Write Up: Hack.lu 2013 - Internals 250 - What's wrong with this?

M. Konrad, M. Lambertz

Abstract

This write up gives a short summary for the Hack.lu 2013 CTF challenge "Internals: What's wrong with this?" which was solved by 58 of 413 groups and gave 250 of 5372 points.

1 Description

"We managed to get this package of the robots servers. We managed to determine that it is some kind of compiled bytecode. But something is wrong with it. Our usual analysis failed - so we have to hand this over to you pros. We only know this: The program takes one parameter and it responds with "Yup" if you have found the secret code, with "Nope" else. We expect it should be obvious how to execute it."
The file `hello.tar.gz` was given.

2 First Analysis

Besides a few shared libraries the archive contains a Python interpreter (`py`), an excutable Zip file (`library.zip`) and the executable `hello` which is the one we have to crack. Since we already knew that the the program outputs "Nope" and "Yup", we grepped for these strings to get an idea where the output takes place. However, the command

```
$ grep -e 'Yup\|Nope' *
```

returned no results. Only after extracting `library.zip` and running the `grep` command again, we obtained the result:

```
Binary file __main__hello__.pyc matches
```

This indicates that the bytecode in `__main__hello__.pyc` seems to be a good starting point for further analysis.

3 Decompiling the Python bytecode: First attempt

We used Decompyle++ [1] to decompile `__main__hello__.pyc` and obtained the following result:

```
# Source Generated with Decompyle++
# File: __main__hello__.pyc (Python 2.7)

import sys
import dis
import multiprocessing
import UserList

def encrypt_string(s):
Unsupported opcode: <255>
    pass
```

```
12 # WARNING: Decompyle incomplete
13
14 def rot_chr(c, amount):
15     None = chr(((ord(c) + 33) % amount) / 94 % 33)
16
17 SECRET = 'w*0;CNU[\\gwPWk}3:PWk"#&:ABu/:Hi,M'
18 if encrypt_string(sys.argv - 1) == SECRET:
19     print
20     print >>'Yup'
21 else:
22     print
23     print >>'Nope'
24 None = None
```

This looks definitively like what we are looking for. We have a function called `encrypt_string`, a string variable called `SECRET`, a check whether some first command line argument when supplied to the `encrypt_string` function matches `SECRET`, and output.

Unfortunately, the decompilation is somewhat broken. First, the `encrypt_string` is not decompiled because of an invalid opcode (a list of valid opcodes and their meanings can be found in [2]). Second, the generated Python code contains several syntax errors, e.g. in line 16 (you can't assign to `None`) or in line 19 (you can't subtract 1 from a list).

If we disassemble `__main__hello__.pyc`, again using Decompyle++, we observe even more oddities. For instance, every function ends with the instruction `IMPORT_STAR` instead of `RETURN_VALUE` which would be the proper last instruction of a function. This strongly indicates that the opcodes of the Python interpreter shipped within `hello.tar.gz` have been tampered with. A comparison of the opcode map of the interpreter with the map of our system's Python interpreter confirms this assumption: the mapping is different.

4 Decompiling the Python bytecode: Second attempt

Decompyle++ stores the opcode mappings in `.map` files which are transformed into `.cpp` files during the compilation. To teach Decompyle++ how to correctly decompile `__main__hello__.pyc`, we simply replaced the original `python_27.map` with a modified one. We wrote a simple Python script for this task:

```
import opcode
import sys

opmapfile = sys.argv[1]

with open(opmapfile, "r") as fd:
    opmap = {}
    for line in fd:
        code, op = line.split()
        op = op.replace("+", "_")
        opmap[op] = int(code)

for k,v in opcode.opmap.items():
    op = k.replace("+", "_")
    if opmap.has_key("_".join((op, "A"))):
        op = "_".join((op, "A"))
    opmap[op] = v

with open(opmapfile, "wt") as fd:
    for k,v in sorted(opmap.items(), key = lambda x: x[1]):
        fd.write("%-3s␣%s\n" % (v,k))
```

If we call this script with the modified interpreter and `python_27.map` as the first argument, we can successfully decompile `__main__hello__.pyc` to

```python
# Source Generated with Decompyle++
# File: __main__hello__.pyc (Python 2.7)

import sys
from hashlib import sha256
import dis
import multiprocessing
import UserList

def encrypt_string(s):
    new_str = []
    for (index, c) in enumerate(s):
        if index == 0:
            new_str.append(rot_chr(c, 10))
            continue
        new_str.append(rot_chr(c, ord(new_str[index - 1])))

    return ''.join(new_str)


def rot_chr(c, amount):
    return chr(((ord(c) - 33) + amount) % 94 + 33)

SECRET = 'w*0;CNU[\\gwPWk}3:PWk"#&:ABu/:Hi,M'
if encrypt_string(sys.argv[1]) == SECRET:
    print 'Yup'
else:
    print 'Nope'
```

5 Capture the Flag

In order to obtain the secret, we have to write a `decrypt_string` function which implements the inverse of `encrypt_string`. This is pretty straightforward actually:

```python
SECRET = 'w*0;CNU[\\gwPWk}3:PWk"#&:ABu/:Hi,M'

def decrypt_string(s):
    new_str = []
    for (index, c) in enumerate(s):
        if index == 0:
            new_str.append(rot_chr(c, 10))
            continue
        new_str.append(rot_chr(c, ord(s[index - 1])))
    return ''.join(new_str)

def rot_chr(c, amount):
    return chr(((ord(c) - 33) - amount) % 94 + 33)

print decrypt_string(SECRET)
```

This gives us the flag: `modified_in7erpreters_are_3vil!!!`

References

[1] https://github.com/zrax/pycdc

[2] http://docs.python.org/2/library/dis.html