Memory analysis and password manager
All your passwords are belong to us

y0ug - Hugo Caron

itrust.lu

9 June 2012

Plan

Introduction

We're going to explain how we manage to recover data from different password manager, by analysing two cases:

- PasswordKeeper with no protection against memory dump
- KeepessX where data are encrypted in memory

Tools

To manage this task we used:

- Volatility and the module
  - memdump
  - procmemdump
  - volshell
- IDA (for KeepassX)

All our tests are done on memory dump of WinXP SP3 in VirtualBox

Information

Description from official website

"Password Keeper is a small utility useful for storing your frequently used passwords. Password information can be stored, edited and printed with this easy to use program."

User interface

Dump creation

In the next slides we work on a system memory dump with a running process of Password Keeper.

To facilitate the password container contains this following data:

- Account: Foo
- Username: Baz
- Password: rqNBikbt
- URL: http://www.bar.baz
- Comment: SuperComment

Recover information

With volatilty we dump the PasswordKeeper processes

```
0xffbc7da0 PassKeep.exe            284   1464      2     38 2012-05-10 06:54:23
0x80fbb250 PassKeep.exe            292    284      3    137 2012-05-10 06:54:23
0x80fb8da0 alg.exe                 920    636      7    104 2012-05-10 06:54:26
0xffaa5020 wuauclt.exe            1212    980      8    173 2012-05-10 06:55:07
0x80f6fda0 wuauclt.exe            1684    980      5    138 2012-05-10 06:55:22
y0ug@laptop:~/passwordkeeper$ vol.py procmemdump --pid 292 --dump-dir dump/ -f dump.raw
Volatile Systems Volatility Framework 2.0
************************************************************************
Dumping PassKeep.exe, pid:    292 output: executable.292.exe
```

And strings our password on it

```
y0ug@laptop:~/passwordkeeper$ strings --radix=x dump/executable.292.exe | grep -B3 -A3 -i rqNBikbt
  2feba .:3q
  2fee3 -64OS
  2ff04 NKeb
  43120 rqNBikbt
  43150 http://www.bar.baz
  431d0 SuperComment
  570bb "Small Fonts
--
  6192c Password Keeper 2000 v7.1
  6198c Copyright
  61997  1996-2008 by Gregory Braun. All rights reserved.
  61a90 rqNBikbt
  61ac0 http://www.bar.baz
  61b40 SuperComment
 161a40 c:\program files\software by design\passkeep
```

Extracting data structure

Information

Data are in clear text, good news.

With more investigation we manage to recover:

- The structure
- A signature to match the position in dump

structure

```
struct S_DATA_PKEEPER{
        char account[48];
        char user_name[48];
        char password[48];
        char url[111];
        char comment[769];
}
```

Signature

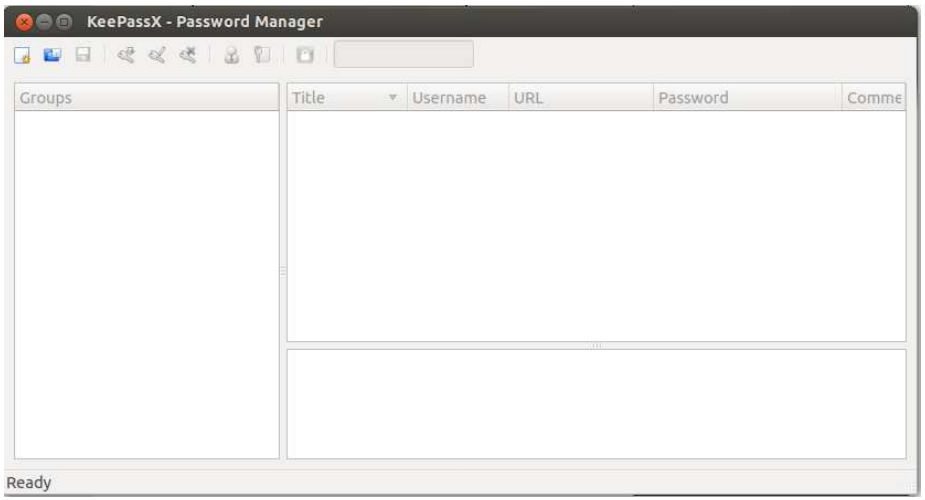
```
00061900  01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00061910  00 00 00 00 00 00 00 00  00 00 00 00 f0 de bc 0a  |................|
00061920  07 00 01 00 79 96 4e 24  01 00 00 00 50 61 73 73  |....y.N$....Pass|
00061930  77 6f 72 64 20 4b 65 65  70 65 72 20 32 30 30 30  |word Keeper 2000|
00061940  20 76 37 2e 31 00 00 00  00 00 00 00 00 00 00 00  | v7.1...........|
00061950  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00061980  00 00 00 00 00 00 00 00  00 00 00 00 43 6f 70 79  |............Copy|
00061990  72 69 67 68 74 20 a9 20  31 39 39 36 2d 32 30 30  |right . 1996-200|
000619a0  38 20 62 79 20 47 72 65  67 6f 72 79 20 42 72 61  |8 by Gregory Bra|
000619b0  75 6e 2e 20 41 6c 6c 20  72 69 67 68 74 73 20 72  |un. All rights r|
000619c0  65 73 65 72 76 65 64 2e  00 00 00 00 00 00 00 00  |eserved.........|
000619d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
000619f0  00 00 00 00 00 00 00 00  00 00 00 00 cb 10 00 00  |................|
00061a00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00061a20  01 00 00 00 bf c4 01 00  00 00 00 00 00 00 00 00  |................|
00061a30  46 6f 6f 00 00 00 00 00  00 00 00 00 00 00 00 00  |Foo.............|
00061a40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00061a60  42 61 7a 00 00 00 00 00  00 00 00 00 00 00 00 00  |Baz.............|
00061a70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00061a90  72 71 4e 42 69 6b 62 74  00 00 00 00 00 00 00 00  |rqNBikbt........|
00061aa0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

Script

With this information we wrote a ruby script to extract automatically all entry:

Introduction

- Is a cross platform password manager.
- On the official website no specifics information about protection against memory dump
- It uses AES cbc to encrypt the file container
- Advantage we've the code



Introduction

We do some classic check with strings to try to recover data in a memory dump of the process:

- No conclusive results
- We've the source code so we use it

```
y0ug@laptop:~/keepassx$ vol.py  -f winxp.dmp memdump --dump-dir ./ --pid 204
Volatile Systems Volatility Framework 2.0
************************************************************************
Writing KeePassX.exe [   204] to 204.dmp
y0ug@laptop:~/keepassx$ strings 204.dmp | grep rqNBikbt
y0ug@laptop:~/keepassx$ strings -el 204.dmp | grep rqNBikbt
y0ug@laptop:~/keepassx$
```

Data loading

With the source code we understand how it loads data from the container in memory:

- Load the file in memory
- Recover information need to decrypt the file like EncryptionIV, ContentHash, FinalRandomSeed etc..
- Check magic, version and flag
- Compute a SHA256 composed of FinalRandomSeed + Masterkey
- Transform the SHA256 of the masterkey
- Decrypt the container with AES CBC, the EncryptionIV and the last compute hash
- Check if ContentHash match the decrypt data

Memory protection

We know that the MasterKey are in `SecData RawMasterKey`

`SecData` is a class that secure data in memory, when the software don't need it, it locks the data (encrypt data with a cutsom RC4)

The variable can be represented like this:

```
class SecData{
        public:
                SecData(int len);
                ~SecData();
                void lock();
                void unlock();
                void copyData(quint8* src);
                void copyData(SecData& secData);
                quint8* operator*();

        private:
                quint8* data;
                int length;
                bool locked;
};
```

Memory representation

```
|   pdata|length|   locked|
|XXXXXXX|     20|00000001|

Length = 32 (SH256 len)
Locked = 1  (Lock flag)
```

RC4 key

To encrypt data in memory it uses the same key for all data (in `class SecString`)
`static quint8 *sessionkey;`

Is a static variable, so it will be at same memory address in each dump.

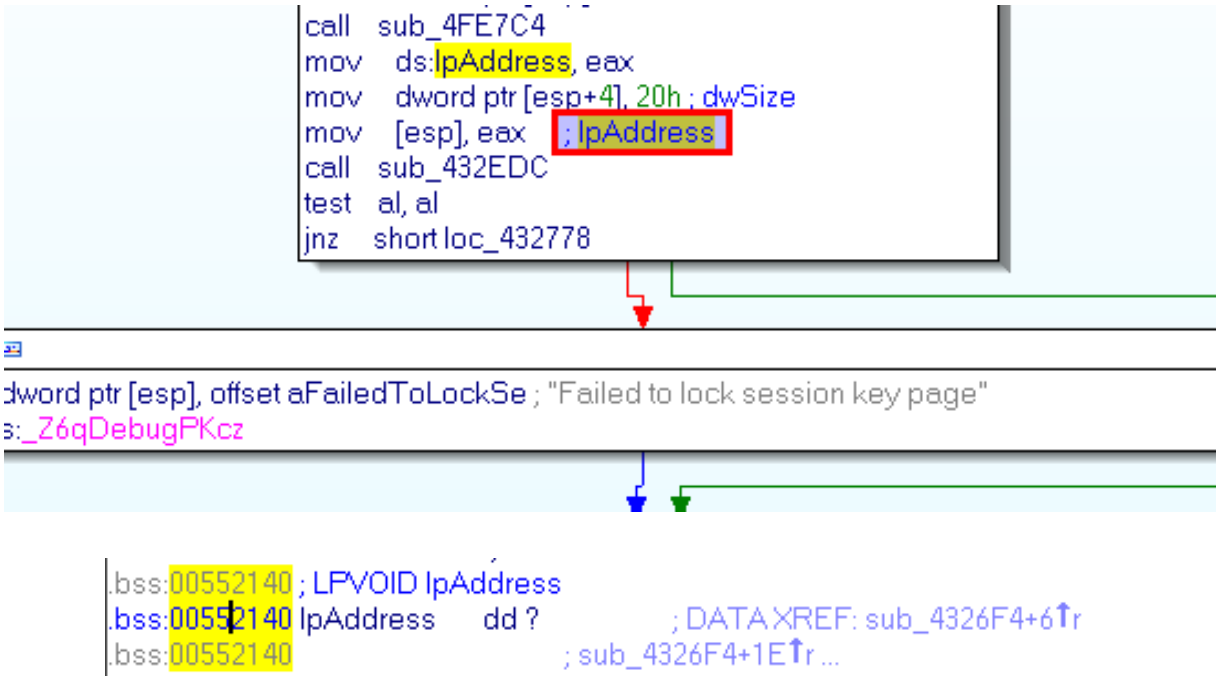
The algorithm used to encrypt data is modified RC4 version

```
private:
        static CArcFour RC4;
        static quint8* sessionkey;
        bool locked;
        QByteArray crypt;
        QString plain;

};
```

RC4 key

To recover the address of: `static quint8* sessionkey` we use IDA with the targeted version of KeepassX.
We've to look in `SecString::generateSessionKey()`:



In our case the address is 00552140h

Summarize

Summarize, we know:

- The file format of kdb
- The address of the RC4 key used to crypt the master key in memory
- Structure to seek to find the pointer to the encrpyted master key
- Algorithm to decrypt the container

So we've all information to recover data

Recover address of encrypted master key

To recover the address of the encrypted master key we used a regex, on the memdump of the process.

```
def adrfinder(data):
    print ' [*] Search for encrypted master key...'
    print '*'*87
    print ' start  -  end   |    adr1    |    adr2    |    match'
    print '*'*87
    guess_adr = None
    for m in re.finditer(b"(.{4})\x20\x00{3}\x01.{3}(.{4})\x20\x00{3}\x01", data):
        if re.match(b"\x00{2}", m.group(1)):
            continue

        adr1 = struct.unpack('<I', m.group(1))[0]
        adr2 = struct.unpack('<I', m.group(2))[0]

        if guess_adr == None:
            guess_adr = adr1

        print '%08x-%08x: 0x%08x | 0x%08x | %s' % (m.start(), m.end(), adr1, adr2, m.group(0).encode("hex"

    print '*'*87
    print ' [*] Encrypted master key (guess): 0x%08x' % guess_adr
    return "0x%08x" % guess_adr
```

The function take in input the memdump of the process, and return the first address that match the regex. The first is valid normally.

Extract the encrypted master key

To extract the encrypted masterkey we used volatilty with the plugin volshell and the address found in the previous slides.

```
y0ug@laptop:~/keepassx$ python Kfind.py 204.dmp
KeepassX adr finder in dump
 [*] Static ARC4 key pointer: 0x00552140
 [*] Search for encrypted master key...
**********************************************************************************
 start   -  end    |    adr1     |    adr2     |    match
**********************************************************************************
00053464-00053479: 0x00c62448 | 0x00c62358 | 4824c60020000000010074005823c6002000000001
0005347c-00053491: 0x00c62638 | 0x00c62660 | 3826c60020000000010065006026c6002000000001
**********************************************************************************
 [*] Encrypted master key (guess): 0x00c62448   <-- Address of the encrypted raw masterkey
y0ug@laptop:~/keepassx$ vol.py  -f winxp.dmp volshell
Volatile Systems Volatility Framework 2.0
Current context: process System, pid=4, ppid=0 DTB=0x312000
Welcome to volshell! Current memory image is:
file:///home/y0ug/keepassx/winxp.dmp
To get help, type 'hh()'
>>> cc(None, 204)
Current context: process KeePassX.exe, pid=204, ppid=1516 DTB=0xbb801a0
>>> db(0x00c62448,32)
00c62448  5b e2 a9 63 35 14 cf 08 93 29 d1 df 74 4b 09 74  [..c5....)..tK.t
00c62458  86 56 43 9a 29 05 ae 2b 65 fd 95 d6 86 b0 25 25  .VC.)..+e.....%%
                                                  <-- Encrypted raw masterkey
>>>
```

Extract the session key

To extract the session we used the same technique as above, with one more step to extract the value of the pointer founds with IDA.

Extract data from the container

We decrypt the masterkey and try to decrypt the password container by following the step define above.

```
y0ug@laptop:~/keepassx$ python Karc4.py 17f2bfb694792ea189d4878ed093a600e4a39fe3ffae4779663998e94171cbe3
5be2a9633514cf089329d1df744b09748656439a2905ae2b65fd95d686b02525
Result:      decrypted raw masterkey
48c5a1d217fe85082464d2ca1e90a16d15464fabe20f8610d79b63aa58797b9b
y0ug@laptop:~/keepassx$ python KdecSHA.py pass.kdb 48c5a1d217fe85082464d2ca1e90a16d15464fabe20f8610d79b63
aa58797b9b
Write decrypted content in pass.kdb.dec
y0ug@laptop:~/keepassx$ strings pass.kdb.dec
Internet
eMail
Foo Baz
http://www.bar.baz
rqNBikbt
SuperComment
```

session key

encrypted raw masterkey

Demo

DEMO

Conclusion

- In this 2 cases we saw, the most current no protect at all and data encrypted in memory.
- We manage to prove that is possible to recover the container or the data from a a memory dump
- From a forensics point of view, this can help during investigation if the authorities have the idea to dump the memory before seizure.
- From an attacker point of view, this techniques can be apply too, with only a dump of the running process. It can be interesting to avoid for the attacker to wait with a key-logger that the target type the password.

Conclusion

Thank you for your attention.

Any question? (or not)