and execution engine - the mechanism responsible for executing the instructions contained in loaded classes. When JVM runs a program it requires memory to store bytecode and other information extracted from classes, objects the program initiates, parameters, return values from methods, local variables and intermediate results of calculations. The JVM organizes the memory it needs into several runtime data areas. Specification of runtime areas is very abstract and, therefore, gives the designer freedom to decide the structural details of implementations.

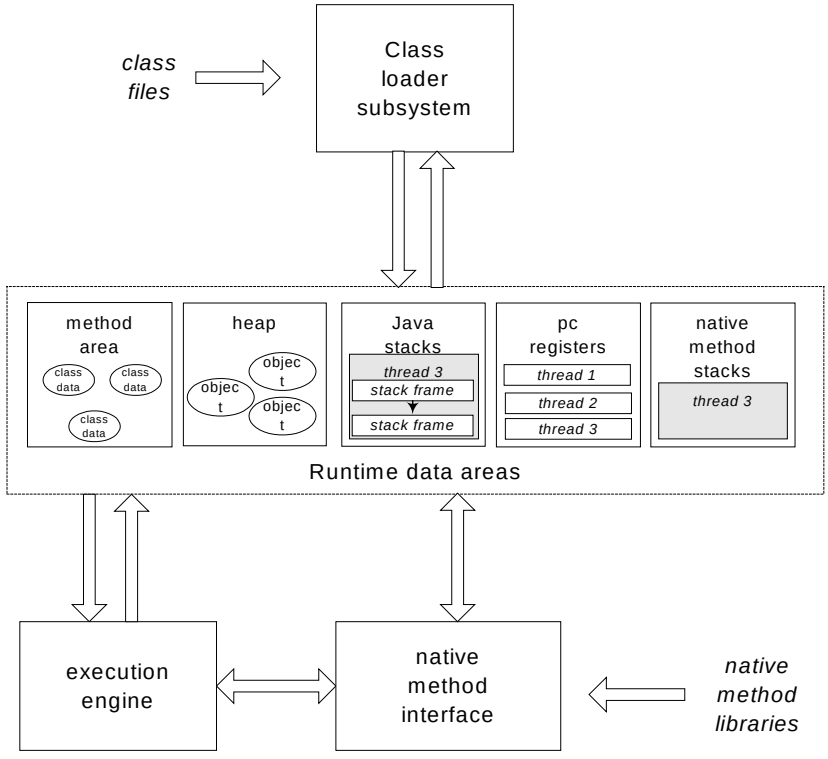


Figure 16. Inner architecture of the Java virtual machine.

Some of the runtime data areas are shared among all threads and others are unique to individual threads of an application. These per-threaded data areas are created when a new thread is created and destroyed when the thread exists [20]. Every new thread gets its own *PC register* (program counter) and *Java stack*. A thread cannot access the PC register or Java stack of another thread. The value of the PC register indicates the next instruction to execute when a thread is executing a Java method (not a native method). A Java stack stores the state of the Java method invocation for the thread. The state of method invocation comprises its local variables, the parameters with which it was invoked, its possible return value and the intermediate computation results. Correspondingly, the state of native method invocations are stored in native method stack in an implementation-dependent way, and, possibly, in registers or other implementation-dependent memory areas as well. Each runtime instance of JVM has one *method area* and one *heap*. All threads running inside JVM share these two areas. The method area is for type information parsed from loaded class files. Objects that the program initiates during execution are placed onto the heap. The Java stack comprises *stack frames* that contain the state of one Java method invocation. JVM pushes a new frame onto the thread's stack when it invokes a method. When the method completes, JVM pops and discards the frame for that method. The JVM does not have registers but the instruction set uses the Java stack to store intermediate values. The reason for this approach is to keep the instruction set compact and to ease implementation of computer architectures with few general purpose registers, as well as to facilitate the code optimization work done by just-in-time and dynamic compilers that operate at runtime in some virtual machine implementations.

3.3 Java's built-in security model

The Java platform emphasizes the networked environment and offers solutions to many issues that originate from network-oriented software. One of the major issues in networked software is security, which is also covered by Java with the extensive built-in security model which has evolved along with the Java platform. Java's security model is one of the main reasons why it is considerable technology for networked environments. Java makes it possible to download software components across the network and execute them locally. In most cases downloading of new classes and other software components is automatic and

does not need any interaction with the user [21]. Without any security, this offers a very easy way of distributing hostile code. The main focus of Java's security model is to protect end-users from malicious programs coming from untrusted sources across the network.

3.3.1 Evolution of the sandbox model

JDK 1.0 (Original sandbox model)

The original sandbox model, depicted in Figure 17, offered a very restricted environment for executing untrusted code downloaded from an open network. In Figure 17, local trusted code can get full privileges to the system resources (e.g. file system) but untrusted code (i.e. applet) from the network gets only restricted access to the system resources defined by the sandbox. Access control is taken care of by the security manager.

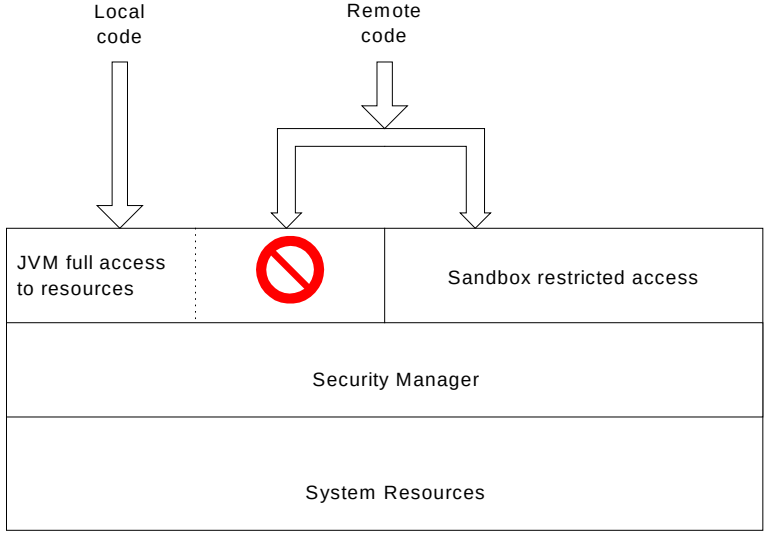


Figure 17. Original sandbox model of Java version 1.0.

The sandbox prohibits many activities, including the following [20, pp. 41–44]:

- Reading or writing to the local disk.
- Making a network connection to any hosts except the host from which the applet came.
- Creating a new process.
- Loading a new dynamic library.

JDK 1.1

JDK 1.1 provided the new concept of a "signed applet". A digitally signed applet is treated like a trusted local code and gets full access to the system resources if the signature key is recognized as trusted by the system that receives the applet. Unsigned applets are executed in the sandbox. Signed applets are delivered along with the respective signatures in signed JAR-files (Java ARchive). This concept is depicted in Figure 18.

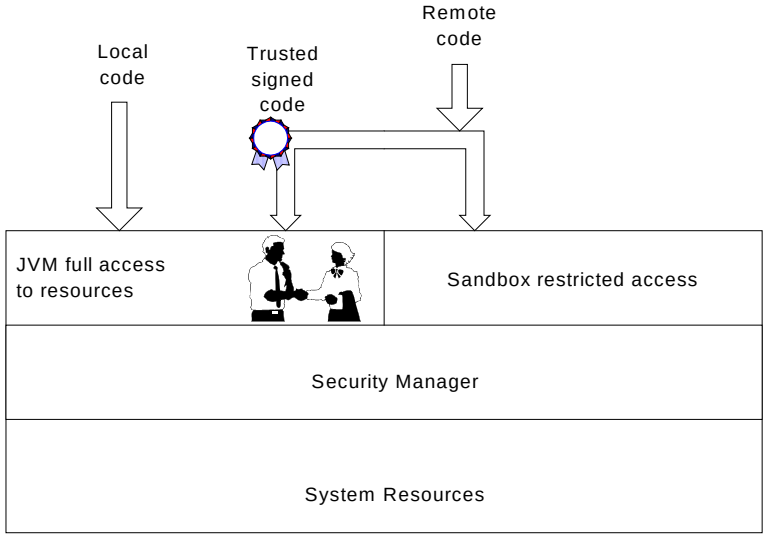


Figure 18. Sandbox model of Java version 1.1.

JDK 1.2

JDK 1.2 offers many improvements over the earlier security model. All code, local or remote, can now be subject to a security policy which defines a set of privileges for the code (remote/local, signed/unsigned) to be executed. Permissions are granted to *code sources* which are composed of a codebase URL from which the code was loaded and a set of signers that guarantee the code. Permissions can be configured by the user or system administrator. Configuration is made with a security policy file, which contains any number of permission grant entries. The default policy file looks like the following:

```
grant codeBase "file:${java.home}/lib/ext/"
{
      permission java.security.AllPermission;
};
```

To limit the privileges, the policy file needs to be modified. After deleting the default grant entry, a new one can be entered for one or more of the following limited permissions:

```
java.awt.AWTPermission
java.io.FilePermission
java.net.NetPermission
java.util.PropertyPermission
java.lang.reflect.ReflectPermission
java.lang.RuntimePermission
java.security.SecurityPermission
java.io.SerializablePermission
java.net.SocketPermission
```

Each permission defines the privilege granted to a particular resource - such as read and write to a specified file or directory, or connect to a given host and port.

The runtime system organises the code into individual domains. Each domain handles a set of classes which have the same set of permissions. A domain can be configured to be same as a sandbox, so applets can be still executed in a restricted environment if the user or system administrator wishes to do so. By default, applications are executed without restrictions, but, optionally, security policies can be defined. The domains presented in Figure 19 have more privileges than the sandbox but less than local applications.

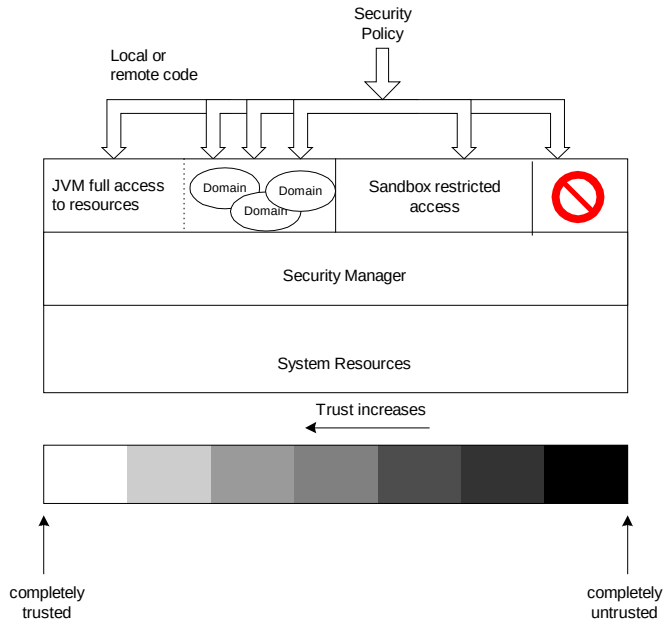


Figure 19. Domain-based security model of Java version 1.2.

3.3.2 Secure class loading and verification

The class loader brings the code into the JVM. The class loader architecture has three ways of contributing to Java's security model [20, pp. 45–59]:

1. Preventing malicious code from interfering with benevolent code.
2. Guarding the borders of the trusted class libraries.
3. Placing code into categories (protection domains) that decide which actions the code is able to take.

The class loader architecture uses separate *name spaces* for classes loaded by different class loaders in order to prevent malicious code from interfering with considerate code. A name space is set of unique names - one for each class. Once the class loader has loaded a class named *Wolf* into a particular name space, it cannot load a different class with the same name to that same name space. However, multiple *Wolf* classes can be loaded into a Java virtual machine because it is possible to create multiple name spaces inside a Java application by creating multiple class loaders.

Name spaces are important from a security point of view because they form a shield between classes loaded with different class loaders and, therefore, different name spaces. Inside the JVM, classes in the same name space can interact with each other without restriction, but classes in different name spaces cannot even detect each other's presence without a mechanism that distinctly allows interaction between them. Figure 20 depicts an example of two types with the same name. In this case, *Wolf* can be loaded to different name spaces.

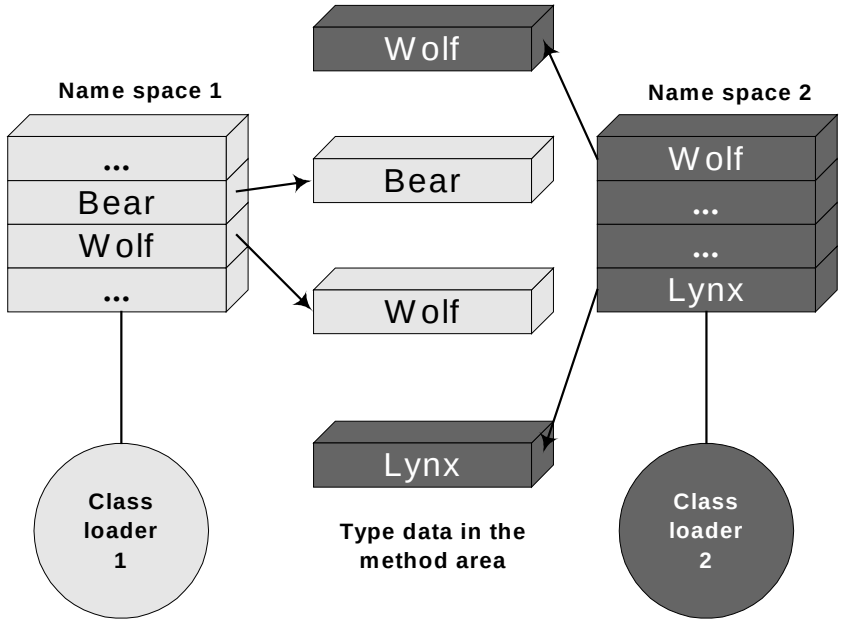


Figure 20. Two class loaders with separate name spaces.

Every name in a name space is associated with the type data in the method area that defines the type with that name. Figure 21 shows arrows from the names in the name spaces to the types in the method area that define the corresponding type. The class loader on the left, which is shown dark grey, has loaded the two dark grey types named Wolf and Lynx. Class loader 1, which is shown light grey, has loaded the two light grey types named Bear and Wolf. Because of the nature of name spaces, when the Bear class mentions the Wolf class, it refers to the dark grey Wolf - the Wolf loaded in the same name space. It has no way of knowing that the other Wolf, which is sitting in the same virtual machine, even exists.

Trusted packages can be loaded with a different class loader than untrusted packages and, thus, the class loader architecture guards the borders of the trusted libraries. Java version 1.2 uses the so-called *parent delegation model* for class loading. In this model each loader (except the bootstrap class loader) has a parent class loader, to which a particular class loader delegates its job by asking its parent to load the type. The parent then delegates the job to its parent. This process continues all the way up to the bootstrap class loader, which is the last class loader in the delegation chain. If a class loader's parent is able to load the type, the class loader returns that type, otherwise the class loader tries to load the type itself. Figure 21 shows the parent-child delegation model, where, at the other end of the chain, is the bootstrap class loader which is responsible for loading only the class files of core Java API that are needed to "bootstrap" the JVM and are considered as most trusted. This class loader is always present in JVM and, in addition to the bootstrap class loader, at least one user-defined class loader exists. User-defined class loaders are responsible for class files for the application, class files for installed or downloaded standard extensions, class files found from *class path,* etc. All these class loaders are connected in one chain of parent-child relationships.

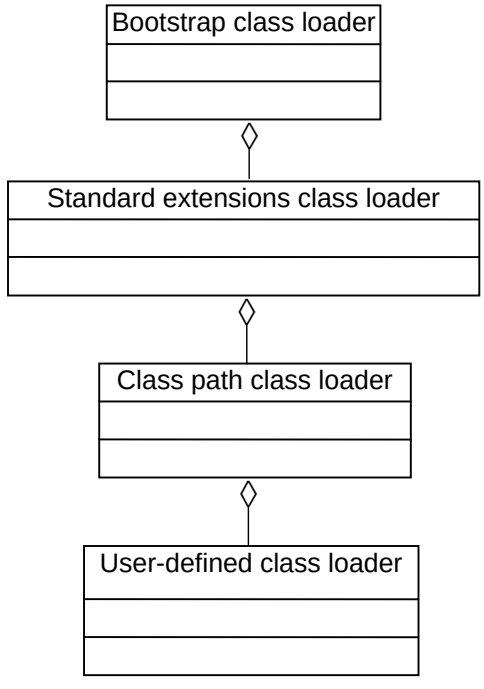


Figure 21. A class loader delegation chain.

This delegation chain makes it possible for the class loader architecture to protect trusted libraries because the bootstrap class loader is always able to attempt to load types before the standard extensions class loader, which is able to attempt to load types before the class path class loader, which is able to attempt to load types before the user-defined class loader and so on. This prevents the mobile code attempting to download a type across the network with same name as something in the Java API. In other words, it is not possible to download a class file for *java.lang.Integer* across the network as long as it exists in the local Java API because it will be loaded locally by the bootstrap loader. Untrusted codes cannot replace trusted classes with their own versions.

Another threatening scenario is when an untrusted code tries to add new type into a trusted package. Let us assume that a user-defined class loader manages to download and define a type named *java.lang.Outlaw*. Java allows classes within the same package to grant each other special privileges that are not enabled outside the package. Because the new type *java.lang.Outlaw*, by its name, declares itself to be a part of the Java API, the first assumption is that it will get the same privileges as the rest of the types that belong to the *java.lang* package. However, this is prevented by using separate name spaces for different class loaders, as defined earlier. The user-defined class loader, which is used for downloading the *java.lang.Outlaw,* has a distinct name space from the bootstrap class loader that locally loads the trusted *java.lang* package. Therefore, *java.lang.Outlaw* is not able to see any of the trusted types in the locally loaded *java.lang* package and vice versa. They do not belong to same *runtime package,* which is defined as the set of types that are loaded with the same class loader [20].

In addition to providing separate name spaces for classes and protecting the borders of trusted libraries, class loaders place each loaded class into a protection domain, which defines, as described earlier, what permissions the code is going to be given as it runs.

The class file verifier

The purpose of the class file verifier is to ensure that class files have a proper internal structure and are consistent with each other. A problematic class file causes the class file verifier to throw an exception. The main reason for verifying class files after loading is that JVM does not know how the particular

class file was generated and, as a consequence, there must be a technique for detecting the possibility of an ill-bred class file. The class file verifier has a rather large impact on program robustness because not only intentionally dangerous class files but also class files that are generated with a buggy compiler are detected.

The class file verifier's operation is four-pronged:

1. *Structural checks on the class file*

   The internal structure of the class file is checked to make sure it is safe to parse. (for example every class file must start with the same four bytes, "magic number", 0xCAFEBABE)

2. *Semantic checks on the type data*

   The verifier ensures that individual components are well-formed instances of their type of component. This is done without looking at the bytecodes. In addition, in this phase it also checks that the class itself fits the specifications of the Java programming language.

3. *Bytecode verification*

   The bytecode streams that represent Java methods are comprised of a series of one-byte instructions called *opcodes,* each of which may be followed by one or more *operands*. The JVM performs a data-flow analysis of each method, ensuring that all method and local variable accesses and invocations are done using values of appropriate types and arguments.

4. *Verification of symbolic references*

   Pass four is part of the process of dynamic linking of a class file. A class filecontains symbolic references to other classes and their fields and methods, and dynamic linking is the process of resolving these links into direct references. During the resolution, the JVM finds the class being referenced - loading it, if necessary - and replaces the symbolic reference with a direct reference. Pass four ensures that the reference is valid.

3.3.3. JVM's responsibility in Java security

JVM has many built-in security mechanisms that are a vital part of Java's robustness and security. From a security point of view, the most important features of JVM are [20, pp 59–61]:

- type-safe reference casting
- structured memory access (no pointer arithmetic)
- automatic garbage collection
- array bounds checking
- checking references for *null.*

Each of these features enhances security by minimizing the possibility of executing a corrupted code. In addition to JVM's internal architecture, that is specified quite abstractly, the unspecified manner of runtime memory areas also increases security. A Java class itself does not appoint any specific memory addresses but, when loading the class file, JVM decides where in its internal memory to put bytecode and associated data it parses from the class file. Thus a malicious user cannot look at the bytecode and predict where in the memory the data representing the class will be kept. In addition to that, it is not possible to gather any information about the memory layout of the JVM just by reading the virtual machine specification because these issues are left to the JVM's implementers.

Besides the features described above, JVM supports *exceptions* for error handling. This structured error handling mechanism contributes to security because when security violation (or other error situation) occurs, instead of crashing the program, the JVM can *throw* an exception or error - which may kill the offending thread but, in most cases, should not crash the whole system.

What must be recalled is that all the security features defined above are only valid when dealing with bytecodes written in Java and compiled with a well-designed Java compiler. When a Java program calls a *native method* - a method written with non-Java language - Java's security model is useless. The security model for native methods is simple: the native method must be trustworthy

before it is called. This approach is traditionally used with all binary, non-interpretable code. Basically, when binary code is accepted it gets full access to the system resources within the restrictions declared for the current user by an underlying operating system.

3.3.4 The security manager

While the class loader, class file verifier and safety features built into Java are intended to enhance the internal integrity of the JVM instance and application it is running, the security manager acts as a central point for access control. The security manager works within a running JVM and controls access to external resources. It defines the outer boundaries of the sandbox and, because it is customizable, a custom security policy can be defined for the application. The Java API supports the custom security policy by asking the security manager for permission before it takes any action that can be considered as unsafe. Asking for permission is done by invoking *check methods* on the security manager object - for example, the *checkWrite()* method determines whether or not a thread is allowed to write to a specified file. Use of these methods defines the custom security policy of the application. Prior to Java version 1.2 these check methods were the only way of establishing a custom security policy because *java.lang.SecurityManager* was an abstract class and had to be implemented - in other words, the developer had to write his own security manager by subclassing the abstract *SecurityManager* class. While providing flexibility, this customizability of the security manager is a potential security threat because writing an own security manager is a difficult task and includes many pitfalls that can lead to security holes at runtime [20, pp. 62–68]. Version 1.2 introduced concrete implementation of the *SecurityManager* class and allows the developer to define a custom policy in an ASCII file instead of in a code. The policy file was presented in Chapter 3.3.1.

3.3.5 The protection domain and access control mechanism

A domain, as defined earlier, is a set of objects that are accessible to a principal - i.e. an entity in a computer system to which authorizations are granted – and, therefore, Java's sandbox is, in a sense, a protection domain with a fixed

boundary [22]. Permissions are not granted to classes and objects directly but they belong to protection domains to which permissions are granted. This is depicted in Figure 22. The protection domain is defined with a policy file, the structure of which is defined in 3.3.1. The class loader gets information about the signer and the codebase from the policy file and creates a *CodeSource* object.

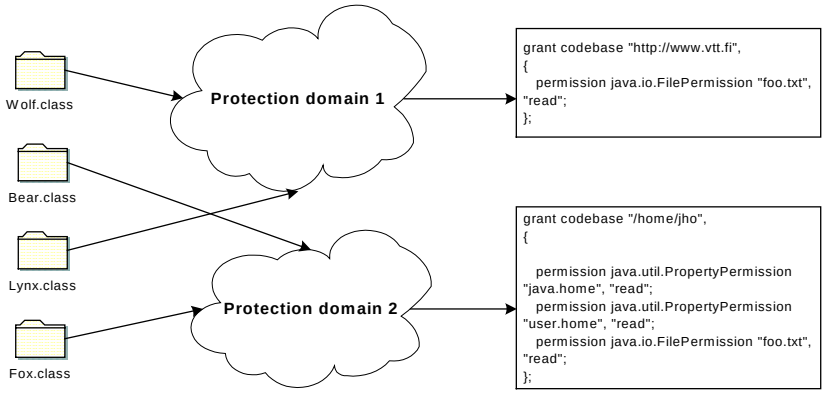


Figure 22. Mapping a class to a protection domain.

The access controller is responsible for enforcing the default security policy mechanism that uses stack inspection to determine whether potentially unsecure action should be permitted. The class *java.security.AccessController* provides this functionality and is not an object but a collection of static methods wrapped to one class. The method *checkPermission* that was mentioned earlier is a member of the *AccessController* class. This is the most important because of its responsibility for deciding whether the particular action is allowed or not. If permission is granted, *checkPermission()* simply returns without a return value, but if permission is denied, an *AccessControllerException* is thrown. Java's default security manager always calls the access controller's *checkPermission()* method and, therefore, the access controller is practically responsible for every threatening action that is taken. The access controller's *checkPermission()* method ensures that every stack frame has permission to perform a threatening action. Figure 23 illustrates how every stack frame is indirectly associated with each set of permissions. The stack is inspected from top to bottom and when the access controller encounters a frame without permission, an exception is thrown.

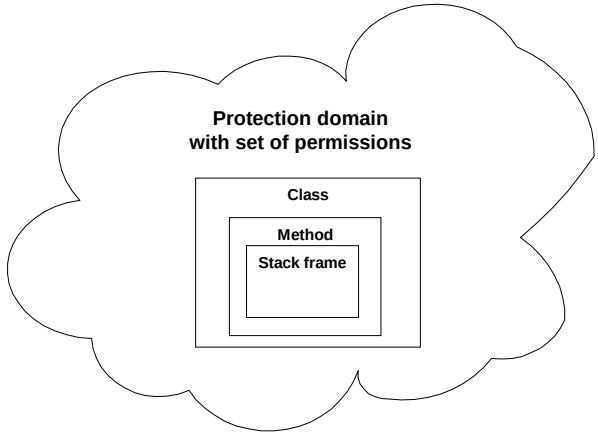


Figure 23. Associating stack frame to set of permissions.

3.4 Security management in Java

3.4.1 Code signing and authentication

An important part of Java's security model is the support for authentication (since Java 1.1). Authentication allows the developer to establish multiple security policies by making a sandbox that has different privileges, depending upon who has signed the code. Thus more trust enables more privileges to the application. Authentication, as defined in Chapter 2.1, allows the receiver to verify that the code has come from a trusted source and that the class file itself has not been altered by some malicious third party.

Java's authentication is based on public key infrastructure and is described in more detail (along with hash codes and digital signatures) in Chapter 2.4. Each file, class file or associated data file must be placed into a *JAR file*, which is a platform-independent file format that collects multiple files into one. This makes downloading of applet and associated files more efficient than loading files one by one. Since Java 1.2, JDK has had a tool called *jarsigner* that is used to sign the entire JAR file. The signing and authentication processes are depicted in Figure 24. First, *jarsigner* generates a hash code of the contents of the JAR file. Second, the resulting hash will be signed using the developer's private key. Finally, the outcome of the process - the encrypted hash code, i.e. digital

signature - is added to the JAR file. Anyone receiving this signed JAR file can authenticate it, ensuring that it has not been altered en route to the receiver and that it really was signed by the claimed developer (the latter is possible only if the sender and the receiver share the same certification authority). Signing and authentication in JDK1.2 are done with the same tool, *jarsigner*.

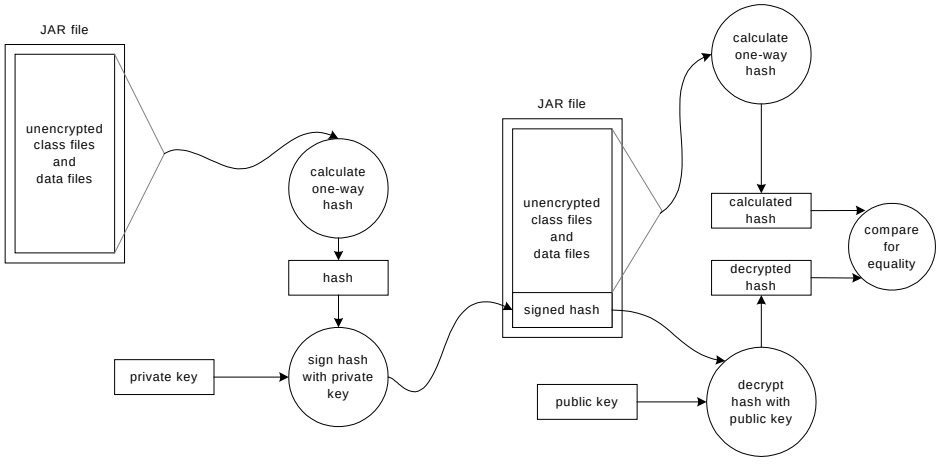


Figure 24. Digitally signing and authenticating a JAR file.

3.4.2 JDK's security-related tools

The Java development kit has three useful tools that help to set security policies, and manage keys and applications: *keytool* for key and certificate management, *jarsigner* for generating signatures as described above and *policytool* for the creation and modification of policy files.

Keytool is a command line tool that enables users to manage their keypairs and certificates that are stored in *keystore*. Sun Microsystems provides built-in implementation of *keystore* named JKS. Each key is protected with a password - as is the entire keystore's integrity. JKS *keystore* supports multiple keypair generation and digital signature algorithms via *service provider interface (SPI)*. The default keypair generation algorithm is DSA and with that signature algorithm is SHA1withDSA. If the keypair generation algorithm is RSA, the signature algorithm is MD5withRSA by default.

Keytool handles X.509 certificates versions 1, 2 and 3 and is able to generate certificate signing requests which are sent to the certification authorities for signing. The certification authority returns the issued certificate (or chain of certificates), which is then imported to a keystore.

Jarsigner is a tool for signing Java archive files. Signing and authenticating the jar file is described in the previous chapter. *Jarsigner* uses key and certificate information from a specified keystore. The signed jar contains a copy of the certificate for the public key corresponding to the private key used for signing. Furthermore, *Jarsigner* is capable of verifying signed jar files. With a signed jar file it is possible to assign different privileges to applications that are signed with different keys.

The policy tool is a graphical tool for specifying, generating, editing, exporting or importing a security policy without need to know about the syntax of the policy file. A used keystore can be defined to find the information specified in the *SignedBy* part of a policy file. Figure 25 illustrates a screenshot from the policy tool with the policy entry window opened. In the policy entry window it is possible to create new policies, or modify existing ones, by defining the codebase and signer and adding, editing or removing permissions.

Figure 25. Screenshot from the policy tool.

4. Middleware protection profile for the networked home environment

4.1. Protection Profile (PP) Overview

The middleware protection profile was constructed in this work. This defines the minimum functional security requirements for middlewares used in a home environment where multiple computing platforms and physical networks, both wireless and wired, exist and handle information which can partly be defined as sensitive. The protection profile presented here adopts some terminology and notation from certified protection profiles (e.g. Controlled Access Protection Profile [23]). The home network is connected to a public network (i.e. the Internet) via a secure gateway. The middleware of this PPs scope is capable of advertising and registering new services, and authenticating and authorizing users - which, in this system, can be considered as humans or services that use other services in the virtual home environment (VHE). It is important to note that ITEA's [24] definition of VHE differs notably from 3GPP's (3rd Generation Partnership Project) definition - which specifies VHE to be "a concept for personal service environment portability across network boundaries and between terminals." [25]. In the sense of the ITEA project, VHE is defined as a networked platform for home appliances allowing plug-and-play communication and shared communications between in-home appliances and external services supporting mobile and stationery terminals with a variety of user interfaces.

4.2. Target of evaluation (TOE) description

The purpose of the middleware is to advertise and provide services in a reliable and secure way, both from the in-home network and from public networks via proper authentication. The middleware hides the underlying operating system, physical network and transport media from the application and provides security services to it. The home network is dynamic in the sense that clients can register with the network and unregister from it at any time.

The middleware defined in the ITEA-VHE project allows users to do the following tasks:

1. Build a home network by just switching them on and, optionally, plugging them into a wired network
2. Control in-home appliances with various terminals having different kinds of user interfaces
3. Access networked home appliances from the outside world through public networks
4. Personalize in-home and external services.

The main concepts of the VHE are described in Figure 26, which also depicts the boundaries of the in-home network and public networks. Users of the TOE consist of human users and applications or services that have registered (or are attempting to register) themselves with the network. A mobile phone or other client that the human user utilizes to access registered services can also be considered users of the VHE whether they provides service(s) or not. The VHE client is identified and authenticated and its resources (CPU, keypad, and screen) are examined to provide the appropriate user interface (UI) for accessing the desired service. Users are able to use appropriate services remotely over untrusted public networks. Users are also able to download applications and install services to the network. A service can be updated, removed and installed remotely by the service provider. The service provider can be, for example, the local cable television channel distributor from whom the home owner buys watching time for a limited period and, when the contract expires, the service stops and it is removed from the services list. The service provider can only perform administrative actions on the service it has provided and, therefore, owns.

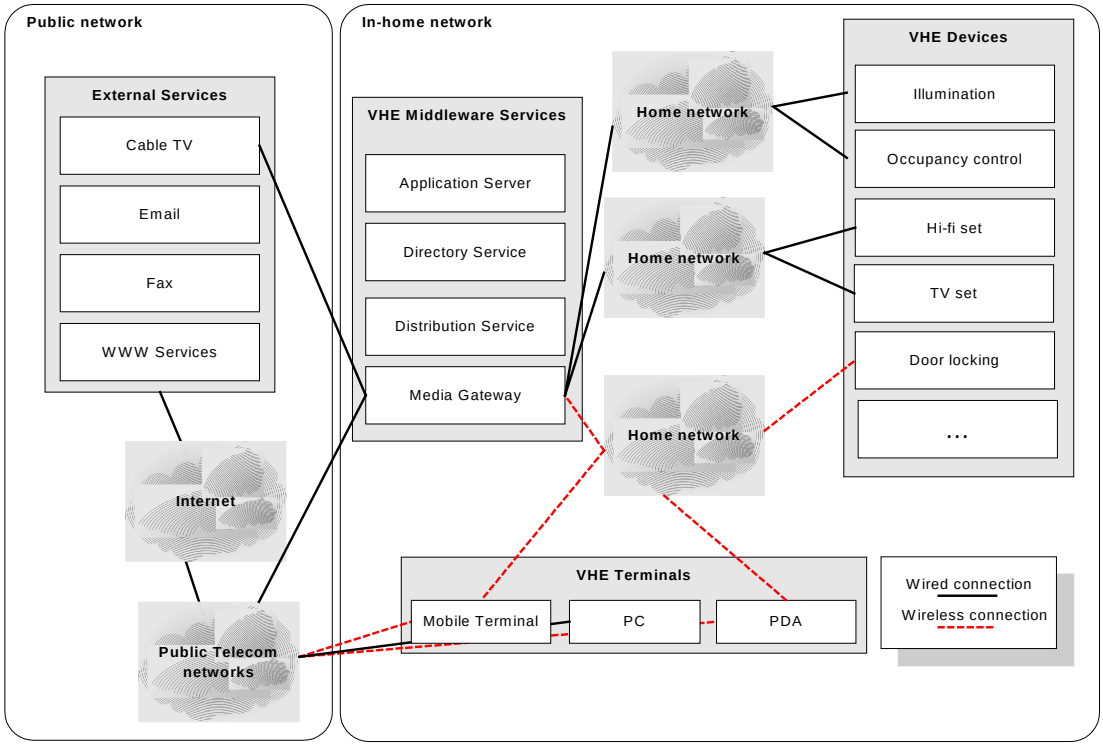


Figure 26. Key concepts of the Virtual Home Environment.

4.3. TOE Security Environment

The middleware must provide some basic security functionality, including user authentication and authorization and encryption of any sensitive information that is sent to the public network. From the application developer's point of view, the security service must be transparent and authentication of new services must be invisible to the human user if possible. The middleware security manager is responsible for the authentication and authorization of any new downloaded service. If it comes from a trusted source (for example, from the network operator), it gains more privileges than an unknown application from an untrusted source. Users are attached to security domains and all users within the same domain have an equal set of privileges. The security manager can optionally generate audit logs of appropriate security-related events in the system and these logs are only readable by the authorized user.

4.3.1. Assumptions

From the security environment's point of view, it can be assumed that the underlying network is properly configured. Also, the reliability and availability of the network are out of security system's scope and are handled by the directory service and underlying infrastructure. Furthermore, it can be assumed that each service and user can be distinguished from the others by using multiple alternative mechanisms.

Very few assumptions regarding the users and their behaviour can be made. An in-home network does not necessarily have a skilled system administrator and the average user is not aware of good security practices and is capable of errors that can lead to compromised security. Therefore, it must be assumed that the more complicated the security system and its administration, the more likely it will be bypassed and left unused if possible. However, it is assumed that users cannot access system resources without proper authorization and all information that flows to and from the public network must pass through the security manager. The security manager is also an arbitrator when making a connection with another client or service in a local in-home network. The authorized administrator of each service can perform administrative actions by accessing the system locally or remotely.

4.3.2. Threats

The threats discussed here are addressed either by the TOE or the environment. The threat agents are either unauthorized persons or external IT entities not authorized to use the TOE itself. Unauthorized use of the system can be either unintentional or an active attempt to harm security. The threats are listed in Table 4.

Table 4. Security threats to the system.

Threat	**Explanation**
T.NOAUTH	Attempt to bypass the security of the system by an unauthorized person.
T.REPEAT	Authentication data may be guessed by an unauthorized user repeatedly to gain access to the system.
T.REPLAY	An unauthorized user may use valid authentication and identification data to gain access to the system.
T.ASPOOF	An unauthorized user from an external network may attempt to disguise authentication data by spoofing the source address.
T.MEDIAT	An unauthorized user may send illegal data through the system, which results in the exploitation of resources in the network.
T.PROCOM	An unauthorized person may be able to view and/or modify security-related information that is sent over the network (between a remotely located authorized administrator and system).
T.AUDACC	An attacker may escape detection because of a lack of audit record reviews.
T.SELFPRO	Modification of critical system security configuration data by an unauthorized user.
T.AUDFUL	An unauthorized person may destroy the audit records or prevent the recording of future records - for example, by exhausting the audit storage capacity (denial of service).

4.3.3 Security policies

Access control policy:

Controlled access protection policy [7] is where every individual user is accountable for its actions. Users have to identify themselves and their identity must be authenticated. Audit trails of security-relevant events, described in Table 6, must be kept. Every user belongs to an appropriate security domain that grants access rights to the user on a need-to-know basis. A separate domain for security-related actions exists.

Information flow policy:

User roles at a general level:

- Home-network administrator
- Service administrator
- Service end-user.

The home-network administrator has the most privileges in the system, so that he is able to perform administrative actions on all services, unlike the service administrator, who is only able to work within the scope of the service and access to other services is granted on a need-to-know basis. The end-user's rights are also restricted to one service by default and no administrative actions can be performed by the end-user. Additional access rights are also granted on a need-to-know basis. The information flow of the TOE is illustrated in Figure 27. A discretionary information flow is mediated by the distribution platform and all data transmission to and from public networks is likely to be encrypted.

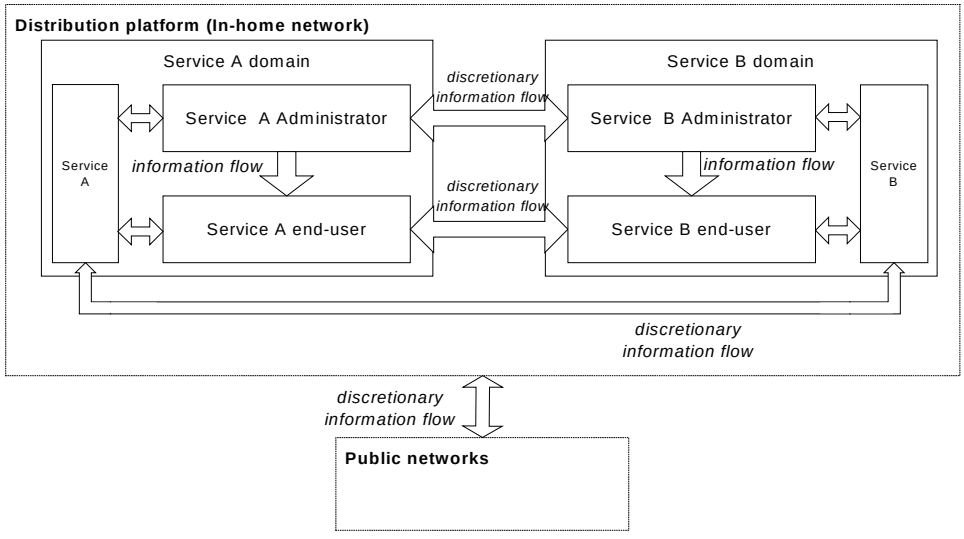


Figure 27. Information flow of the TOE.

4.4. Security Objectives

This section defines the security objectives of the TOE security functions (TSF) and its supporting environment.

Table 5. Security objectives of the system.

Objective	**Explanation**
O.IDAUTH	The System must identify and authenticate the claimed identity of all users before granting a user access to the system. Untrusted users can get restricted rights to use the system.
O.UDOMAIN	Users are divided into security domains based on their roles on the system.
O.SELPRO	The system must be protected against unauthorized users attempting to bypass, deactivate or tamper with security functions.
O.ENCRYPTADM	The connection for remote administration of the system and security-related system data should be encrypted to ensure confidentiality.
O.ENCRYPTUSR	Remote access by a user can be encrypted if desired.
O.AUDREC	The system must provide the means to record audit data associated with an individual user from selective system events with authentic time stamps.
O.SECFUN	The TOE must provide functionality that enables an authorized administrator to use the TOE security functions and must ensure that only authorized administrators are able to access that functionality.
O.MEDIAT	The TOE must mediate the flow of all information between users on an internal network connected to the TOE and users on an external network connected to the TOE.
O.DATAINT	The origin and receipt of data, and the authenticity and integrity of data can be optionally ensured via digital signatures and message authentication codes.
O.SESSIONLMT	The number of concurrent sessions for the same user must be limited to prevent misuse of resources.

4.5. Security Requirements

This chapter presents the functional security requirements divided into the classes defined in Common Criteria version 2.1. At the end of the chapter, Table 7 summarizes these requirements.

Class FAU: Security audit

The audit class defines the requirements for gathering, analyzing and storing the security audit data of security-related events. These collected audit records can be used to examine possible security violations of the system. Each audit record is associated with an individual user, as defined in the access control policy. Secure audit storage is also present in this class to avoid unauthorized reading and modification of audit trails. Each audit record has a reliable time stamp in order to track when the auditable event took place. Every auditable event is recognized here, based on the chosen level of audit data (minimal, basic or detailed).

FAU_GEN.1 Audit data generation

FAU_GEN.1.1 - The TOE Security Function (TSF) shall be able to generate an audit record of the following auditable events:

1. Start-up and shutdown of the audit functions

2. All auditable events for the basic level of audit (all attempted uses of the user security attribute administration functions and basic identification of which user security attributes have been modified)

FAU_GEN.1.2 - The TSF shall, within each audit, record at least the following information:

1. Date and time of the event, type of event, subject identity, outcome (success or failure) of the event.

2. Each audit event type, based on the auditable event definitions of the functional components included in the PP/ST information specified in Table 6.

FAU_GEN.2 User identity association

FAU_GEN.2.1 - The TSF shall be able to associate each auditable event with the identity of the user that caused the event.

FAU_SAR.1 Audit review

FAU_SAR.1.1. - The TSF shall provide [an authorized administrator] with the capability to read [all audit trail data] from the audit records.

FAU_SAR.1.2. - The TSF shall provide the audit records in a manner suitable for the user to interpret the information.

FAU_STG.1 Protected audit trail storage

FAU_STG.1.1. - The TSF shall protect the stored audit records from unauthorized deletion.

FAU_STG.1.2. - The TSF shall be able to ***prevent*** modifications to the audit records.

Table 6. Auditable events.

Functional component	Auditable event
FAU_SAR.1	Reading of information from audit records.
FAU_STG.4	Actions taken due to audit storage failure.
FCO_NRO.1	Identification of the information and destination, and a copy of the evidence provided.
FCO_NRR.1	Identification of the information and destination, and a copy of the evidence provided.
FCS_COP.1	Any applicable cryptographic mode(s) of operation, subject attributes and object attributes.
FDP_ACF.1	All requests to perform an operation on an object covered by the SFP.
FDP_IFF.1	All decisions on request for information flow.
FDP_UCT.1	The identity of any unauthorized user or subject attempting to use the data exchange mechanism.
FDP_UIT.1	The identity of any unauthorized user or subject attempting to use the data exchange mechanism.
FIA_AFL.1	The reaching of the threshold of unsuccessful authentication attempts and the subsequent restoration to the normal state.
FIA_UAU.1	All use of the authentication mechanism.
FIA_UID.2	All use of the user identification mechanism, including the user identity provided.
FIA_USB.1	Success and failure of binding of user security attributes to a subject (e.g. success and failure to create subject).
FMT_MOF.1	All modifications in the behaviour of the functions in the TSF.
FMT_MSA.1	All modifications of the values of security attributes.
FMT_MSA.3	• Modifications of the default setting of permissive or restrictive rules. • All modifications of the initial values of security attributes.
FMT_MTD.1	All modifications to values of TSF data
FMT_SMR.1	Modifications to the group of users that are part of a role.
FPT_STM.1	Changes to the time
FTA_MCS.1	Rejection of a new session based on the limitation of multiple concurrent sessions.

FAU_STG.4 Prevention of audit data loss

FAU_STG.4.1. - The TSF shall *overwrite the oldest stored audit records.*

Class FCO: Communication

The class FCO concentrates on the security requirements of information transportation, non-repudiation of the originator and receipt of transmitted information - i.e. assurance of the identity of both data transmission parties. Basically, non-repudiation means that the originator cannot deny having sent the message nor can the receiver deny having received it. In most cases the identity of the originator or receiver is the identity of the user who sent or received the information.

FCO_NRO.1 Non-repudiation of origin

FCO_NRO.1.1. - The TSF shall be able to generate evidence of origin for transmitted *service proxy* at the request of the *receiver.*

FCO_NRR.1 Non-repudiation of receipt

FCO_NRR.1.1. - The TSF shall be able to generate evidence of receipt for transmitted *service proxy* at the request of the *originator.*

Class FCS: Cryptographic support

This class is used to implement cryptographic functions in the system. These functions include identification, authentication and encrypted data transmission. The FCS class comprises two classes: cryptographic key management (FCS_CKM) and cryptographic operation (FCS_COP). FCS_CKM concentrates on the management of keys while FCS_COP takes care of cryptographic functions - i.e. use of cryptographic keys. Cryptographic operation typically denotes digital signature generation and verification, cryptographic checksum generation and verification (message authentication), data encryption and decryption.

FCS_COP.1 Cryptographic operation

FCS_COP.1.1. - The TSF shall perform *decryption, encryption, digital signatures and message authentication* in accordance with a specified cryptographic algorithm *to be defined later, using widely accepted standards* and cryptographic key sizes of *appropriate length* that meet the following: *standard key lengths accepted by cryptographic protocols (e.g. SSL)*

Class FDP: User data protection

This class specifies requirements relating to protecting user data. FDP is used to construct traditional access control models, e.g. the discretionary access control and mandatory access control that were presented in Chapter 2.3.1. It specifies the access control in terms of *operations,* which can be, for example, "read/write" operations or more complex operations like "update the database". The access control policy is the policy that controls access to the information container. The information flow policy controls access to the information itself, independently of the container. The policies focus on satisfying system's confidentiality, integrity and availability requirements. All objects should be subjected to at least one security policy and the policies should not be in conflict with each other.

FDP_ACC.1 Subset access control

FDP_ACC.1.1. - The TSF shall enforce the *access control SFP defined in Chapter 4.3.3* on *all subjects acting on behalf of the user within TSC.*

FDP_ACF.1 Security-attribute-based access control

FDP_ACF.1.1 - The TSF shall enforce the *access control SFP* on objects based on the following types of subject security attributes:

1. The user identity and domain membership(s) associated with the subject

2. Audit records associated with individual users

3. Other relevant security attributes.

FDP_IFC.1 Subset information flow control

FDP_IFC.1.1 - The TSF shall enforce the *information flow control SFP* on *all subjects acting on behalf of an authenticated user, information and operations that caused controlled information to flow to and from the controlled subjects covered by SFP.*

FDP_IFF.1 Simple security attributes

FDP_IFF.1.1. - The TSF shall enforce the *control information flow SFP* based on the following types of subject and information security attributes:

1. Presumed addresses of source subject and destination subject.

2. Subject's security domain.

3. *Other relevant security attributes.*

FDP_UCT.1 Basic data exchange confidentiality

FDP_UCT.1.1. - The TSF shall enforce the *access control SFP/control information flow SFP* in order to *transmit and receive* objects in a manner protected from unauthorized disclosure.

FDP_UIT.1 Data exchange integrity

FDP_UIT.1.1 - The TSF shall enforce the *access control SFP/control information flow SFP* in order to *transmit and receive* objects in a manner protected from *modification, deletion, insertion and replay* errors.

Class FIA: Identification and authentication

One of the most important security requirements is the identification of the user of the system. The FIA class specifies requirements not only for this but also for verifying the claimed identity of the user. Identification and authentication is required to associate users with appropriate security attributes, e.g. identity, security domain and security role. Unsuccessful authentication attempt scenarios are also met with this class.

FIA_AFL.1 Authentication failure handling

FIA_AFL.1.1. - The TSF shall detect when a *later defined number of* unsuccessful authentication attempts occur related to *authorized TOE entity access.*

FIA_AFL.1.2 - When the defined number of unsuccessful authentication attempts has been met or surpassed, the TSF shall *prevent the unauthenticated entity from successfully authenticating until a later defined unit of time has passed.*

FIA_ATD.1 User attribute definition

FIA_ATD.1.1. - The TSF shall maintain the following list of security attributes belonging to an individual user:

1. *Identity*
2. *Association of identity and authorized administrator role*
3. *Any other relevant security attribute*

FIA_UAU.1 Timing of authentication

FIA_UID.1.2. - The TSF shall require each user to be successfully authenticated before allowing any other TSF-mediated actions on behalf of that user.

FIA_UID.1 Timing of identification

FIA_UID.1.2. - The TSF shall require each user to be successfully identified before allowing any other TSF-mediated actions on behalf of that user.

FIA_UID.2 User identification before any action

FIA_UID.2.1. - The TSF shall require each user to be successfully identified before allowing any other TSF-mediated actions on behalf of that user.

FIA_USB.1 User-subject binding

FIA_USB.1.1. -The TSF shall associate the appropriate user security attributes with subjects acting on behalf of that user.

Class FMT: Security management

The Class FMT is used to define requirements for the management of security attributes in terms of users, subjects and objects. An example of such an attribute is the user role. Management of these attributes can be assigned to an authorized role that is responsible for security-related actions - e.g. reading and deleting the audit trail. Security function management concentrates on access control and authentication functions, and other user security characteristics.

FMT_MOF.1 Management of security functions behaviour

FMT_MSA.1 - The TSF shall enforce the *access control SFP/ information flow SFP* to restrict the ability to *enable and disable* the operation of TOE, audit functions, authentication functions and security management functions.

FMT_MSA.1 Management of security attributes

FMT_MSA.1 - The TSF shall enforce the *access control SFP information flow SFP* to restrict the ability to *modify and delete*

the security attributes *listed in FDP_IFF.1* to an *authorized administrator of the service.*

FMT_MSA.3 Static attribute initialisation

FMT_MSA.3.1. - The TSF shall enforce the *access control SFP/ information flow SFP* to provide *restrictive* default values for security attributes that are used to enforce the SFP.

FMT_MTD.1 Management of TSF data

FMT_MTD.1.1. - The TSF shall restrict the ability to *modify and delete the system clock and other system-related configuration parameters.*

FMT_SMR.1 Security roles

FMT_SMR.1.2. - The TSF shall be able to associate users with roles.

Class FPT: Protection of TSF

This class has some duplicate components with class FDP but is more concentrated on the protection of security-related functions, while FDP took care of user data. This class includes the requirements for executing security-related functions in separate domains, which ensures that the TSF has not been subjected to tampering. Assignment of time stamps is included to enable audit trails to achieve reliable audit records.

FPT_SEP.1 TSF domain separation

FPT_SEP.1.2. - The TSF shall enforce separation between the security domains of subjects in the TSC.

FPT_STM.1 Reliable time stamps

> FPT_STM.1.1. - The TSF shall be able to provide reliable time stamps for its own use.

Class FTA: TOE Access

The class FTA controls the user's session, which is defined as the time between the user identification/authentication and the moment when the user terminates the session by de-allocating all related subjects. Session controlling includes the requirements for limiting the number of sessions for the same user, which can be set for one user domain or individual user.

FTA_MCS.1 Basic limitation on multiple concurrent sessions

> FTA_MCS.1.1. - The TSF shall restrict the maximum number of concurrent sessions that belong to the same user.

Table 7. Summary of functional security requirements.

Functional components	
Audit data generation	FAU_GEN.1
User identity association	FAU_GEN.2
Audit review	FAU_SAR.1
Protected audit trail storage	FAU_STG.1
Prevention of audit data loss	FAU_STG.4
Selective proof of origin	FCO_NRO.1
Selective proof of receipt	FCO_NRR.1
Cryptographic operation	FCS_COP.1
Subset access control	FDP_ACC.1
Security-attribute-based control	FDP_ACF.1
Subset information flow control	FDP_IFC.1
Simple security attributes	FDP_IFF.1
Basic data exchange confidentiality	FDP_UCT.1
Data exchange integrity	FDP_UIT.1
Authentication failure handling	FIA_AFL.1
User attribute definition	FIA_ATD.1
Timing of authentication	FIA_UAU.1
Timing of identification	FIA_UID.1
User identification before any action	FIA_UID.2
User-subject binding	FIA_USB.1
Management of security functions behaviour	FMT_MOF.1
Management of security attributes	FMT_MSA.1
Static attribute initialization	FMT_MSA.3
Management of TSF data	FMT_MTD.1
Security Roles	FMT_SMR.1
Domain separation	FPT_SEP.1
Reliable time stamps	FPT_STM.1
Basic limitation on multiple concurrent sessions	FTA_MCS.1

4.6. Rationale

4.6.1 Security objective rationale

This chapter presents the rationale of objectives and functional requirements, and evidence mappings between threats and objectives, as well as between objectives and requirements. The threats described in this protection profile are meant to be faced with objectives and the corresponding functional requirements. Thus, threats that have no direct solution with the functional requirements presented in CC are omitted. The main reason for this is to maintain consistency between threats, objectives and requirements to form a complete and correct protection profile. One good example of unresolved threats is a scenario where a malicious user tricks people into revealing a password or other information needed for compromising a target system's security. This kind of threat is almost impossible to prevent with technical solutions because it relies on people's ignorance rather than the weakness of the system. Next, Table 8 shows how objectives meet the threats pointed out in Chapter 4.4.2, then the motive for each requirement's existence is rationalized and, finally, dependencies between the security functional requirements are presented.

Table 8. Mapping between threats and objectives.

SECURITY OBJECTIVE / **Threat**	**T.NOAUTH**	**T.REPEAT**	**T.REPLAY**	**T.ASPOOF**	**T.MEDIAT**	**T.PROCO**	**T.AUDACC**	**T.SELFPR**	**T.AUDFUL**
O.AUDREC							X		
O.DATAINT	X	X			X				
O.ENCRYPTADM	X			X		X			
O.ENCRYPTUSR	X			X		X			
O.IDAUTH	X								
O.MEDIAT				X	X				
O.SECFUN	X	X	X						X
O.SELPRO	X							X	X
O.SESSIONLMT					X				
O.UDOMAIN	X							X	

4.7.2 Security functional requirement rationale

Audit data generation FAU_GEN.1

This component outlines what data must be included in audit records and what event must be audited. It traces back to the following objective: O.AUDREC.

User identity association FAU_GEN.2

This component associates the user's identity with audit records to meet the defined security policy associating audit records with an individual user. This traces back to the following objective: O.AUDREC.

Audit review FAU_SAR.1

This component ensures that the audit trail is understandable and meets following objective: O.AUDREC.

Protected audit trail storage FAU_STG.1

This component adds the requirement which states that the audit trail must always be protected from tampering. Only the authorized administrator is permitted to do anything to the audit trail. This traces back to the O.AUDREC objective.

Prevention of audit data loss FAU_STG.4

This component makes certain that the audit trail does not become full and that the oldest audit records will be overwritten so that resources will not be compromised because of full audit trail storage. Furthermore, this component is responsible for ensuring that no other auditable events than those defined in FAU_GEN.1 occur. This component helps to meet the following objectives: O.SELPRO and O.SECFUN.

Selective proof of origin FCO_NRO.1

This component ensures that receipt of a transmitted object can be verified. This component traces back to the following objective: O.DATAINT.

Selective proof of receipt FCO_NRR.1

This component ensures that the origin of a transmitted object can be verified. This component traces back to the following objective: O.DATAINT.

Cryptographic operation FCS_COP.1

This component adds cryptographical functionality to a system to ensure that if the TOE has to support remote administration, all traffic can be encrypted. It also ensures that support for data integrity is ensured by using message authentication codes and digital signatures. It traces back to the following objectives: O.ENCRYPTADM, OENCRYPTUSR and O.DATAINT.

Subset access control FDP_ACC.1

This component accomplishes the use of a security policy in every action taken in the system and specifies the scope of subjects, objects and operations under control. The following objectives are met: O.SELPRO, O.UDOMAIN.

Security-attribute-based control FDP_ACF.1

This component specifies the rules of the security policy and traces back to the following objectives: O.DOMAIN, O.SELPRO

Subset information flow control FDP_IFC.1

This component identifies the entities involved in the information control flow SFP and helps to meet the following objective: O.MEDIAT.

Simple security attributes FDP_IFF.1

This component identifies the attributes of the users sending and receiving the information in the SFP and, furthermore, the attributes of the information itself. The policy is defined by stating the conditions under which information is permitted to flow. This component traces back to, and helps to meet, the following objective: O.MEDIAT

Basic data exchange confidentiality FDP_UCT.1

This component defines the requirement for ensuring confidentiality of user data when it is transferred using an external channel between distinct TOEs. This helps to aid the following objectives: O.ENCRYPTADM, O.ENCRYPTUSR.

Data exchange integrity FDP_UIT.1

This component provides integrity for user data in transit between the TSF and another trusted IT product. At minimum, this means monitoring the data integrity for modifications. This traces back to, and helps to meet, the following objective: O.DATAINT

Authentication failure handling FIA_AFL.1

This component ensures that a user's authentication failures are handled, so that after a limited number of unsuccessful authentication attempts the user has to wait a certain length of time before attempting to authenticate again. This helps to meet the following objective: O.SELPRO

User attribute definition FIA_ATD.1

This component provides users with attributes to distinguish one user from another and associate user-identities with roles chosen in FMT_SMR.1. This traces back to, and helps to meet, the following objective: O.IDAUTH

Timing of Authentication FIA_UAU.1

This component ensures that no security-related operations other than identification are taken before authentication. This traces back to, and meets, the following objectives: O.SECFUN and O.IDAUTH.

Timing of Identification FIA_UID.1

This component ensures that no security-related operations are taken before identification. This traces back to, and meets, the following objectives: O.SECFUN and O.IDAUTH.

User identification before any action FIA_UID.2

This component ensures that before any operation can take place on behalf of a user, the TOE idetifies the user's identity. The following objective is met: O.IDAUTH.

User-subject binding FIA_USB.1

This component associates user security attributes with subjects acting on behalf of the user after successful authentication. This traces back to, and helps to aid, the following objectives: O.DOMAIN, O.IDAUTH, O.SELPRO

Management of security functions behaviour FMT_MOF.1

This component assures that the TSF restricts the ability to modify the behaviour of security functions (e.g. audit trail management). This helps to meet the following objective: O.SECFUN

Management of security attributes FMT_MSA.1

This component ensures that the TSF enforces the SFP to restrict unauthorized modification of security attributes to authorized administrators. This helps to meet the following objectives: O.MEDIAT, O.SECFUN, O.DOMAIN

Static attribute initialization FMT_MSA.3

This component ensures that there is a default security policy for information flow control security rules and this helps to meet the following objectives: O.MEDIAT, O.SECFUN.

Management of TSF data FMT_MTD.1

This component ensures that only an authorized administrator is allowed to modify and delete system-related configuration data and other security-related information. This traces back to, and helps to meet, the following objective: O.SECFUN

Security Roles FMT_SMR.1

Each FMT class component depends on this component and, therefore, this requires the PP writer to choose the roles. This helps to meet the following objective: O.SECFUN.

Domain separation FPT_SEP.1

This assures that the TSF has a domain of execution that is separate and cannot be violated by unauthorized users. This helps to meet the following objective: O.SECFUN.

Reliable time stamps FPT_STM.1

This component is needed by FAU_GEN.1 to gain reliable audit trails with the correct time and date stamps. This traces back to, and helps to meet, the following objective: O.AUDREC

Basic limitation on multiple concurrent sessions FTA_MCS.1

This ensures that one user cannot have an enormous number of concurrent sessions open. This helps to meet the following objective: O.MEDIAT.

Table 9 depicts the mapping between the objectives and functional requirements - that is, how the objectives are fulfilled within individual functional requirements. For example, with the requirement *Subset access control* (FDP_ACC.1), objectives related to users' security domain division and protection against unauthorized use are met as defined earlier in this Chapter.

Table 10 depicts dependencies for functional requirements. Only *directly required* dependency is shown; optional and indirect requirements are omitted. If there is 'X' marked in the cell, it means that those two requirements are dependent on each other - e.g. existence of the requirement *Audit review* (FAU_SAR1) demands that *Audit data generation* (FAU_GEN1) also exists. Each functional requirement is assigned a row and the 'X' in the cell denotes that that column label component is required by the row label component. This table proves that all dependencies are met and all obligatory requirements are present in this profile.

Table 9. Mappings between objectives and functional requirements.

Functional Requirement / **Security Objective**		O.AUDREC	O.DATAINT	O.ENCRRYPTA	O.ENCRYPTUSR	IDAUTH	O.MEDIAT	O.SECFUN	O.SELPRO	O.SESSIONLMT	O.UDOMAIN
Audit data generation	FAU_GEN.1	X									
User identity association	FAU_GEN.2										
Audit review	FAU_SAR1	X									
Protected audit trail storage	FAU_STG.1	X									
Prevention of audit data loss	FAU_STG.4							X	X		
Selective proof of origin	FCO_NRO.1		X								
Selective proof of receipt	FCO_NRR.1		X								
Cryptographic operation	FCS_COP.1		X	X	X						

Subset access control	FDP_ACC.1								X		X
Security-attribute-based control	FDP_ACF.1								X		X
Subset information flow control	FDP_IFC.1						X				
Simple security attributes	FDP_IFF.1						X				
Basic data exchange confidentiality	FDP_UCT.1			X	X						
Data exchange integrity	FDP_UIT.1		X								
Authentication failure handling	FIA_AFL.1								X		
User attribute definition	FIA_ATD.1					X					
Timing of authentication	FIA_UAU.1					X		X			
Timing of identification	FIA_UID.1					X		X			
User identification before any action	FIA_UID.2					X					
User-subject binding	FIA_USB.1					X			X		X
Management of security functions behaviour	FMT_MOF.1							X			
Management of security attributes	FMT_MSA.1						X	X			X
Static attribute initialization	FMT_MSA.3						X	X			
Management of TSF data	FMT_MTD.1							X			
Security Roles	FMT_SMR.1							X			

Domain separation	FPT_SEP.1							X			
Reliable time stamps	FPT_STM.1	X									
Basic limitation on multiple concurrent sessions	FTA_MCS.1						X			X	

Table 10. Security functional requirement dependency table.

	FAU_GEN.1	FAU_GEN.2	FAU_SAR.1	FAU_STG.1	FAU_STG.4	FCO_NRO.1	FCO_NRR.1	FCS_COP.1	FDP_ACC.1	FDP_ACF.1	FDP_IFC.1	FDP_IFF.1	FDP_UCT.1	FDP_UIT.1	FIA_AFL.1	FIA_ATD.1	FIA_UAU.1	FIA_UID.1	FIA_UID.2	FIA_USB.1	FMT_MOF.1	FMT_MSA.1	FMT_MSA.3	FMT_MTD.1	FMT_SMR.1	FPT_SEP.1	FPT_STM.1	FTA_MCS.1
FAU_GEN																											X	
FAU_GEN	X																	X										
FAU_SAR.	X																											
FAU_STG.	X																											
FAU_STG.	X																											
FCO_NRO																		X										
FCO_NRR.																		X										
FCS_COP.																							X					
FDP_ACC.										X																		
FDP_ACF.									X																			
FDP_IFC.1													X															
FDP_IFF.1												X																
FDP_UCT.																												
FDP_UIT.1																												
FIA_AFL.1																	X											
FIA_ATD.																												
FIA_UAU.																		X										
FIA_UID.1																												
FIA_UID.2																												
FIA_USB.																X												
FMT_MOF																									X			
FMT_MSA																									X			
FMT_MSA																					X				X			
FMT_MTD																									X			
FMT_SMR																		X										
FPT_SEP.1																												
FPT_STM.																												
FTA_MCS.																		X										

5. LONTONEXTG Distribution platform

5.1 Introduction to LONTONEXTG environment

The software distribution platform examined here was a result of VTT's research project which aimed to develop a platform that supports the integration of home appliances with an ubiquitous computing environment and enable the development of home automation controlling services. This work developed a security framework for this distribution platform to counter the security threats identified in Chapter 4, along with the functional requirements. The distribution concept itself does not require any particular technology but in the test phase LON-automation and distribution concepts like Jini were observed. Some requirements for the service provider were appointed:

- The service provider (for example, the electricity supplier) must produce services so that the end-user is able to use them remotely - for example, control the home's heating system while travelling by using compatible terminal equipment.

- The service provider must programme the functions of the service using a user interface and make the user interface accessible to the end-user. The service provider is also responsible for setting user rights within its own domain - i.e. the service's scope.

- Various types of services can be made and the service provider must always have adequate knowledge about the target environment concerned.

Figure 28 illustrates the general structure of the system. The end-user is able to browse the services using the *directory service* via terminal equipment. When the desired service is found, the *proxy* of that service is downloaded to the end-user's terminal. The proxy is the client part of the service software. The service itself is usually a more complicated entity that is used via the proxy. In other words, the proxy can be considered to be one view of a service and that view is dependent on the terminal equipment's user interface and computing capabilities, etc. The distributor server lies in the distribution platform, which is located in the In-home network server and its purpose is to distribute and advertise the services it maintains, and provide their proxies upon request. In the *processing*

platform all the services needed in the home environment are performed. The processes are defined as being able to carry out their tasks independently and, furthermore, must provide adequate interface to support the distribution. There are various types of processes that can be performed:

- Controlling of entertainment equipment (TV, Video, etc.)
- Management of other electrical appliances (refrigerator, sauna stove, etc.)
- Controlling of intelligent system (automatic "away from home" mode)

The same process usually provides more than one view of the service, depending on who is using it. The service administrator carries out different tasks than the end-user and they thus need different user interfaces - i.e. views of the system.

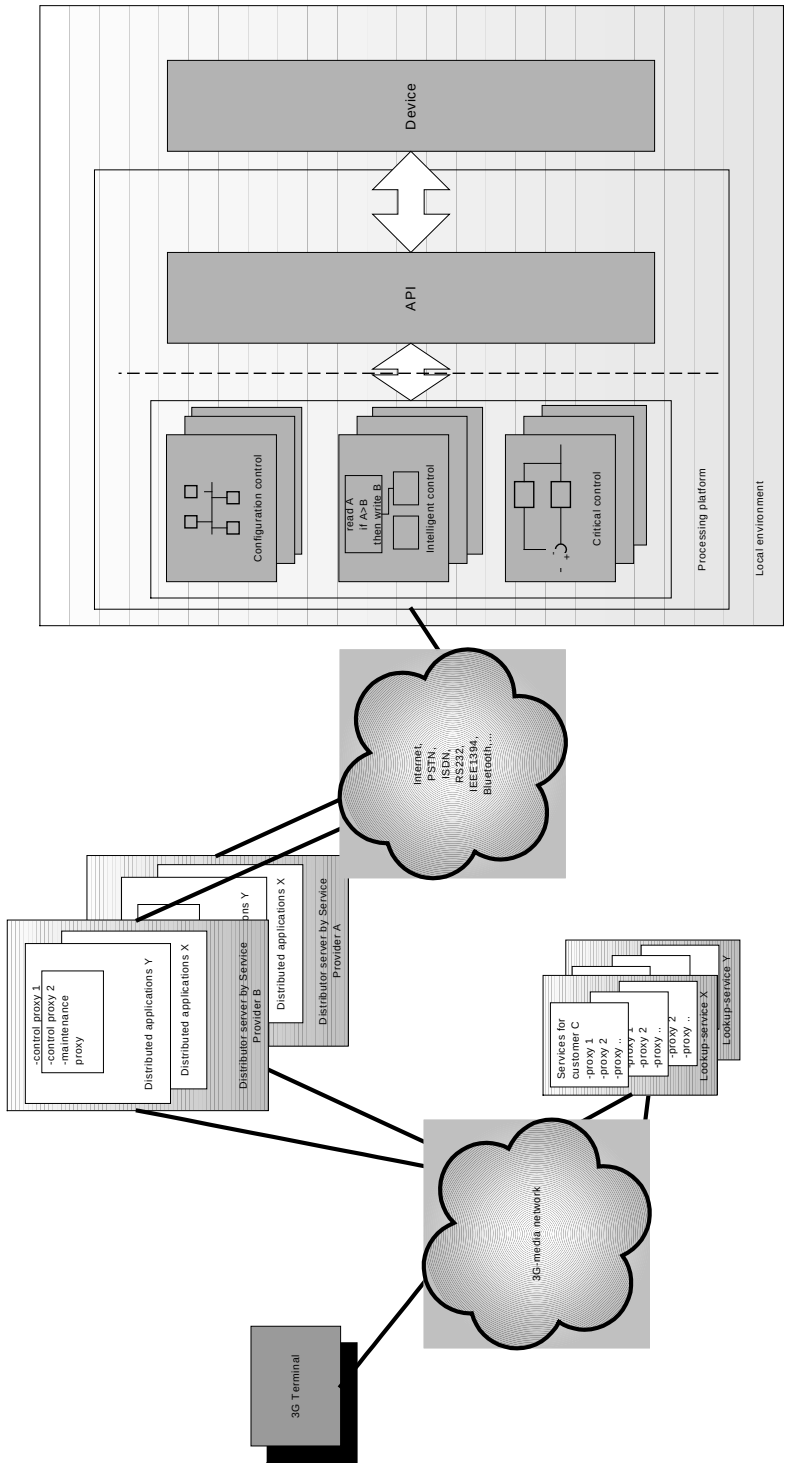


Figure 28. Structure of the system.

5.2 Example service

In this chapter an imaginary electricity supplier's service is defined as a case example to provide an understanding of a real application's characteristics and its security matters, and to observe the different aspects of the presented security framework from the service's point of view. With this service, the electricity supplier is able to observe the homeowner's electricity consumption remotely, without the need to visit the home to read the meter, and make use of this information in charging for the use of the electricity. The homeowner is also able to monitor the household's electricity consumption and, in addition, is able to report some fault conditions - for example, fuses that have blown - directly via the homeowner's mobile phone or other terminal equipment. To make this possible, the service must provide at least two kinds of user interfaces (UIs): one to the electricity supplier and the other to the homeowner. A distinction between the users must be made in order to provide the appropriate UI to the user. In addition to that, there are separate UIs depending on the terminal equipment in question. This is depicted in Figure 29.

In above electricity supplier's service there are various types of users:

- *Service administrator*: the authorized maintenance person from the electricity supply company.

- *Service end-user*: The homeowner who is interested in monitoring the electricity consumption and wants to be informed of any fault condition in the normal use of that electricity

- *Service application developer*: The trusted person who has signed the service application with a private key that has been granted by a trusted CA to present the identity of the electricity supply company.

The application developer does not use the program after it has been installed in an electricity consumption meter. Any updates that might be made to the service software are installed by the authorized service administrator. In other words, the application developer is invisible to the homeowner because it interacts only with the electricity supply company. The service administrator is responsible for every maintenance activity in respect of the electricity meter, starting with

installing the meter. Most of the actions, despite the physical installation and start-up of the service, can be done remotely. The end-user interaction with the service is both remote and local - in other words, within the home network or from a public network outside the home.

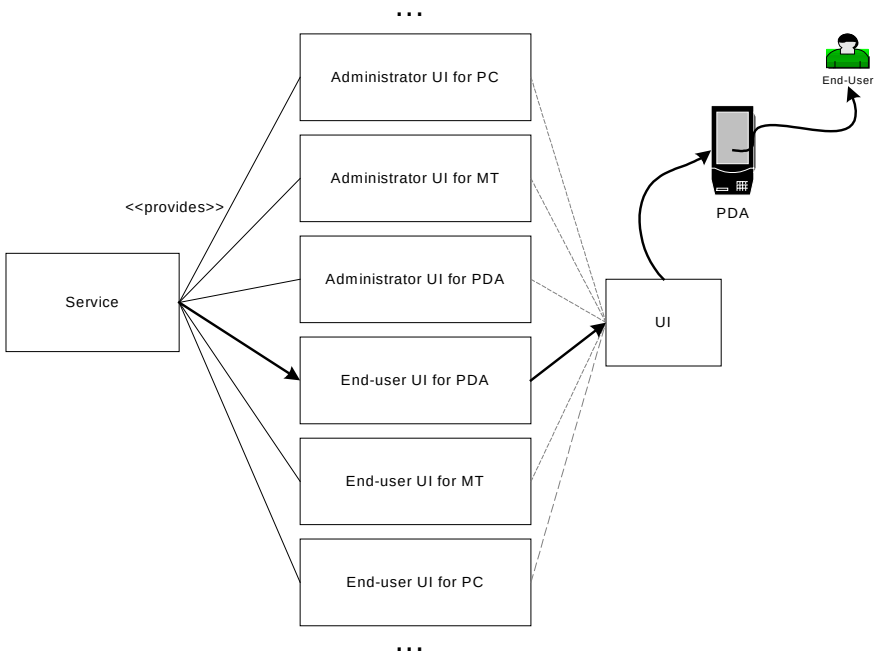


Figure 29. Providing user interface to terminal equipment.

5.3 Security framework of the system

The security framework of the distribution platform extends the distribution concept with security services and security management enforcing the chosen security policies and user domains. The distribution server itself can be started as *trusted* to ensure the authenticity of the security services it manages. This is illustrated in Figure 30. When the distribution server is started as trusted, it authenticates the user, who is the administrator of the distribution server. Checking the administrator's certificate, which is defined as including the public key associated with the user's identity, does the authentication. The security services associated with this trusted distribution service are digitally signed with

the administrator key - in other words, the administrator certifies the authenticity of the security services. In this case, the security services denote the authentication based on the X.509 certificates and key management services. Some of these are only accessible by the administrator. Encrypted connections with SSL are features of the secure distribution platform.

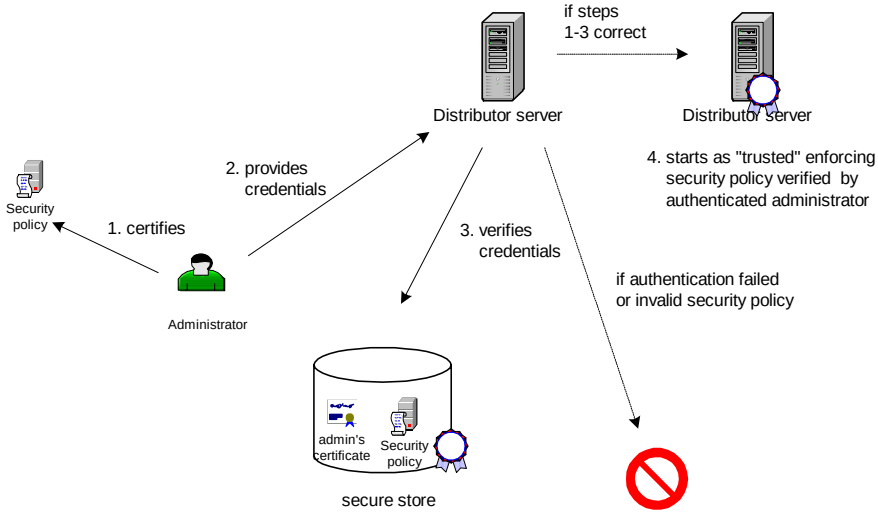


Figure 30. Starting distributor server as trusted.

The administrator's certificate and security policy file are located in the secure store, which is service-specific so that it is encrypted with the administrator's public key. If a user tries to start the distributor server in trusted mode without the proper policy or authentication, or the user fails, the appropriate exception is thrown. The service's additional security policy must not exceed that assigned to the distributor service. An example of Java's policy file for the service is the following:

```
grant signedBy "ServiceAdministrator"
{
   permission java.net.SocketPermission "128.0.0.1:80",
   "accept, connect, listen, resolve";

   permission java.io.FilePermission "<<ALL FILES>>",
   "read";
   permission java.security.SecurityPermission "setPolicy";
};
```

Thus the degree of trust is decided by the signer defined in the policy file. If the signer is unknown - i.e. the signer's public key or certificate does not exist in the secure store - or if the service does not define the signer at all, the service is considered as untrusted and the additional policies are neglected. This is also the case when the signer defined in the policy file is unknown or non-existent. The distributor server's default policy defines the degree of well-known signers' trust.

Figure 31 illustrates different user roles in the system. *Service Provider, VHE Service and Home server Operator* are all special cases of *end-user* and all of these roles expand the end-user's operations within their own role-specific operations. The home server operator is the administrator defined earlier - i.e. the user who starts the distribution server for the service in question. This service is delivered by the service provider who has the right to not only update and install the service but also to start, stop and remove it. This is required in a situation where, for example, the service to be installed is a paid service (e.g. an extension to the electricity provider's service). The service provider is willing to stop the service after the contract between the customer and the electricity supply company expires. An example of a service update requirement could be an online payment application. The service provider - in this case a bank - wants to keep its service as secure as possible and thus wants to be able to update its service and service proxy with new security patches or a more secure SSL client, etc, without the end-user's intervention. If a sufficiently skilled home server operator exists, he may also want to be able to stop or completely remove problematic services, install new ones downloaded from a network or update services with new versions. A new VHE service looks up the service and enforces the security policy dependant on the service's trust. After listing these requirements, it must kept in mind that the end-user's most important requirement is trouble-free and secure use of the service without thinking about administrative issues - thus other roles are meant to help reach this requirement. All user roles are capable of using the service within their own security domain. This means that the service provider is able to perform administrative operations only on the service it owns.

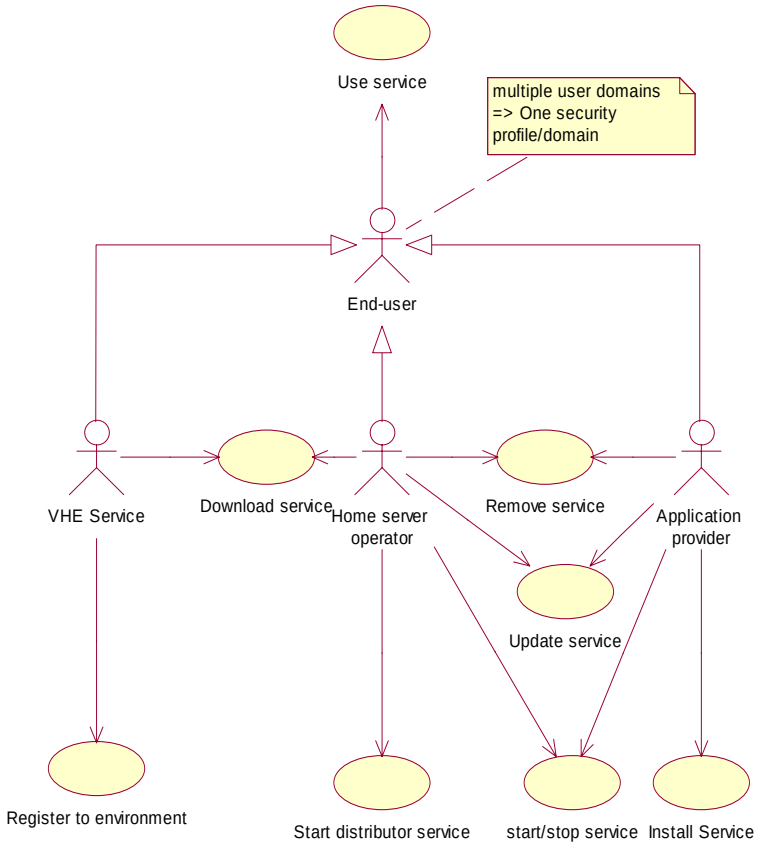


Figure 31. User roles in the system.

6. Discussion

This chapter analyzes the presented work against the research problems that were described in the Introduction. Java as the implementation platform is analysed from the security point of view.

Cryptographic protocols and secured network connections provide the means for creating the secure in-home network. Security is never completely solved with the cryptography, but it is a necessary part of it. Public key infrastructure and mathematical algorithms, presented in this chapter, are widely used and accepted technologies. The networked home environment is a computing environment with many special characteristics and different kinds of applications with different emphases on security requirements. For example, applications that transfer personal data (social security numbers, etc.) require more confidentiality and secure network connections than some other application that has no need for sensitive data transfers across the network. Secure socket layer, described in Chapter 2.6.2, is technology to establish a confident connection between a client at home and a banking service somewhere in the network. The banking application also needs a good authentication mechanism in order to avoid malicious users making illegal money transfers, etc, and these requirements are solved by using, for example, the X.509 certificates presented in Chapter 2.6.1. The scope of users' access rights - i.e. authorization - must be covered by means of user roles in the system. User roles in a networked home environment are more complicated than in a ”normal” office computing environment where, for example, assumptions of skilled administrators are realistic.

Technology itself is not sufficiently adequate to solve security problems in the system - implementation of security policies, services and procedures is also important [4, pp. 79-83]. The security of the networked home environment starts from defining requirements, good security policies and user roles. Good software development practises, adequate documentation and quality processes are also required when reaching the goal of the desired level of security. Common Criteria is a good help when organising and defining the functional and quality requirements of the system. This is discussed in Chapter 2.3.3. It also provides a formal way in which to document threats, assumptions and system requirements with a rationale for each requirement. Implementation of security is made using the technologies presented in the following chapters to create a

system that provides the required confidentiality, integrity and availability. Furthermore, the home network is connected to public networks and many services are controlled by society with laws and regulations. Authentication can be done using an electronic identification card issued by the government. Technically speaking, it has taken on-board the certificate used for authentication but more noteworthy is the fact that society officially emphasizes securing networked transactions. The European Union is very visibly working to come out with common European regulations concerning electronic commerce, encryption standards and content transferred in networks. Society's action in legal issues is required to create secure and trustworthy policies common to all types of networks in the future and to gain users' acceptance of many networked services to come.

6.1 User roles

In Chapter 5.3, four types of users were pointed out:

1. Service provider
2. VHE Service
3. Home server operator
4. End-user.

This derivation of all user roles from the end-user type enables the addition of new user types, if needed, because all users share some common functionality and, in addition, this enforces object-oriented thinking in the development phase. In other words, user types 1–3 are all special cases of end-user as defined in Chapter 5.3 and are extended from the end-user type, each adding its own characteristic functionality and behaviour.

The home server operator is a somewhat vague user type and raises questions that were not solved in this work. It is not physically defined who is able to act as a home server operator but, after saying that, it must be emphasized that it has administrative privileges to all services in the home network. Thus the home server operator is able to start and stop all services if needed, and so on. If the homeowner is skilled enough, it is likely that he will take responsibility for this

role but in practice this cannot be assumed. Another option is that the provider of the set-top box, home computer or other computing platform that takes care of the distribution services is the home server operator. In this case, the home network is just another service that is maintained by the vendor. This melts the service provider and home server operator into the one role and the homeowner does not have to perform any administrative actions on the home network or its services. It must be pointed out that in the last option the home service operator is able to perform administrative actions only on the distribution service, not the other services.

6.2 Functional requirements and security policy definition

Chapter 4 describes the security functional requirements for the networked home environment in an implementation independent way. The form of the chapter follows the Common Criteria standard. During the construction of this *protection profile,* one tool, called *CCToolbox* (version 6.1e), was evaluated. *CCToolbox* constructs the requirements defined in the CC by asking questions to decide whether the requirement is needed or not. Some questions were quite general : "Does the TSF manage cryptographic keys?", while others were confusingly intricate. These questions were not considered to be helping the thinking process during the requirements specification phase because answering the questions can easily lead to absent-minded answering of questions. Eventually, this introduces a long list of meaningless requirements. By comparison, constructing the protection profile manually is an iterative process where every requirement is carefully examined and rationalized. A system under development must be familiar to the developer in order to be able to identify threats to the system and to decide what the security framework is meant to protect against. Table 11 depicts the advantages and disadvantages using *CCToolbox* and, on the other hand, manual reasoning for the construction of the PP.

Table 11. Comparison between automated tool and manual reasoning.

	CCToolbox	**Manual reasoning**
Advantages	• Automated tool for generating the PP as an output from user's inputs. • Takes care of the PP's consistency and completeness.	• The writer has more freedom to define the PP's structure within the required parts of the PP. • The manual thinking process evades meaningless requirements and leads to a more rationalized PP.
Disadvantages	• Hand-made changes to generated PPs are difficult to make. • Using the tool without basic knowledge about the CC can lead to a list of meaningless list of requirements. • The end product of the interview is heavily dependent on how well the questions are understood. Generality of questions can be a problem.	• Needs more knowledge about the Common Criteria than the automated tool. • The writer must be careful to maintain consistency and completeness, which practically requires many iterations during the construction of the PP.

Chapter 4 also defined the access control and information flow policies common to all services. It was done by exploiting the security models and general policies defined in TCSEC. It is an absolute requirement for the distribution platform to enforce common security policies to establish strict boundaries between the services and to assure a safe and secure operation and controlled information flow between the services. The services can define their own additional policies and are enforced if the service's additional policy does not exceed the common security policies. In other words, additional policies are restrictive by nature. Policies are defined by emphasizing the system's user roles to add to the security hierarchy of the user roles.

6.3 Java as the implementation platform

Java provides many advantages for implementing secure networked software. It is the first programming language that has a security model and many APIs are provided for creating authentication frameworks, encrypted connections and cryptographic operations. Java provides relatively advanced tools for managing, for example, different protection domains. This is a very important issue for home environment as it is a combination of many protection domains and user roles. Many applications must have access to some basic services of the distribution platform and, in addition to that, some applications may need access rights to other applications within the home network or outside the home. These protection domains must be carefully examined in order to obtain reliable security policies and strict boundaries between user roles and domains. An example of this is the administrator of the application or service who must have greater access rights than an end-user, but only within the service's scope - i.e. the protection domain where that service belongs. Java also gives good tools for handling certificates and implement applications that use the secure socket layers that were presented in Chapter 2.6.2.

Furthermore, it can be seen that, in the future, Java will be supported by many application vendors. One good example of this is the *Multimedia Home Platform* (MHP) that provides a generic interface to digital applications which are enforced in digital media applications - i.e. digital television, set-top boxes and so on. Up-to-date information about MHP can be found from Reference [26].

There is no bullet-proof solution to security and Java makes no exception. It does have security problems, some of them because of the Java virtual machine and others trace back to, for example, problems of public key infrastructure that cannot be solved at the implementation phase.The idea of setting security policies to applications is simple but, in practice, creating a consistent policy is a rather difficult task and requires security expertise [21]. Questions like who can be a trusted application developer, what roles are in the system and what kind of policies users and applications of the system must enforce still have to be answered - irrespective of the implementation platform and programming language.

However, Java is evolving quite rapidly and the security model comes alongside the Java platform. At the moment, Java can be seen as the most promising platform for networked applications. Although competing technologies exist, none of them integrate all the advantages of Java: platform independence, rather good scalability and top-down security model.

6.4 Characteristics of PKI-based security services

The X.509 authentication framework presented in Chapter 2.6.1 is an example of a public-key-infrastructure-based framework that uses certificates for authentication of the user. X.509 is a widely accepted and used standard, especially in the Internet world, and thus gives a common way for establishing authentication. The Java platform supports the management of X.509 certificates - in other words, it provides tools to generate, display, import and export X.509 certificates with *keytool* utility.

There are some fundamental problems with PKI that must be taken into consideration when constructing, for example, PKI-based authentication services or encrypted connections based on certificates. Because X.509 requires the existence of a certification authority (CA) who is granted permission to issue certificates to guarantee the customer's affability, it must be assumed that this CA is trusted and is capable of storing its own private key securely. The certificate includes a chain of public keys from issuer(s) that are considered as trusted. If a malicious user can add his key to this chain, he can act as a legal certificate issuer. It is also possible to entirely replace an issuer's real public key with the evil one.

Another remarkable question is the use of multiple CAs. A certificate is made to be unique within one CA - for example, it is possible to distinguish two certificate owners with the same name - but this no longer holds with two or more CAs and, furthermore, it is not reasonable to assume that the user knows which CA the certificate is coming from. If the user does not notice where the certificate came from, the user has no way of knowing who the another participant in the communication is.

These problems are mostly caused by a lack of common established practices in certification issuing and they are presumably to be around as long as CAs are working as competing corporations, with varying levels of trust, and without any kind of co-operation – such as cross-certification. After saying that, it must be pointed out that PKIs - especially the X.509 framework - are quite widely used and many services now and in the future — for example in third generation mobile networks — are taking advantage of X.509 certificates, and that is likely to lead to the evolution of well-designed certification authorities and co-operation between them.

7. Conclusions

This thesis concentrated on the networked security issues in a networked home environment. Security threats, technologies and typical user groups of a networked home were examined. Java was inspected from the security point of view as an implementation platform for networked applications and, as a case example, a software distribution platform developed at VTT Electronics was presented. In this work, the security framework for that distribution platform was defined, starting from the requirements specification by using the Common Criteria's protection profile as a starting point for documentation of security environment - i.e. threats, policies, users and requirements.

A networked home environment presents a special kind of computing environment, with many differences compared to a typical office environment. Home users, for example, represent a very heterogeneous group of users: skilled computer users, elderly persons and children. Thus very few assumptions concerning the user's level of expertise can be made. Definition of security functional requirements is possible only when the designer is familiar with the system under development. Security threats must be pointed out because security issues cannot be solved until there is knowledge of the threats that the security framework is meant to answer.

Java presents a relatively promising environment for software developers to come out with more secure networked software. Java's application programming interfaces enforce existing standards for cryptography and authentication frameworks. In addition to this, Java's internal security model is responsible for various security checks inside the runtime environment – including, for example, structural checks of the bytecode.

From the technical side, security threats are rather easy to answer with many existing, mature and mathematically secure cryptographic techniques. Public key infrastructure (PKI) provides a technical solution to ensuring the confidentiality and integrity of the system. One example of such PKI presented in this thesis, the X.509 authentication framework, is a widely accepted standard and is likely to be used in the future as well. The home environment is very complex and is connected to public networks, and this, along with different kinds of users, presents many security issues that cannot be solved entirely by adding the latest

security technologies to the system. The need for common policies for user authentication and a better organized hierarchy of certification authorities is required to prevent the variety of certification authorities competing with diverse trust.

References

[1] Gove, P. (1961) Webster's Third New International Dictionary of the English Language Unabridged. Springfield, Mass, Merriam-Webster. 2662 p.

[2] Common Criteria version 2.1 (Part 1: Intro & General Model, Part 2: Functional Requirements, Part 3: Assurance Requirements). ISO/IEC 15408. URL: http://www.commoncriteria.org (09.03.2001)

[3] Glossary of Computer Security Terms ("Teal Green Book") (1988). National Computer Security Center, NCSC-TG-004. 52 p. URL: http://www.radium.ncsc.mil/tpep/library/rainbow/index.html (09.03.2001)

[4] Fournier, R. (1999) A Methodology for Client/Server and Web Application Development. Prentice Hall, Inc. 648 p.

[5] Security in open systems (1994). NIST Special publication 800-7, US Department of Commerce. 300 p.

[6] Howard, J.D. (1997) An Analysis of Security Incidents on the Internet 1989–1995. Carnegie Mellon University, 246 p.

[7] Department of Defense Trusted Computer System Evaluation Criteria ("Orange Book") (1985). US Department of Defense standard, DoD 5200.28-std. 116 p. URL: http://www.radium.ncsc.mil/tpep/library/rainbow/index.html (09.03.01)

[8] Trusted Network Interpretation of the TCSEC ("Red Book") (1987). National Computer Security Center, NCSC-TG-005. 332 p. URL: http://www.radium.ncsc.mil/tpep/library/rainbow/index.html (09.03.01)

[9] Gollmann, D. (1999) Computer Security. John Wiley & Sons, Inc. 320 p.

[10] Schneier, B. (1996). Applied Cryptography, Second Edition. John Wiley & Sons, Inc. 758 p.

[11] Kaksonen, R. (1997) Salaus ja varmennus sulautetuissa järjestelmissä. Diplomityö. Oulun Yliopisto, Sähkötekniikan osasto, Oulu.107 p .(In Finnish)

[12] Stallings, W. (1999) Cryptography and Network Security, Principles and Practise, Second Edition. Prentice Hall, Inc. 569 p.

[13] Krause, M. & Tipton, H. (1999) Handbook of Information Security Management, Fourth edition. Auerbach Publications. 728 p.

[14] Integrity in Automated Information Systems (1991) National Computer Security Center Report 79-91.

[15] Information Technology – Open System Interconnection – The Directory: Authentication Framework (1993). Recommendation X.509. ISO/IEC 9594-8. 34 p.

[16] Dierks, T. & Allen, C. (1999). The TLS Protocol, Version 1.0. RFC 2246. IETF. URL: http://www.ietf.org/rfc/rfc2246.txt (23.03.01)

[17] SSL Protocol Version 3.0 URL: http://home.netscape.com/eng/ssl3/ssl-toc.html (02.01.2001)

[18] Lindholm, T. & Yellin F. (1999). The Java Virtual Machine Specification, Second edition. Addison Wesley, Inc. 473 p.

[19] The Java Platform, A White Paper, Douglas Kramer, May 1996, Sun Microsystems. URL: http://java.sun.com/docs/white/index.html (09.03.2001)

[20] Venners, B. (1999) Inside the Java 2 Virtual Machine. McGraw-Hill, Inc. 703p.

[21] Chen, E. Poison Java. IEEE Spectrum August 1999 (pp. 38–43)

[22] Gong, L. & Schemers, R. Implementing Protection Domains in the Java™ Development Kit 1.2. In proceedings of the Internet Society Symp. on Network and Distributed System Security, San Diego, CA, March 1998. (pp.125–134)

[23] Controlled Access Protection Profile (1999). Information Systems Security Organization, National Security Agency. URL: http://www.radium.ncsc.mil/tpep/library/protection_profiles/CAPP-1.d.pdf (09.03.2001)

[24] ITEA. URL: http://www.itea-office.org (26.03.2001)

[25] 3G TR 21.905 Vocabulary for 3GPP Specifications (Release 4).Technical Specification Group Services and System Aspects, 3rd Generation Partnership Project. URL: http://www.3gpp.org/ftp/Specs/2000-12/Rel-4/21_series/ (09.03.2001)

[26] What is MHP ?. URL: http://www.mhp.org/what_is_mhp/overview.html (09.03.2000)

ublished by

Vuorimiehentie 5, P.O.Box 2000, FIN–02044 VTT, Finland
Phone internat. +358 9 4561
Fax +358 9 456 4374

Series title, number and report code of publication

VTT Publications 444
VTT–PUBS–444

Author(s)
Holappa, Jarkko

Title

Security threats and requirements for Java-based applications in the networked home environment

Abstract

This work presents the networked home environment from the security point of view. Threats, technologies and the special characteristics of the users are examined. 'Common Criteria' is used in this thesis as a security evaluation criterion to construct a protection profile for the software distribution platform of a networked home environment. 'Protection profile' describes the target of the evaluation - the networked home environment and its security environment, along with access control and information flow policies. This environment sets the context for the security requirements that are established as a result of this thesis to counter the threats that are also identified in the protection profile as a part of the security environment.

Java is a relatively promising platform for the networked software because of its security model, which has evolved since the first versions of Java. Java's application programming interfaces provide support for widely used cryptographic techniques and public key infrastructure frameworks, including the X.509 authentication framework. Java's security features are applied to the software distribution platform developed at VTT Electronics. The security framework for the platform is developed and presented in this work.

'Home', as a distributed computing environment, presents many new issues when compared to typical corporate office networks. Users are very heterogeneous and their needs differ from one to another. The requirements specification must be done with care, and by using knowledge of the system and existing security techniques to develop a system that provides adequate confidentiality, integrity and availability for its users.

Keywords
public key infrastructure, security policy, Java, distributed software, protection profile

Activity unit
VTT Electronics, Embedded Software, Kaitoväylä 1, P.O.Box 1100, FIN–90571 OULU, Finland

ISBN	Project number
951–38–5865–0 (soft back ed.) 951–38–5866–9 (URL:http://www.inf.vtt.fi/pdf/)	

Date	Language	Pages	Price
September 2001	English	116 p.	B

Series title and ISSN	Sold by
VTT Publications 1235–0621 (soft back ed.) 1455–0849 (URL: http://www.inf.vtt.fi/pdf/)	VTT Information Service P.O.Box 2000, FIN-02044 VTT, Finland Phone internat. +358 9 456 4404 Fax +358 9 456 4374