PAWAN M RANGANATHA RAO
9/17/13

Secure Turing Complete Sandbox – Building a Turing Complete Sandbox

The main aim of this assignment was to create a turing complete sandbox. The method followed here was to use a subset of existing language features to ensure that the sandbox is secure and turing complete.

The sandbox was created using Python 2.7.3 on an Ubuntu 12.04 32-bit VM. It executes python scripts using a limited set of allowed functions. Given below are the list of disallowed built-in functions and the allowed list of built-in types:

```
Disallowed Functions:
'__import__','apply','bytearray','compile','dir','delattr','callable',
'eval','exec','execfile','file','getattr','globals','hasattr','input',
'intern','id',
'locals','memoryview','open','reload','setattr','type','vars',’zip’
’raw_input’

Allowed Types: 'True','False','int','float','long','complex','list',
‘buffer','unicode'
```

The list of built-in functions to be allowed was obtained using the `inspect` module in Python. From this, the above set of blacklisted functions was subtracted to obtain a list of allowed functions. The list allowable types in the sandbox was specified manually.

Users can also define their own functions but must follow the rules and constraints specified in the sandbox. No imports are allowed inside the sandbox. File and Network operations are not allowed. User input using raw_input() is disallowed. To ensure no malicious attacks on the sandbox, the resources used by the sandbox namely the memory, CPU etc. are restricted. Passing command line arguments to the script to be run is allowed.

The sandbox was tested against several test cases to ensure it functions as expected. These test cases include:

- Counting from 10 to 1(countdown.py)
- Calculating the 1st 10 Fibonacci numbers (Fibonacci.py)
- Binary search on a user input array (bin_search.py)
- Invalid test script containing blacklisted functions (invalid_test.py)

The user code is run inside the sandbox using the execfile() call with a dictionary of safe functions passed to it to ensure safe eval(). Before performing the execfile() call, the sandbox first creates an Abstract Syntax Tree(AST) from the script to be run. The AST is then checked for nodes that are unsafe and if any unsafe nodes are found, an exception is raised.

For example, consider the test case countdown.py:

The AST constructed for this script is as below:

```
Module(None, Stmt([For(AssName('cnt', 'OP_ASSIGN'),
CallFunc(Name('range'), [Const(10), Const(0), UnarySub(Const(1))],
None, None), Stmt([Printnl([Name('cnt')], None)]), None)]))
```

The AST is then traversed and checked for nodes that are unsafe and only then the user code is executed using the call to execfile().

Since the sandbox primarily is unable to import modules, it was not possible to implement a sandbox within a sandbox. However the turing completeness of the sandbox can be successfully proved by saying that if it successfully executes code from a turing complete language.

Therefore, for proving this, the Brainfuck language which is turing complete was considered. The sandbox has the ability to act as a Brainfuck interpreter thus proving its turing completeness.

The Brainfuck programming language consists of eight commands, each of which is represented as a single character.

>	Increment the pointer.
<	Decrement the pointer.
+	Increment the byte at the pointer.
-	Decrement the byte at the pointer.
.	Output the byte at the pointer.
,	Input a byte and store it in the byte at the pointer.
[	Jump forward past the matching] if the byte at the pointer is zero.
]	Jump backward to the matching [unless the byte at the pointer is zero.

To prove that the sandbox can interpret brainfuck scripts, a python Brainfuck interpreter is passed to the sandbox which prints out all the Fibonacci numbers below 100.

The interpreter code as well as the Brainfuck code are committed as part of the test cases.

References:

- Brainfuck Programming Language
  http://www.muppetlabs.com/~breadbox/bf/
- Brainfuck Fibonacci Code
  http://esoteric.sange.fi/brainfuck/bf-source/prog/fibonacci.txt
- Brainfuck Python Interpreter
  https://github.com/DoctorLai/PyUtils/blob/master/bf.py