Audit OpenDNSSEC

A.E.T. Europe B.V.

IJsselburcht 3

NL - 6825 BS Arnhem

The Netherlands

Warning Notice

Document Information

Filename: DOC20090514DNSSEC01.doc

Edition 1.0

Project Information:

Document revision history

Version	Date	Author	Changes
1.0	20090514	Jan Rochat	Initial version

Table of contents

1 Introduction 1

2 Scope of the audit 1

3 Findings 2

3.1 The SoftHSM 2

3.2 The PKCS#11 Handling in Signer tools 4

4 Conclusions and possible next steps 5

1 Introduction

Using DNS “as-is” creates the problem that there is no way to check the integrity of the provided information. This may lead to miss-use and jeopardize the security of entire information systems. In order to circumvent this problem a new standard has been introduced to overcome this problem by digital signing the DNS information by the owner of the information. One of the implementations of this standard is implemented and maintained by the opensource project OpenDNSSEC (www.opendnssec.se). One of the steps necessary to see if the product can be used in real life situation is a code review. This document describes the outcome of the security audit conducted by A.E.T. Europe B.V. (AET) on the security related parts of the project.

2 Scope of the audit

This audit concentrated on the source code and only the parts contained in the folders softHSM and signer/tools.

The audit tries to answer the question if the project can be used in a real life situation.

To answer this question the following aspects were addressed during the audit:

- Are the cryptographic primitives applied in the correct way
- Is the program constructed in such a way that it will work in different environments using a real crypto device.

It was not checked if the provided code actually compiles and worked.

3 Findings

3.1 The SoftHSM

File : softHSM/src/bin/softhsm.cpp

File version : /* $Id: softhsm.cpp 609 2009-05-05 11:44:51Z rb $ */

Line 168: soPIN = getpassphrase("Enter SO PIN (4-255 chars): ");

The SoftHSM is using a PIN length of minimal 4 characters, pleas consider using a bigger length e.g. when using a smart card (which has the possibility for a retry counter) the international accepted minimal length is 6 characters.

File : softHSM/src/bin/softhsm.cpp

File version : /* $Id: softhsm.cpp 609 2009-05-05 11:44:51Z rb $ */

Line 363: // We do not use any salt

Pipe *digestPIN = new Pipe(new Hash_Filter(new SHA_256), ...

Hashing a PIN without SALT makes the hashed PIN vulnerable for dictionary attacks.

File : softHSM/src/lib/main.cpp

File version : /* $Id: main.cpp 548 2009-04-29 13:11:04Z rb $ */

Line 441: *pulCount = 14;

Do not use static values if not really needed, in this case the number 14 (which is the number of supported algorithms mentioned several times).

If you have to change the code in the future it is very well possible that one forget to update a value.

Consider using a dynamic lookup (for example by using a sizeof()) or if this is not possible by using a define.

File : softHSM/src/lib/main.cpp

File version : /* $Id: main.cpp 548 2009-04-29 13:11:04Z rb $ */

Line 458 : pMechanismList[2] = CKM_MD5;

The use of MD5 for this use case can no longer be regarded as secure; consider using SHA256 or better.

File : softHSM/src/lib/main.cpp

File version : /* $Id: main.cpp 548 2009-04-29 13:11:04Z rb $ */

Line 492 : **pInfo->ulMinKeySize = 512;**

When building a PKCS#11 implementation on top of another crypto library this information should be retrieved from the underlying abstraction.

3.1.1 General remarks on SoftHSM

The PIN is stored in the database as a hash value. Consider using the PIN as a shared secret e.g. use the value of the PIN to derive a symmetric encryption key (e.g. as described in PKCS#5). This symmetric key should then be used to encrypt a session key which is being used to encrypt the private objects. At this moment having access to the database is enough to get access to the private key.

3.2 The PKCS#11 Handling in Signer tools

File : signer/tools/ldns_pkcs11.c

File version : * $Id: ldns_pkcs11.c 608 2009-05-05 10:54:07Z jelte $

Line 717 :

```
/* TODO: depends on type and key, or just leave it at current
 * maximum? */
CK_ULONG signatureLen = 128;
```

The comment says it all; this should be a dynamic process that uses the key and the chosen algorithm to calculate the size of the expected output. Consider using a RSA private key having a length of at least 2048 bit.

File : signer/tools/ldns_pkcs11.c

File version : * $Id: ldns_pkcs11.c 608 2009-05-05 10:54:07Z jelte $

Line 745 :digest_mechanism.mechanism = CKM_MD5;

The use of MD5 for this use case can no longer be regarded as secure; consider using SHA256 or better.

3.2.1 General remarks on the PKCS#11 Signer tool

In the current implementation the PKCS#11 library provided by the HSM is used as a generic cryptographic library. PKCS#11 is not meant as a generic cryptographic library but as a token abstraction. Which in practice means that there are several types of PKCS#11 implementations. To make sure that more then one type (or even brand) of HSM can be used with the Signer tool you need an abstraction that compensates when a specific cryptographic function is not provided by the PKCS#11 implementation of the HSM used to secure the private key.

If you have a HSM that only supports signing a DigestInfo but doesn't support Digesting, you need an additional Cryptographic device or library to do the digesting.

What applies for cryptographic features also applies for other properties like the need to Logon. Before providing a PIN one should check if this is really needed or not. It may well be that the used HSM has its own KeyBoard and Display and that you need to type in the PIN on this dedicated Keyboard. Another example: It might be that the application using the HSM has to provide the PIN every time it wants to sign something and that the authentication is dropped after signing.

4 Conclusions and possible next steps

The examined implementation is a good starting point towards an implementation of DNSSEC. However to use it in a real life environment at least the following points have to be addressed:

- Tighten the security of the SoftHSM
- Make sure that you use state of the art algorithms
- Apply the algorithms in the correct way
- Add another abstraction that handles different types of HSMs