Primera semana de trabajo [01 - 04 de abril]

Milton Inostroza Aguilera

April 18, 2008

resumen

Al terminar la semana creo tener bastante avanzado el capturador de huella de ejecución. Este trata de capturar:

- llamada a funciones.
- llamada a métodos.
- definición de atributos.
- definición de variables locales.
- cambio de valor de atributos.
- cambio de valor de variables locales.

Contents

1 Desarrollo

Aún no se muy bien cual es la forma mas óptima de hacer lo anterior, pero al terminar la semana ya se tiene una aproximación. A continuación explico el código resultante:

```
import sys
from inspect import isfunction, ismethod, getmembers
import dis
```

Se importan las librerías que se utilizarán.

```
no_register_attributes = ( 'self','__module__',)
no_register_method = (
'__setattr__',
'__getattribute__',
'__setitem__',
'ismethod',
'isfunction',
'genera_id',
'locals_var_key',
'update',
)
locals_var = {}
```

Estructuras que sirven básicamente para no registrar en locals_var los métodos o atributos utilizados por nuestro script para capturar la huella de ejecución de los otros scripts. locals_var es la estructura más importante ya que en ella se almacenarán los objetos capturados de la huella de ejecución del programa objetivo.

```
def genera_id():
    id = 0
    for k,v in locals_var.items():
        id = id + len(v)
        return id
```

Función que genera un id a partir del tamaño que tenga las estructuras internas de locals_var, esto se hace ya que cada objeto que resida en locals_var debe ser identificado con id único y mayor igual a uno.

```
def locals_var_key(value):
    for i in locals_var:
        if i.co_name == value:
            return i
```

Función que retorna el *code* correspondiente al value que se manda. El *code* es la llave del diccionario *locals_var*. Generalmente esta función se utiliza para cuando se registra un atributo de una clase en *locals_var*.

```
class Descriptor(object):
    def __setattr__(self, name, value):
        id = genera_id()
        key = type(self).__name__
        key = locals_var_key(key)
        locals_var[key].update({name:id},id,key)
        object.__setattr__(self, name, value)
```

Esta clase es un *descriptor* y permite que cada vez que definamos un atributo en nuestra clase el método *__setattr__* funcione como un trigger y nos permita registra esta acción en nuestro *locals_var*.

```
class Diccionario(dict):
    def __setitem__(self,k,v):
        if k in no_register_attributes or isfunction(v) or ismethod(v):
            return
        #frame se puede transportar desde la llamada de register_locals
        dict.__setitem__(self,k,v)

    def update(self,d,id,code):
        for k,v in d.items():
            if not k in no_register_attributes or isfunction(v) or ismethod(v):
                if not self.has_key(k):
                    self[k] = id + 1
                        id = id + 1
```

Esta clase permite ir registrando todos los objetos en *register_locals*, en realidad es la estructura interna de este diccionario. El método *update* verifica que el objeto no pertenezca a ciertas categorías y luego de satisfacer esa condición los almacena y incrementa el identificador en una unidad.

```
class Trace(object):
    def __init__(self,locals={}):
        self.locals = Diccionario(locals)

    def trace(self,frame,event,arg):
        lineno = frame.f_lineno
        code = frame.f_code
        locals = frame.f_locals
        if event == \call":
            if code.co_name in no_register_method:
                return
            #dis.disassemble(code,frame.f_lasti)
            self.register_locals(code,locals)
            return Trace(locals).trace
        elif event == \line":
            self. register_locals(code,locals)
            return self.trace
        elif event == \return":
            self.register_locals(code,locals)
```

```
    def register_locals(self,code,locals):
        id = genera_id()
        if locals_var.has_key(code):
            locals_var[code].update(locals,id,code)
        else:
            locals_var[code] = Diccionario()
            locals_var[code].update(locals,id,code)
```

Esta clase permite registrar todos los movimientos del programa a través del método *trace*.

```
trace = Trace()
sys.settrace(trace.trace)
```

Se instancia la clase *Trace* y luego asignamos el método *trace* a *sys.settrace*.

Siempre es bueno ver los resultado de lo que hemos construido, para esto capturaremos la huella de ejecución del siguiente programa:

```
from pyTOD import *
def prueba():
    x = 10
    a = x
    x = 15

class ClasePrueba(Descriptor):
    z =1
    def __init__(self):
        x=1
        self.w = 20

    def impresion(self):
        print 'hola'

    def asignacion(self):
        j=4

prueba()
p = ClasePrueba()
p.impresion()
p.asignacion()

for i,k in locals_var.items():
    print i,k
```

El resultado de la captura de huella es la siguiente:

```
<code object asignacion at 0xb7d3c848, file \examples.py", line 16>
    {'j': 9}
<code object __init__ at 0xb7d3c4e8, file \examples.py", line 11>
    {'x': 7}
<code object prueba at 0xb7d3c608, file \examples.py", line 3>
    {'a': 6, `x': 5}
<code object impresion at 0xb7d3c6e0, file \examples.py", line 14>
    {}
<code object ClasePrueba at 0xb7d3cc80, file \examples.py", line 9>}
    {'asignacion': 4, `w': 8, `z': 1, `impresion': 3, `__init__': 2}
```

Explicaré la siguiente línea:

```
<code object asignacion at 0xb7d3c848, file \examples.py", line 16>
    {'j': 9}
```

Significa que la llave de este diccionario es un tipo de dato *code* que contiene al objeto asignación escrito en la linea 16 del archivo examples.py como parte de la clase *clasePrueba* y dentro de este objeto se define la variable local j y su número único de identificación es 9.

2 Conclusión

Al término de esta primera semana de trabajo se reafirman conocimientos primarios sobre la depuración omnisciente y el conocimiento que se debe tener acerca del lenguaje de programación en el cual se implementará el depurador.

La función *settrace* es de gran ayuda, ya que en estos momentos permite tener control linea a linea de lo que hace el programa objetivo, permitiendo hasta el momento no instrumentar el código del programa objetivo.

References

[1] Library Reference: http://docs.python.org/lib/debugger-hooks.html

[2] Module of Python: http://lfw.org/python/inspect.py