# ringzerOteam – Coding – I saw a little elf

**– Kileak**

This one bugged me for a while, though it seemed to be a quickly to solve challenge :)

The challenge page presented us with  simple base64 encoded message and the challenge caption already hinted, this being an encoded elf binary.

At first I just thought, ok let's decode it, execute it and get some easy points. Well, it had some small surprises coming up, but let's start with retrieving the page, decoding the base64 message and save it to a file.

```
import rz
import base64
import re
import subprocess
from bs4 import BeautifulSoup

print("\nringzer0team - Coding Challenge 15 Solver")
print("-----------------------------------------")

# Retrieve the challenge text
code = rz.GetChallenge(15)
soup = BeautifulSoup(code)
soup = soup.find ('div', {'class':'message'})
soup = soup.get_text()
soup = soup.replace ('----- BEGIN Elf Message -----', '').replace ('----- End
Elf Message -----', '').strip()

rz.ShowAction("Decode base64 message")
soup = base64.b64decode(soup)

# Write decoded string to a file
rz.ShowAction("Write decoded message to 'file56'")
with open("file56", 'wb') as f:
      f.write(soup)
      f.close()
[SNIP]
```

Looking at the generated file revealed it was just another base64 code. So I tried to decode it twice. Well, the file was filled with binary garbage at least. But rerunning the code multiple times ended up with the file containing binary and sometimes base64 again. The challenge was encoding it randomly multiple times in base64 . We need a way to find out when we're finished with the base64 decoding.

After some searching for a suiting regular expression (it could have been so much easier having a more indepth look at the file for some static strings (like CBILG for example), but well... ;-))

```
# The challenge string might be encoded multiple times
while re.match(r'^[A-Za-z0-9+/]*={0,3}$', soup):
      soup = base64.b64decode(soup)
```

With this change I always ended up with a binary file, but still no valid ELF. But looking at the content it quickly becomes clear, that the content is just reversed (CBILG <=> GLIBC), so let's reverse it also before writing to the file:

```
# Reverse the string
soup = soup[::-1]
```

Now we ended up with an ELF64 binary, which returned a string value, that needs to be sent back to the challenge page. Let's add this up in the solver

```
# Execute the file and retrieve the output
rz.ShowAction("Execute file");
solution = subprocess.check_output(["./file56"])
rz.ShowAction("File output: %s" % solution)

# Send it back to ringzer0
response = rz.SendSolution(15, solution)
flag = rz.GetFlag(response)

rz.ShowAction("Received flag : %s\n" % flag)
```

Though everything seemed to be correct, the challenge page just didn't want to accept my challenge response:

```
$ python code15.py

ringzer0team - Coding Challenge 15 Solver
-----------------------------------------
 [+] Sending request: http://ringzer0team.com/challenges/15
 [+] Decode base64 message
 [+] Write decoded message to 'file56'
 [+] Execute file
 [+] File output: ljFn84
 [+] Sending solution: http://ringzer0team.com/challenges/15/ljFn84
 [+] Received flag : Fail
```

Back to the challenge page it stated „You have 3 seconds to get the secret word". Hmm, it did take some while after „Execute file" and the output from the file appeared. Let's check what happens there:

```
$ ltrace ./file56
__libc_start_main(0x4005cc, 1, 0x7fff2f409208, 0x400630 <unfinished ...>
sleep(2)                                          = 0
puts("99wKke"99wKke
)                                 = 7
+++ exited (status 0) +++
```

Ok... It just sleeps for 2 seconds making it harder to meet the time limit. Let's teach it some manners.

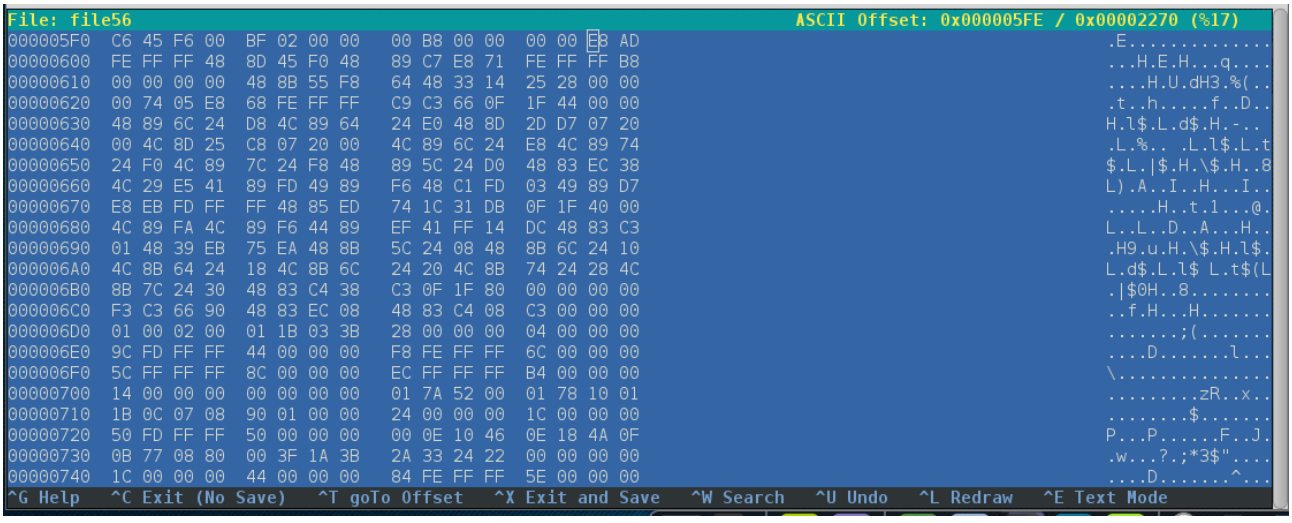
A brief look at the disassembly from the file with objdump shows us the unwanted call to sleep:

```
$ objdump -S file56

file56:     file format elf64-x86-64

[SNIP]
00000000004005cc <main>:
  4005cc:   55                      push   %rbp
  4005cd:   48 89 e5                mov    %rsp,%rbp
  4005d0:   48 83 ec 10             sub    $0x10,%rsp
  4005d4:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  4005db:   00 00
  4005dd:   48 89 45 f8             mov    %rax,-0x8(%rbp)
  4005e1:   31 c0                   xor    %eax,%eax
  4005e3:   c7 45 f0 39 39 77 4b    movl   $0x4b773939,-0x10(%rbp)
  4005ea:   66 c7 45 f4 6b 65       movw   $0x656b,-0xc(%rbp)
  4005f0:   c6 45 f6 00             movb   $0x0,-0xa(%rbp)
  4005f4:   bf 02 00 00 00          mov    $0x2,%edi
  4005f9:   b8 00 00 00 00          mov    $0x0,%eax
  4005fe:   e8 ad fe ff ff          callq  4004b0 <sleep@plt>
  400603:   48 8d 45 f0             lea    -0x10(%rbp),%rax
  400607:   48 89 c7                mov    %rax,%rdi
  40060a:   e8 71 fe ff ff          callq  400480 <puts@plt>
  40060f:   b8 00 00 00 00          mov    $0x0,%eax
  400614:   48 8b 55 f8             mov    -0x8(%rbp),%rdx
  400618:   64 48 33 14 25 28 00    xor    %fs:0x28,%rdx
  40061f:   00 00
  400621:   74 05                   je     400628 <main+0x5c>
  400623:   e8 68 fe ff ff          callq  400490 <__stack_chk_fail@plt>
  400628:   c9                      leaveq
  400629:   c3                      retq
  40062a:   66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
[/SNIP]
```

Searched the position of the corresponding opcodes with hexedit and found them at 0x05FE




Let's wake it up then. We'll just overwrite this with a nop-sled („\x90), so it won't be sleeping anymore at start.

```
# Reopen the file and overwrite the sleep function with NOP's
nopsled = "\x90\x90\x90\x90\x90"
with open("file56", 'rb+') as f:
      f.seek(1534)            # 0x05FE
      f.write(nopsled)
      f.close()
```

Works. After this change our solver retrieves the challenge page, decodes the file, executes it (now it spits out the passcode immediately) and the challenge page accepts our solution

```
$ python code15.py

ringzer0team - Coding Challenge 15 Solver
-----------------------------------------
 [+] Sending request: http://ringzer0team.com/challenges/15
 [+] Decode base64 message
 [+] Write decoded message to 'file56'
 [+] Execute file
 [+] File output: hnMaHO
 [+] Sending solution: http://ringzer0team.com/challenges/15/hnMaHO
 [+] Received flag : FLAG-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXx
```

### *code15.py*

```
import rz
import base64
import re
import subprocess
from bs4 import BeautifulSoup

print("\nringzer0team - Coding Challenge 15 Solver")
print("-----------------------------------------")

# Retrieve the challenge text
code = rz.GetChallenge(15)
soup = BeautifulSoup(code)
soup = soup.find ('div', {'class':'message'})
soup = soup.get_text()
soup = soup.replace ('----- BEGIN Elf Message -----', '').replace ('----- End
Elf Message -----', '').strip()

rz.ShowAction("Decode base64 message")

# The challenge string might me encoded multiple times
while re.match(r'^[A-Za-z0-9+/]*={0,3}$', soup):
      soup = base64.b64decode(soup)

# Reverse the string
soup = soup[::-1]

# Write decoded string to a file
rz.ShowAction("Write decoded message to 'file56'")
with open("file56", 'wb') as f:
      f.write(soup)
      f.close()

# Reopen the file and overwrite the sleep function with NOP's
nopsled = "\x90\x90\x90\x90\x90"
with open("file56", 'rb+') as f:
      f.seek(1534)
      f.write(nopsled)
      f.close()

# Execute the file and retrieve the output
rz.ShowAction("Execute file");
solution = subprocess.check_output(["./file56"])
rz.ShowAction("File output: %s" % solution)

# Send it back to ringzer0
response = rz.SendSolution(15, solution)
flag = rz.GetFlag(response)

rz.ShowAction("Received flag : %s\n" % flag)
```

### *rz.py (my base library for the ringzer0 challenges)*

```
import urllib2
from bs4 import BeautifulSoup

# Common ringzer0 variables
RZ_BASE_URL = 'http://ringzer0team.com'
RZ_AUTH_COOKIE = 'PHPSESSID=XXXXXXXXXXXXXXXXXXXXXXXX'

def ShowAction(msg):
      print(" [+] %s" % msg)

def GetChallenge(challID):
      request = urllib2.build_opener()
      request.addheaders.append(('Cookie', RZ_AUTH_COOKIE))
      requestURL = '%s/challenges/%s' % (RZ_BASE_URL, challID)
      ShowAction('Sending request: %s' % requestURL)
      response = request.open(requestURL)
      return response.read()

def SendSolution(challID, sol):
      request = urllib2.build_opener()
      request.addheaders.append(('Cookie', RZ_AUTH_COOKIE))
      requestURL = '%s/challenges/%s/%s' % (RZ_BASE_URL, challID, sol)
      ShowAction('Sending solution: %s' % requestURL)
      response = request.open(requestURL)
      return response.read()

def GetFlag(response):
      soup = BeautifulSoup(response)
      flag = soup.find ('div', { 'class':'alert alert-info'})

      if flag != None:
            flag = flag.get_text()
            return flag

      return "Fail"
```