All your private keys are belong to us

Extracting RSA private keys and certificates from process memory

Tobias Klein
tk@trapkit.de

Version 1.0, 2006/02/05.

1 Overview

This paper discusses a method to find and extract RSA private keys and certificates from process memory. This method can be used by an attacker to steal sensitive cryptographic material. As a proof of concept an IDA Pro plugin as well as an exploit payload will be discussed.

Imagine the following scenarios:

- An attacker gains access to a webserver while exploiting a vulnerability within the webserver service or a vulnerability within the web application. As a result, the attacker is in the same security context as the webserver process. The goal of the attacker is to steal the SSL private key as well as the certificate. As the attacker is in the unprivileged context of the webserver process he cannot reach this information because of the access controls of the filesystem.

- An attacker gains privileged access to a webserver system. The private key is secured by a passphrase. The goal of the attacker is to steal the SSL private key in cleartext.

One solution to solve these problems is to extract the private key as well as the certificate from the webservers process memory. In the following a proof of concept solution will be discussed.

2 Finding private keys and certificates in memory

In 1998, Shamir et al. published a paper called *Playing hide and seek with stored keys* that discusses a method to find cryptographic material within the system memory (see [1]). My approach takes a completely different approach and has therefore nothing to do with the previous work of Shamir et al.

RSA private keys as well as certificates are commonly represented in a standard format. The syntax of the RSA private key information is described in PKCS #8 [2] and the syntax of a SSL certificate is described in x509 v3 [3]. Both, the private key as well as the certificate syntax are represented in ASN.1.

PKCS #8: Private-Key Information Syntax Standard

A private key has the following information syntax (ASN.1):

```
PrivateKeyInfo ::= SEQUENCE {
  version Version,
[..]
```

The following shows the hexadecimal representation of this ASN.1 syntax:

```
30 82 ?? ?? - SEQUENCE (30 82), length of the SEQUENCE (?? ??)
02 01 00    - integer (02), length (01), value (00)
```

As all private keys should be represented in this syntax, we have a pattern to search for.

Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile

A certificate has the following syntax (ASN.1):

```
SEQUENCE {
  SEQUENCE {
[..]
```

The following shows the hexadecimal representation of this ASN.1 syntax:

```
30 82 ?? ?? - SEQUENCE (30 82), length of the SEQUENCE (?? ??)
30 82 ?? ?? - SEQUENCE (30 82), length of the SEQUENCE (?? ??)
```

As all certificates should be represented in this syntax, we have a pattern to search for.

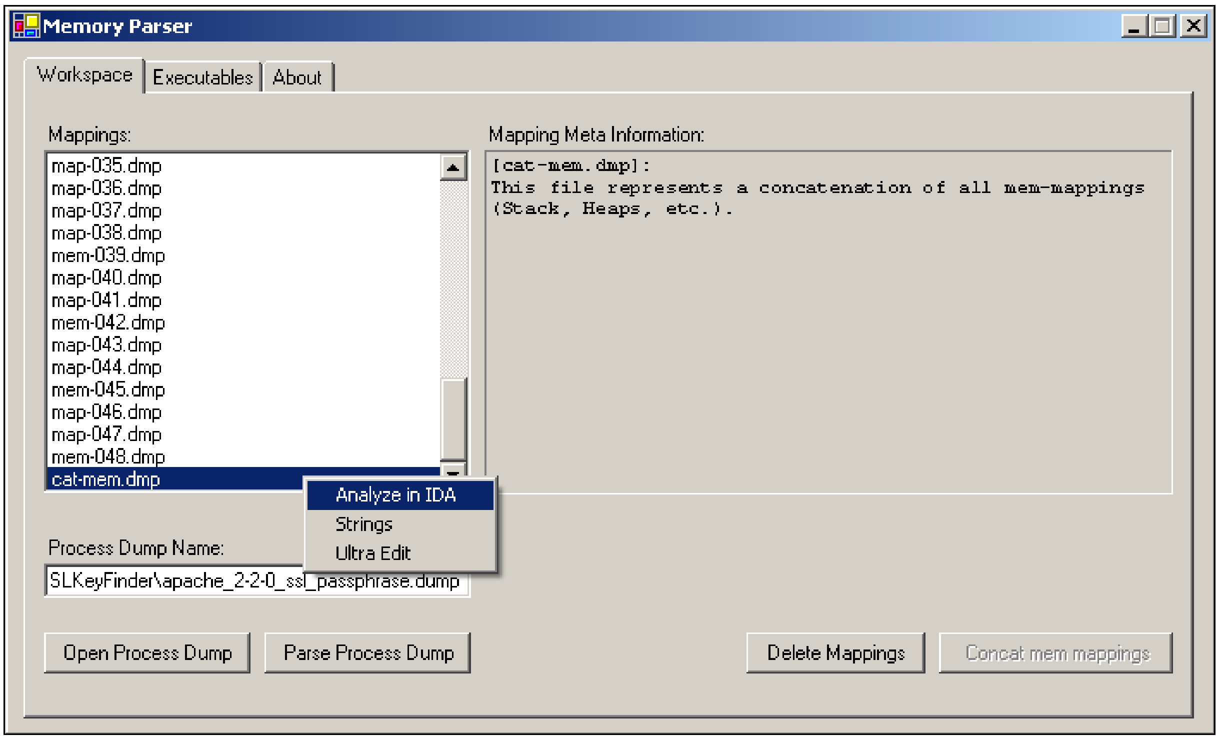
To find private keys and certificates in memory we only have to search for these patterns. Once a pattern is found it is also possible to extract the key or the certificate by interpreting the appropriate SEQUENCE length to get the length of the key or certificate.

3 Implementation

In the following two proof of concept tools will be presented that are capable to extract private keys as well as certificates from process memory.

SSL Key/Cert Finder as IDA Pro plugin

The first implementation is a plugin for the disassembler IDA Pro from Datarescue [4]. In the following example the `pd` [5] utility was used to take a snapshot of an Apache [6] process (version 2.2.0, with SSL support, SSL certificate with passphrase). Then the data mappings (stack, heap, etc.) of the process dump were concatenated with `MMP` [7]. The resulting file with all the data mappings of the process was then searched for RSA private keys and certificates.



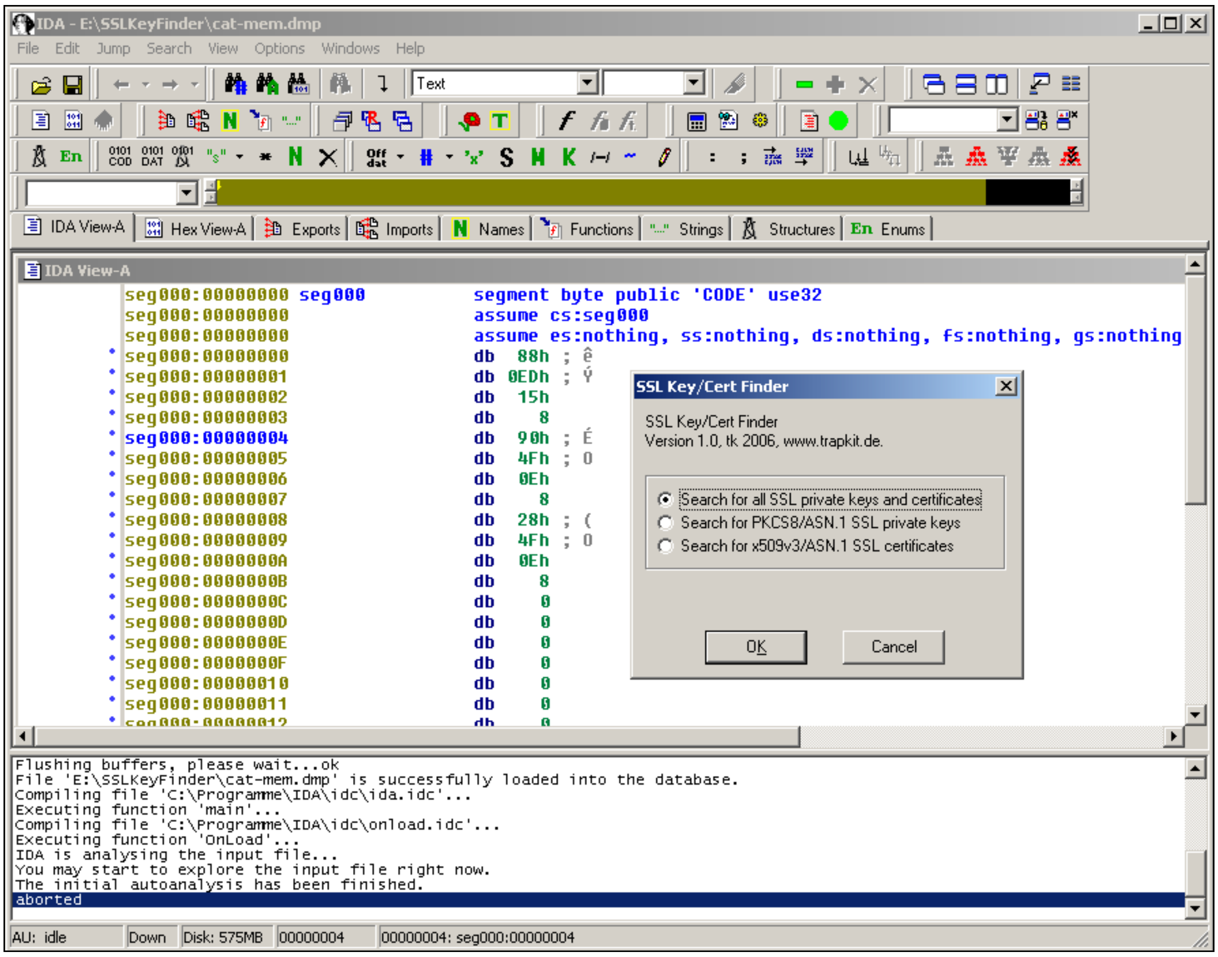
map-035.dmp
map-036.dmp
map-037.dmp
map-038.dmp
mem-039.dmp
map-040.dmp
map-041.dmp
mem-042.dmp
map-043.dmp
map-044.dmp
mem-045.dmp
map-046.dmp
map-047.dmp
mem-048.dmp
cat-mem.dmp

```
[cat-mem.dmp]:
This file represents a concatenation of all mem-mappings
(Stack, Heaps, etc.).
```

SLKeyFinder\apache_2-2-0_ssl_passphrase.dump

Figure 1: Concatenate data mappings with `MMP`

The following screenshot shows the data mapping file of the Apache process loaded into IDA Pro. The SSL Key/Cert Finder plugin can be started with the shortcut `SHIFT + S`.



```
seg000:00000000 seg000          segment byte public 'CODE' use32
seg000:00000000                 assume cs:seg000
seg000:00000000                 assume es:nothing, ss:nothing, ds:nothing, fs:nothing, gs:nothing
seg000:00000000                 db  88h ; ê
seg000:00000001                 db 0EDh ; Ý
seg000:00000002                 db  15h
seg000:00000003                 db    8
seg000:00000004                 db  90h ; É
seg000:00000005                 db  4Fh ; O
seg000:00000006                 db  0Eh
seg000:00000007                 db    8
seg000:00000008                 db  28h ; (
seg000:00000009                 db  4Fh ; O
seg000:0000000A                 db  0Eh
seg000:0000000B                 db    8
seg000:0000000C                 db    0
seg000:0000000D                 db    0
seg000:0000000E                 db    0
seg000:0000000F                 db    0
seg000:00000010                 db    0
seg000:00000011                 db    0
seg000:00000012                 db    0
```

SSL Key/Cert Finder

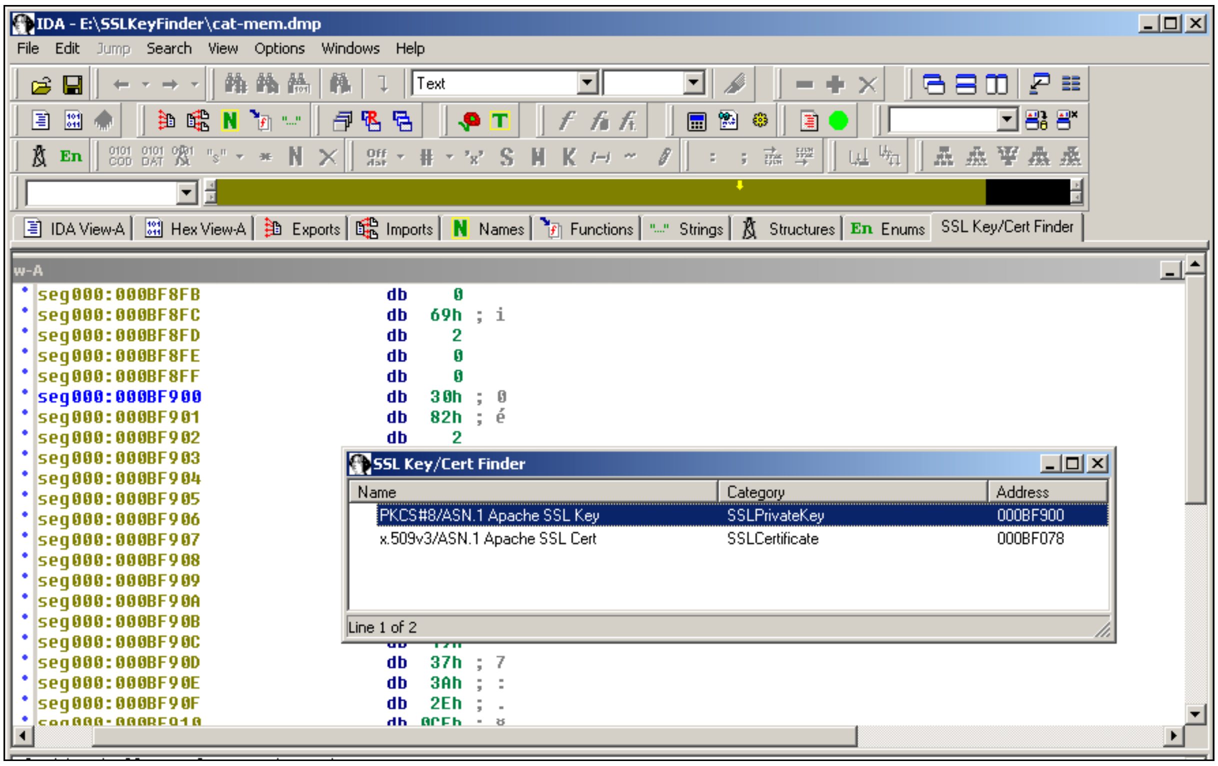
SSL Key/Cert Finder
Version 1.0, tk 2006, www.trapkit.de.

[✓ Search for all SSL private keys and certificates]
Search for PKCS8/ASN.1 SSL private keys
Search for x509v3/ASN.1 SSL certificates

```
Flushing buffers, please wait...ok
File 'E:\SSLKeyFinder\cat-mem.dmp' is successfully loaded into the database.
Compiling file 'C:\Programme\IDA\idc\ida.idc'...
Executing function 'main'...
Compiling file 'C:\Programme\IDA\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
The initial autoanalysis has been finished.
aborted
```

AU: idle | Down | Disk: 575MB | 00000004 | 00000004: seg000:00000004

Figure 2: SSL Key/Cert Finder plugin



```
seg000:000BF8FB                 db    0
seg000:000BF8FC                 db  69h ; i
seg000:000BF8FD                 db    2
seg000:000BF8FE                 db    0
seg000:000BF8FF                 db    0
seg000:000BF900                 db  30h ; 0
seg000:000BF901                 db  82h ; é
seg000:000BF902                 db    2
seg000:000BF903
seg000:000BF904
seg000:000BF905
seg000:000BF906
seg000:000BF907
seg000:000BF908
seg000:000BF909
seg000:000BF90A
seg000:000BF90B
seg000:000BF90C
seg000:000BF90D                 db  37h ; 7
seg000:000BF90E                 db  3Ah ; :
seg000:000BF90F                 db  2Eh ; .
seg000:000BF910                 db 0CEh ; ╪
```

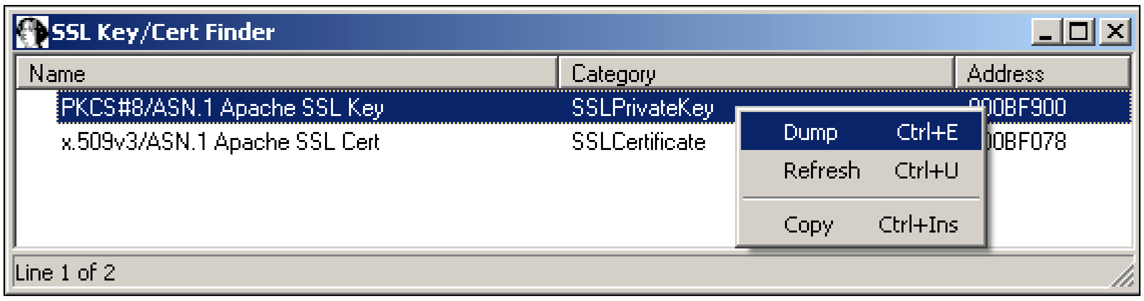
SSL Key/Cert Finder

Name	Category	Address
PKCS#8/ASN.1 Apache SSL Key	SSLPrivateKey	000BF900
x.509v3/ASN.1 Apache SSL Cert	SSLCertificate	000BF078

Line 1 of 2

Figure 3: Found private key and certificate

When right-clicking the found private key or certificate it is possible to extract it to disk.



SSL Key/Cert Finder

Name	Category	Address
PKCS#8/ASN.1 Apache SSL Key	SSLPrivateKey	000BF900
x.509v3/ASN.1 Apache SSL Cert	SSLCertificate	00BF078

Line 1 of 2

Figure 4: Dump the private key from the data mapping file

Now it is possible to use OpenSSL [8] to check if the private key and the certificate are valid.

```
$ openssl rsa -inform DER -check -text -in extracted_key.txt
Private-Key: (1024 bit)
modulus:
    00:be:19:37:3a:2e:cf:2a:6c:fd:8a:44:da:43:d6:
    dd:19:5a:10:5a:f4:bf:93:bb:60:be:1c:24:96:f1:
    40:f5:f1:97:ca:2c:40:ed:dd:85:58:7b:26:68:4c:
    c7:d2:7a:11:82:6f:45:9e:ff:a5:ff:11:ac:da:26:
    7f:6d:9d:90:7f:12:64:ee:03:1b:f9:44:96:c3:3a:
    76:4a:3c:58:9d:f1:32:8b:dc:d2:29:2b:12:89:96:
    a8:b7:fd:5d:b9:7f:76:4c:db:12:e8:b1:33:56:85:
    d3:b2:ed:08:0e:29:7a:05:a3:3e:3c:17:24:69:8d:
    1c:bd:27:8d:b5:38:35:86:c9
publicExponent: 65537 (0x10001)
privateExponent:
    46:f3:c8:6e:39:fc:6e:dc:61:41:93:73:57:f0:c1:
    73:6d:ef:3e:d3:ad:11:a9:d5:70:ff:b6:14:74:95:
    87:76:95:ee:0a:d8:6d:2f:ca:4e:7d:20:97:bb:58:
    b5:d1:83:e9:88:38:97:20:da:47:3a:c4:a6:63:ca:
    1a:12:be:54:59:f2:5d:53:5d:4c:58:70:d1:60:2f:
    ff:1d:7a:c0:37:f7:8d:0d:80:ff:7c:47:8d:8e:92:
    1b:d0:ee:54:cf:5a:b3:b8:d2:0c:6e:bb:31:0c:9b:
    a5:1b:67:92:17:cf:e4:35:9b:0e:d6:e9:30:a0:f1:
    f4:f6:99:64:4e:a6:b9:91
prime1:
    00:f4:59:01:9c:c6:4a:a2:45:f5:af:0b:d9:1d:9a:
    d6:42:6f:d3:ce:56:a3:cb:51:be:39:8f:35:3f:85:
    d3:86:cd:d1:ef:09:29:d7:57:3c:b5:74:3f:91:9b:
    e6:d7:42:a9:13:00:dc:e3:90:73:37:ef:2e:2b:4e:
    a3:64:1b:ed:75
prime2:
    00:c7:29:ef:a0:41:67:1d:56:67:69:0d:9e:73:c6:
    ab:22:a6:28:74:fb:81:62:3b:a9:a7:0d:a8:d8:b7:
    b2:c1:7a:58:c9:c1:4a:0b:db:a9:e0:25:d4:6c:e7:
    49:7f:78:47:9b:24:62:bf:e9:53:26:ac:49:b5:1b:
    92:38:74:65:85
exponent1:
    55:fa:0b:8b:32:6a:88:76:bd:5f:fe:77:42:e7:7c:
    84:9b:fc:97:19:fd:40:49:5e:f9:b9:de:2e:9f:d4:
    32:16:b1:cb:be:19:ae:df:cf:48:b9:c2:b4:65:7a:
    f0:3b:50:6a:93:5f:25:e3:69:e7:40:8d:aa:47:5d:
    4e:98:55:11
exponent2:
    2f:9c:7b:d7:70:ab:28:dd:45:fd:5c:2f:1b:f8:4b:
    63:0e:1b:af:d3:8c:1b:a2:ad:ac:ec:dc:07:6a:ea:
    c5:cb:ec:bb:d6:84:50:0f:64:2d:dc:7d:4a:c7:83:
    cf:80:3e:85:fd:0d:ca:59:09:f2:bd:cf:25:07:81:
    4e:13:ad:4d
coefficient:
    00:c8:12:55:89:1f:5b:bf:52:62:17:bd:b2:a4:dc:
```

```
    02:80:85:2c:be:d3:99:48:03:12:8a:72:27:ac:f1:
    e0:21:29:17:9e:aa:a9:75:b6:5a:5d:91:7d:b4:b1:
    c3:09:47:55:45:fd:2f:d6:17:28:f9:10:dc:de:4a:
    fb:57:a2:81:89
RSA key ok
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQC+GTc6Ls8qbP2KRNpD1tOZWhBa9L+Tu2C+HCSW8UD18ZfKLEDt
3YVYeyZoTMfSehGCbOWe/6X/EazaJn9tnZB/EmTuAxv5RJbDOnZKPFid8TKL3NIp
KxKJlqi3/V25f3ZM2xLosTNWhdOy7QgOKXoFoz48FyRpjRy9J421ODWGyQIDAQAB
AoGARvPIbjn8btxhQZNzV/DBc23vPtOtEanVcP+2FHSVh3aV7grYbS/KTnOgl7tY
tdGD6Yg4lyDaRzrEpmPKGhK+VFnyXVNdTFhwOWAv/x16wDf3jQ2A/3xHjY6SG9Du
VM9as7jSDG67MQybpRtnkhfP5DWbDtbpMKDx9PaZZE6muZECQQDOWQGcxkqiRfWv
C9kdmtZCb9POVqPLUb45jzU/hdOGzdHvCSnXVzy1dD+Rm+bXQqkTANzjkHM37y4r
TqNkG+11AkEAxynvoEFnHVZnaQ2ec8arIqYodPuBYjuppw2o2LeywXpYycFKC9up
4CXUbOdJf3hHmyRiv+lTJqxJtRuSOHRlhQJAVfoLizJqiHa9X/53Qud8hJv8lxn9
QEle+bneLp/UMhaxy74Zrt/PSLnCtGV68DtQapNfJeNp5OCNqkddTphVEQJAL5x7
13CrKN1F/VwvG/hLYw4br9OMG6KtrOzcB2rqxcvsu9aEUA9kLdx9SseDz4A+hfON
ylkJ8r3PJQeBThOtTQJBAMgSVYkfW79SYhe9sqTcAoCFLL7TmUgDEopyJ6zx4CEp
F56qqXW2Wl2RfbSxwwlHVUX9L9YXKPkQ3N5K+1eigYk=
-----END RSA PRIVATE KEY-----
```

```
$ **openssl x509 -inform DER -text -in extracted_cert.txt**
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            f6:26:f7:15:27:7a:ed:ec
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=DE, ST=Some-State, L=HN, O=COMPANY, CN=www.company.net
        Validity
            Not Before: Jan  4 10:21:51 2006 GMT
            Not After : Feb  3 10:21:51 2006 GMT
        Subject: C=DE, ST=Some-State, L=HN, O=COMPANY, CN=www.company.net
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:be:19:37:3a:2e:cf:2a:6c:fd:8a:44:da:43:d6:
                    dd:19:5a:10:5a:f4:bf:93:bb:60:be:1c:24:96:f1:
                    40:f5:f1:97:ca:2c:40:ed:dd:85:58:7b:26:68:4c:
                    c7:d2:7a:11:82:6f:45:9e:ff:a5:ff:11:ac:da:26:
                    7f:6d:9d:90:7f:12:64:ee:03:1b:f9:44:96:c3:3a:
                    76:4a:3c:58:9d:f1:32:8b:dc:d2:29:2b:12:89:96:
                    a8:b7:fd:5d:b9:7f:76:4c:db:12:e8:b1:33:56:85:
                    d3:b2:ed:08:0e:29:7a:05:a3:3e:3c:17:24:69:8d:
                    1c:bd:27:8d:b5:38:35:86:c9
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                BA:A7:CD:BC:B0:B9:C6:CC:10:9C:80:6B:F5:38:DE:20:4F:21:F6:2F
            X509v3 Authority Key Identifier:
                keyid:BA:A7:CD:BC:B0:B9:C6:CC:10:9C:80:6B:F5:38:DE:20:4F:21:F6:2F
                DirName:/C=DE/ST=Some-State/L=HN/O=COMPANY/CN=www.company.net
                serial:F6:26:F7:15:27:7A:ED:EC

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha1WithRSAEncryption
        3f:f4:32:1d:3e:1c:6b:ad:80:03:f4:a1:50:4f:70:a5:01:dd:
        56:ac:54:29:93:ac:d5:ff:6e:97:27:09:a4:7f:57:a2:c5:3f:
        75:2b:1f:69:1e:3d:ae:85:18:c7:43:7f:ad:80:71:f0:a3:8d:
        90:8b:34:b2:a4:dc:a1:da:f0:ce:53:b3:f7:7c:bd:f7:4c:c4:
        36:aa:ec:e3:41:5f:1f:26:ec:ee:e6:78:1e:a8:e9:eb:69:68:
        d5:dd:60:02:44:3d:0a:60:2c:39:2c:69:cf:f8:f5:57:3e:2e:
        85:b9:54:7d:86:f5:1f:ec:69:ab:ff:3e:d5:dc:c0:3e:ea:f2:
```

```
        f5:fb
-----BEGIN CERTIFICATE-----
MIIC9TCCAl6gAwIBAgIJAPYm9xUneu3sMA0GCSqGSIb3DQEBBQUAMFsxCzAJBgNV
BAYTAkRFMRMwEQYDVQQIEwpTb21lLVN0YXRlMQswCQYDVQQHEwJITjEQMA4GA1UE
ChMHQ09NUEFOWTEYMBYGA1UEAxMPd3d3LmNvbXBhbnkubmV0MB4XDTA2MDEwNDEw
MjE1MVoXDTA2MDIwMzEwMjE1MVowWzELMAkGA1UEBhMCREUxEzARBgNVBAgTClNv
bWUtU3RhdGUxCzAJBgNVBAcTAkhOMRAwDgYDVQQKEwdDT01QQU5ZMRgwFgYDVQQD
Ew93d3cuY29tcGFueS5uZXQwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAL4Z
Nzouzyps/YpE2kPW3RlaEFr0v5O7YL4cJJbxQPXxl8osQO3dhVh7JmhMx9J6EYJv
RZ7/pf8RrNomf22dkH8SZO4DG/lElsM6dko8WJ3xMovc0ikrEomWqLf9Xbl/dkzb
EuixM1aFO7LtCA4pegWjPjwXJGmNHL0njbU4NYbJAgMBAAGjgcAwgb0wHQYDVR0O
BBYEFLqnzbywucbMEJyAa/U43iBPIfYvMIGNBgNVHSMEgYUwgYKAFLqnzbywucbM
EJyAa/U43iBPIfYvoV+kXTBbMQswCQYDVQQGEwJERTETMBEGA1UECBMKU29tZS1T
dGF0ZTELMAkGA1UEBxMCSE4xEDAOBgNVBAoTB0NPTVBBTlkxGDAWBgNVBAMTD3d3
dy5jb21wYW55Lm5ldIIJAPYm9xUneu3sMAwGA1UdEwQFMAMBAf8wDQYJKoZIhvcN
AQEFBQADgYEAP/QyHT4ca62AA/ShUE9wpQHdVqxUKZOs1f9ulycJpH9XosU/dSsf
aR49roUYxON/rYBx8KONkIsOsqTcodrwzlOz93y99OzENqrs4OFfHybs7uZ4Hqjp
62lo1d1gAkQ9CmAsOSxpz/j1Vz4uhblUfYb1H+xpq/8+1dzAPury9fs=
-----END CERTIFICATE-----
```

SSL Key/Cert Finder as exploit payload

The second implementation is an exploit payload for the Linux IA-32 platform. This payload was successfully used to extract the private key as well as the certificate from an Apache webserver process memory while exploiting a Memory Corruption Vulnerability within the service [9]. See the source code for further details.

Both proof of concept implementations are publicly available [10].

4 Countermeasures

The only way to secure sensitive cryptographic material is to avoid that it is stored somewhere in memory. This can only be achieved with the use of additional hardware. So called Hardware Security Modules (HSM) provide such functionality.

5 References

[1] Sahmir, A; van Someren, N.: *Playing hide and seek with stored keys*, 1998.

[2] RSA: *PKCS #8: Private-Key Information Syntax Standard*, An RSA Laboratories Technical Note, Version 1.2, Revised November 1, 1993.

[3] Housley, R. et al: *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, Request for Comments: 3280, April 2002.

[4] Datarescue: *IDA Pro Disassembler and Debugger*, *http://www.datarescue.com*.

[5] Klein, T.: *pd – Process Dumper*, *http://www.trapkit.de/research/forensic/pd/*.

[6] Apache Software Foundation: *Apache Webserver*, *http://www.apache.org*.

[7] Klein, T.: *Memory Parser (MMP)*, *http://www.trapkit.de/research/forensic/mmp/*.

[8] OpenSSL, *http://www.openssl.org*.

[9] CAN-2002-0656

[10] SSL Key/Cert Finder implementations: *http://www.trapkit.de/research/sslkeyfinder/*.