Technical Problem Share [09/04/2020]

Edit by Liu Yuancheng

1. Problem

Bug caused by using changeable value as the default value in "python method overload"

2. Problem Back Ground

In python we can set the passed in parameter's default value to overload a method/function in below format:

```
def testFunction(self, param1, param2=None, param3=None):

testFunction(1)
testFunction(1, param2=2)
testFunction(1, param2=2, param3='string')
```

Normally we use "None" as the parameter's default value. We can also use the un-changeable value (such as str/bool) as the default value, but is it OK we use empty list [] (a extendable value) as its default value? Will this cause any problem?

3. Problem Test Case Program

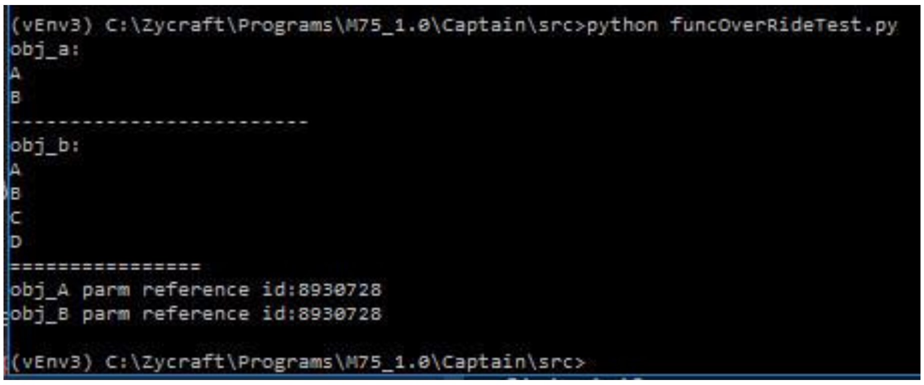
To test whether we can use a changeable value as the method overload default value, we created below test program and we overload the class constructor __init__() function:

```
""" A test program using empty list as the passed-in parameters'
    default value.
"""
class TestFunc(object):

    def __init__(self, id, parm = []):
        self.ObjId = id
        self.parm = parm

    def addNewParm(self, newParm):
        self.parm.append(newParm)

    def printAllParm(self):for item in self.parm:
            print(item)

obj_A = TestFunc('obj_a')
obj_A.addNewParm('A')
obj_A.addNewParm('B')
print(str(obj_A.ObjId)+':')
```

```
obj_A.printAllParm()

print('-------------------------')
obj_B = TestFunc('obj_b')
obj_B.addNewParm('C')
obj_B.addNewParm('D')
print(str(obj_B.ObjId)+':')
obj_B.printAllParm()
print ('================')
print ("obj_A parm reference id:" +str(id(obj_A.parm)))
print ("obj_B parm reference id:" +str(id(obj_B.parm)))
```

When viewing the code, logically as we call the function <addNewParm()> to add character ‘A’ and ‘B’ to object A, the object A’s parameter list should be [A, B] and the object B’s parameter is [C, D].

But if we run the program, the result is different:



```
(vEnv3) C:\Zycraft\Programs\M75_1.0\Captain\src>python funcOverRideTest.py
obj_a:
A
B
-------------------------
obj_b:
A
B
C
D
================
obj_A parm reference id:8930728
obj_B parm reference id:8930728

(vEnv3) C:\Zycraft\Programs\M75_1.0\Captain\src>
```

⇨ The value we added in object A also appeared in the object B. And the object A's parameter and object B's parameter shared same memory id. When this happened, it is difficult for us to find the program's problem during debugging.

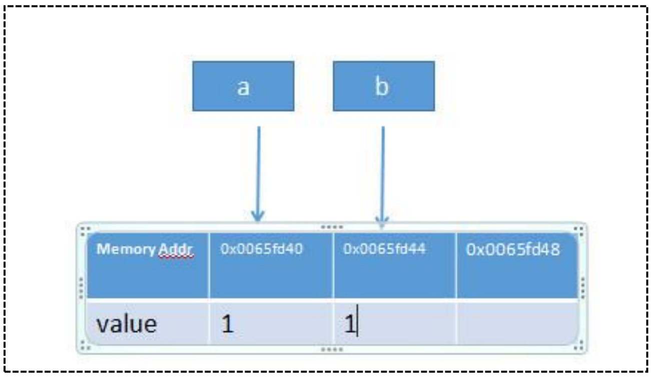
4. Problem Analysis

The reason caused this problem is python's value assignment feature in memory management. In python the parameter will put the value in the memory first and then point the parameter to the memory address during the __init__() process.

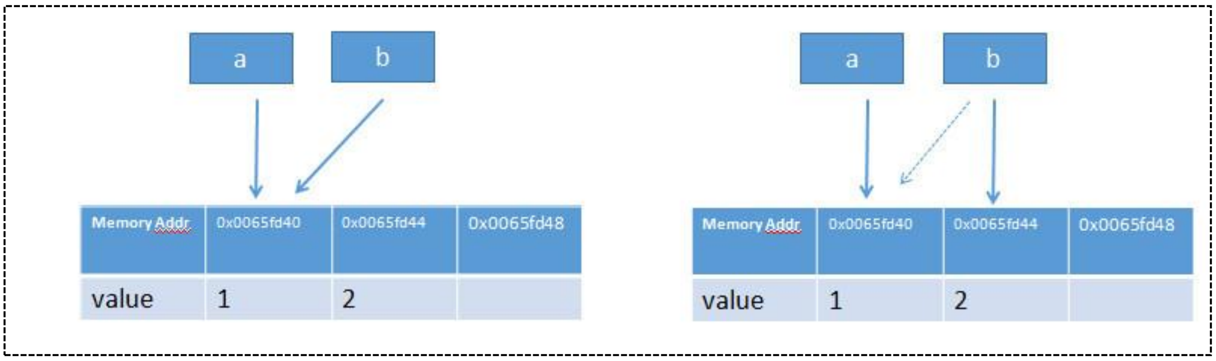
For example, in C if we do this value assignment:

```
a = 1; b = a; b =2;
```

Two memory locations will be arranged for the two parameters and then the value will be put in to the memory like this:



But if we do the same thing in python, python will fill a memory position with value 1, then point <a> and <b> to that position. If we change <b>'s value to 2. Value 2 will be filled in another memory position, then point b to that new memory address.



So if we use a changeable value such as empty list [] as the default value in the __init__() function, the empty list [] will be filled in the memory first, then the two objects' parameter will point to the same memory address which hold the [] at the init process. So in <section 2> program when we init the object B and passed in [C, D], B's parameter list is not [], it has already be filled by [A, B] during object A's __init__() process, and that is the reason why we got the unexpected running result.

5. Problem Situation Usage in cyber security

Assume some company provides a python function API but they have already used pyinstaller to complied and packaged the code so other people cannot changed the code or they only provide the *.pyc file. If in their init() function, they use changeable value such as dictionary or list to do the function overload, we can do duplicate init the object before the real initiation part and passed in some value to do the attack. Then inserted value will change the function behavior during the execution.