Write Up: Hack.lu 2013 - Internals 200 - Packed

R. Ernst, M. Lambertz

Abstract

This write up gives a short summary for the Hack.lu 2013 CTF challenge "Internals: Challenge Packed".

1 Description

"*We just found a dead robot. It seems there is some useful data left but somehow it got confused with other data and now we don't know what's useful and what's junk. We just know there is only one way to go but there are many dead ends.*" The file "packed" was given.

2 Basic File Analysis

We first tried to figure out what file we got. The common Unix tool for this task is *file* [1]. Unfortunatly, we got the generic answer "data" which means "*Any file that cannot be identified as having been written in any of the character sets listed above is simply said to be "data".*" [2]

```
1 $ file packed
2 packed: data
```

Viewing the file in a hex viewer, e.g.,

```
1 $ xxd packed | less
```

shows multiple blocks

- 168 times \x0c
- *#disable-encoding: rot___...13*
- Random looking data
- PDF data
- A block starting with *pvcure*
- Another random looking block

3 Data Viewing

3.1 PDF

You can open the file with a document viewer like evince [3] which will give you a PDF file showing *no hint given*

3.2 Open Document

We use *dd* to extract the last block, convert it with a *base64* decoder, and run *file* again.

```
$ dd spkip=2273 count=12367 if=packed of=output bs=1
$ base64 -d output > out
$ file out
out: OpenDocument Text
```

You can open the resulting file with LibreOffice [4] which will show *still no hint given.*

3.3 Python Script

The random looking block starting with *pvcure*, contains a Rot13 [5] encrypted Python script. To get the script run

```
$ dd spkip=1270 count=986 if=packed of=output.py bs=1
```

and use a Rot13 conversion tool or webpage on the file *output.py*.
Note the small pitfall after decoding. Rot13 conversion causes the decode functions to use *mvc*. This should be *zip* as it was before the conversion.

You will end up with the following script:

```
cipher="H51\\\'Ux2J&+(3Z;Uxcx0Xxs\x13h\x014$V!R($R>\t/)R!\x01<.\x13,N-
    aP4M4aRuG1-VuU0␣GuH+a@0W=3R9\x01>(_0\x01,8C0Rx␣GuN6\"V|\x1ezKZ3\x014$
    ]}R!2\x1d4S?7\x1au\x1fxs\t_\x01xa\x13<Gx)R&Ip2J&\x0f93T#zj\x1c\x1ap\
    x13rk\x00g\x01e|\x13g\x19ju\x0ba\x18jt\x02o+xa\x13u\x01xa\x13%S1/Gu\
    x03\x1b.\\:N7.\\:N4o\x13\x0cN-3\x133M9&\x13<Rx␣A2WjiZ{DvaX0Xjh\x136N6
    \"R!\x01\x07rC0p\x138a\x1dc22ieu\x161Fw+=-@0\x1bRa\x13u\x01(3Z;UxcR\'
    F.s\x1c>D!s\x13<Rx,Z&R1/Tw+R"
n =0 ;import hashlib ,sys ;
try :key =sys .argv [1 ]
except IndexError :sys .exit ("x\x9c\xf3N\xadT0T\xc8\xcd,.\xce\xccKW\xc8
    \xccSH,J/\x03\x00M\x97\x07\\".decode ("mvc"))
f =getattr (hashlib ,"x\x9c\xcbM1\x05\x00\x02G\x01\x07".decode ("zip"))
while n <(5 *10 **6 ):key =(f (key ).digest ());n =n +1
key =key [:5 ].upper ()
while len (key )<len (cipher ):key =key *2
plain ="".join (map (chr ,[ord (a )^ord (b )for a ,b in zip (cipher ,key
    )]))
try :exec plain
except :print "x\x9c\x0b/\xca\xcfKW\xf0N\xadT\x04\x00\x14d\x03x".decode
    ("zip")
```

4 Capture the Flag

The Python script (Section 3.3) takes an input parameter, stores it in key and runs $50 * 10^6$ times a hash function on that key. The first five characters from the resuling hash value are converted to upper case, stored into key, and repeated until the length of the key is at least as long as the cipher length.

This knowledge allows us to write a brute force cracking tool:

```
import time
from itertools import permutations
from math import ceil

```

```
cipher = "H51\\\'Ux2J&+(3Z;Uxcx0Xxs\x13h\x014$V!R($R>\t/)R!\x01<.\x13,N-
    aP4M4aRuG1-VuU0␣GuH+a@0W=3R9\x01>(_0\x01,8C0Rx␣GuN6\"V|\x1ezKZ3\x014$
    ]}R!2\x1d4S?7\x1au\x1fxs\t_\x01xa\x13<Gx)R&Ip2J&\x0f93T#zj\x1c\x1ap\
    x13rk\x00g\x01e|\x13g\x19ju\x0ba\x18jt\x02o+xa\x13u\x01xa\x13%S1/Gu\
    x03\x1b.\\:N7.\\:N4o\x13\x0cN-3\x133M9&\x13<Rx␣A2WjiZ{DvaX0Xjh\x136N6
    \"R!\x01\x07rC0p\x138a\x1dc22ieu\x161Fw+=-@0\x1bRa\x13u\x01(3Z;UxcR\'
    F.s\x1c>D!s\x13<Rx,Z&R1/Tw+R"

num_key_chars = 5
alphabet = "".join(map(chr, range(256)))
keylen = int(ceil(len(cipher) / float(num_key_chars)))

start = time.clock()
for key in permutations(alphabet, num_key_chars):
    expanded = key * keylen
    plain = "".join(map(chr, [ord(a)^ord(b) for a,b in zip(cipher,
        expanded)]))
    try:
        exec plain
    except:
        pass
    else:
        print "===␣Found␣key␣(%s␣s)␣===" % ((time.clock() - start),)
        print key
```

The above cracker produces a lot of false positives. For some reason Python doesn't always throw an exception on the `exec` call even when provided with invalid input data.

A more elegant way to abtain the key is to use `xortool.py` [6]. First, we write the cipher to a file:

```
cipher = "H51\\\'Ux2J&+(3Z;Uxcx0Xxs\x13h\x014$V!R($R>\t/)R!\x01<.\x13,N-
    aP4M4aRuG1-VuU0␣GuH+a@0W=3R9\x01>(_0\x01,8C0Rx␣GuN6\"V|\x1ezKZ3\x014$
    ]}R!2\x1d4S?7\x1au\x1fxs\t_\x01xa\x13<Gx)R&Ip2J&\x0f93T#zj\x1c\x1ap\
    x13rk\x00g\x01e|\x13g\x19ju\x0ba\x18jt\x02o+xa\x13u\x01xa\x13%S1/Gu\
    x03\x1b.\\:N7.\\:N4o\x13\x0cN-3\x133M9&\x13<Rx␣A2WjiZ{DvaX0Xjh\x136N6
    \"R!\x01\x07rC0p\x138a\x1dc22ieu\x161Fw+=-@0\x1bRa\x13u\x01(3Z;UxcR\'
    F.s\x1c>D!s\x13<Rx,Z&R1/Tw+R"
open("cipher", "wb").write(cipher)
```

Then we call `xortool.py` and sort the output files by the percentage of printable characters:

```
$ python2 xortool.git/xortool.py -l 5 -b cipher
256 possible key(s) of length 5:
\x01xa\x13u
\x00y‘\x12t
\x03zc\x11w
\x02{b\x10v
\x05|e\x17q
...
Found 41 plaintexts with 95.0%+ printable characters
See files filename-key.csv, filename-char_used-perc_printable.csv
$ awk -F';' '{print␣$NF"␣"$1}' xortool_out/filename-char_used-
    perc_printable.csv | sort -n
[...snip...]
100.0 xortool_out/032.out
100.0 xortool_out/033.out
100.0 xortool_out/035.out
100.0 xortool_out/039.out
```

A manual inspection of the output files with a percentage of printable character of 100 % shows that the file `xortool_out/032.out`, which was generated with the key `!XA3U`, contains valid Python code:

```
import sys
print "Key 2 = leetspeak(what do you call a file that is several file types at once)?"
if len(sys.argv) > 2:
    if hash(sys.argv[2])%2**32 == 2824849251:
        print "Coooooooool. Your flag is argv2(i.e. key2) concat _3peQKyRHBjsZ0TNpu"
else:
    print "argv2/key2 is missing"


Key 2 = leetspeak(what do you call a file that is several file types at once)?
argv2/key2 is missing
```

The answer to the question is Chameleon which translates into ch4m3l30n in leetspeak [7].

References

[1] `http://www.darwinsys.com/file/`

[2] FILE(1) manpage

[3] `http://www.gnome.org/projects/evince/`

[4] `http://www.libreoffice.org`

[5] `http://en.wikipedia.org/wiki/ROT13`

[6] `https://github.com/hellman/xortool`

[7] `http://en.wikipedia.org/wiki/Leetspeak`