Marc Budofsky
September 18, 2012

Turing-Complete Sandbox Documentation

The Sandbox [Sandbox.py] was created using Python 2.7.3. It executes scripts written in Python using a limited subset of built-in Python functions and types, as listed below:

```
Allowed Built-In Functions (35): 'abs', 'all', 'any', 'bin',
'callable', 'chr', 'cmp', 'coerce', 'divmod', 'filter', 'format',
'hash', 'hex', 'intern', 'isinstance', 'issubclass', 'iter',
'len', 'map', 'max', 'min', 'next', 'oct', 'ord', 'pow', 'print',
'range', 'raw_input', 'reduce', 'repr', 'round', 'sorted', 'sum',
'unichr', 'zip'
Allowed Built-In Types (12): 'True', 'False', 'int', 'float',
'long', 'complex', 'str', 'unicode', 'list', 'tuple', 'buffer',
'xrange'
```

Users can also define their own functions, but must adhere to the limits in place on what built-in functions can be utilized in doing so. If the user wishes to create a script that will make use of 'sys.argv', they should modify the code to look for the variable 'argv' instead and index their arguments from there. No imports are allowed in the script(s) to be executed, and the memory of the Sandbox is limited to 512 MB. File and Network are I/O are also not allowed.

Several test cases were created examine the functionality of the Sandbox. These scripts include: Counting from 10 to 1 [TestCases/TestCase01.py], Calculating the first 10 Fibonacci Numbers [TestCases/TestCase02.py], Computing the Factorial of a User Provided Number [TestCases/TestCase03.py], Adding 2 Numbers using the Binary XOR Function [TestCases/TestCase04.py], Size and Area of a Rectangle using Classes [TestCases/classTest.py] and finally a malicious loop [TestCases/maliciousTest.py] to determine if the code can properly filter lambda functions.

When running a script, the application first creates an Abstract Syntax Tree (AST) out of the file. This AST is the traversed, looking for nodes that are not deemed safe; if one of these nodes is encountered, the application exits with an Exception. If the code does not contain any blacklisted nodes, the execfile() function is used to execute the script. A whitelist of functions is passed to the execfile() call to ensure that only safe functions are being used.

Consider TestCase02.py from above (First 10 Fibonacci Numbers):
The application file is passed into the script and converted into an AST representation, as seen below:

```
Module(None, Stmt([Function(None, 'fib', ['x'], [], 0, None,
Stmt([If([(Compare(Name('x'), [('==', Const(0))]),
Stmt([Return(Const(0))])), (Compare(Name('x'), [('==',
Const(1))]), Stmt([Return(Const(1))]))],
Stmt([Return(Add((CallFunc(Name('fib'), [Sub((Name('x'),
Const(1)))], None, None), CallFunc(Name('fib'), [Sub((Name('x'),
Const(2)))], None, None))))]))])), For(AssName('foo',
'OP_ASSIGN'), CallFunc(Name('range'), [Const(10)], None, None),
Stmt([Printnl([CallFunc(Name('fib'), [Name('foo')], None, None)],
None)]), None)]))
```

This tree is then traversed, and when deemed safe, the code is executed: `execfile(scriptToRun,user_allowed_dict)` – 'user_allowed_dict' is a copy of the allowed functions and types as outlined above, as well as any command line arguments that have been passed.

Due to the nature of how the Sandbox checks for safe code prior to execution, it was not possible to implement a sandbox within the Sandbox. This is due primarily to the inability to import modules in the sandboxed script. However, the Sandbox still proves to be Turing-Complete as it has the ability to act as a Brainfuck interpreter, which is a Turing-Complete language. Brainfuck has 8 functions that encompass the entire language:

> > : increment the data pointer (to point to the next cell to the right).
> < : decrement the data pointer (to point to the next cell to the left).
> + : increment (increase by one) the byte at the data pointer.
> - : decrement (decrease by one) the byte at the data pointer.
> . : output the byte at the data pointer as an ASCII encoded character.
> , : accept one byte of input, storing its value in the byte at the data pointer.
> [: if the byte at the data pointer is zero, then instead of moving the instruction pointer forward to the next command, jump it forward to the command after the matching] command*.
>] : if the byte at the data pointer is nonzero, then instead of moving the instruction pointer forward to the next command, jump it back to the command after the matching [command*.

[http://en.wikipedia.org/wiki/Brainfuck]

By passing the Sandbox a Brainfuck Interpreter as the script to be run, and then providing the command to print 'Hello World!' via the interpreter, the Sandbox is therefore Turing-Complete. The 'Hello World!' command for Brainfuck is:

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++.>+.+++++++..
+++.>++.<<+++++++++++++++.>.+++.------.--------.>+.>.
```

[http://en.wikipedia.org/wiki/Brainfuck]