Breaking and Fixing Python applications

Introduction to Python Secure Coding
Deep dive into Python’s core libraries.

We will talk about some of the most critical issues that have been identified during a two year security code review.

Each issue will be analyzed and when possible we will provide a solution or a mitigation strategy.
Our approach

Normal approach

Python → Application → Tester

Our approach

OS → Cross Check → Application → Cross Check → OS

Python → Cross Check → Tester → Cross Check → Python
We will look at what can happen when:

- Input data is of ‘unknown’ type and size
- Libraries are build using only RFC specs
- Data is processed without proper validation
- Logics are changed to be ‘OS independent’
We will talk about..

1. Date and time —> time, datetime, os
2. Numbers —> ctypes, xrange, len, decimal
3. Strings —> input, eval, codecs, os, ctypes
4. Files —> sys, os, io, pickle, cpickel
5. Protocols —> socket, poplib, urllib, urllib2
import time
initial_struct_time = [tm for tm in time.localtime()]

# Example on how time object will cause an overflow
# Same for: Year, Month, Day, minutes, seconds
invalid_time = (2**63)

# change 'Hours' to a value bigger than 32bit/64bit limit
initial_struct_time[3] = invalid_time

overflow_time = time.asctime(initial_struct_time)

**Python 2.6.x**
OverflowError: long int too large to convert to int

**Python 2.7.x**
OverflowError: Python int too large to convert to C long
OverflowError: signed integer is greater than maximum
“time.gmtime” has a check against platform time_t

import time
print time.gmtime(-2**64)
print time.gmtime(2**63)

ValueError: timestamp out of range for platform time_t

But if value is between (-2^63) and (-2^56) or is between (2^55) to (2^62) then another type error is generated

import time
print time.gmtime(-2**63)
print time.gmtime(2**62)

ValueError: (84, 'Value too large to be stored in data type')
import os
TESTFILE = 'temp.bin'

validtime = 2**55
os.utime(TESTFILE,(-2147483648, validtime))
stinfo = os.stat(TESTFILE)
print(stinfo)

invalidtime = 2**63
os.utime(TESTFILE,(-2147483648, invalidtime))
stinfo = os.stat(TESTFILE)
print(stinfo)

Python 2.6.x,
**OverflowError: long int too large to convert to int**

Python 2.7.x, Python 3.1
**OverflowError: Python int too large to convert to C long**
If we use a time value of ‘2^55’ we can have:

$ ls -la temp.bin
-rw-r--r-- 1 user01 user01 5 13 Jun 1141709097 temp.bin
$ stat temp.bin
A: "Oct 10 16:31:45 2015"
M: "Jun 13 01:26:08 1141709097"
C: "Oct 10 16:31:42 2015"

But in some systems if we use ‘2^56’ we can also have:

$ ls -la temp.bin
Segmentation fault: 11
$ stat temp.bin
A: "Oct 10 16:32:50 2015"
M: "Dec 31 19:00:00 1969"
C: "Oct 10 16:32:50 2015"

!! WARNING !!
RISK OF SYSTEM CRASH
RISK OF DATA LOSS

Do NOT play with “os” module.
Modules do **not** have checks or tests for invalid values.

For example, the maximum value for a 64bit system would be $[2^{63}-1]$, but different errors will be generated depending on the used values.

Any number outside the valid range will generate an Overflow.

**SOLUTION**

Implement proper data validation.
We will talk about..

1. Date and time —> time, datetime, os

2. **Numbers** —> ctypes, xrange, len, decimal

3. Strings —> input, eval, codecs, os, ctypes

4. Files —> sys, os, io, pickle, cPickle

5. Protocols —> socket, poplib, urllib, urllib2
import ctypes

#32-bit test with max 32bit integer 2147483647
ctypes.c_char * int(2147483647)

#32-bit test with max 32bit integer 2147483647 + 1
ctypes.c_char * int(2147483648)

#64-bit test with max 64bit integer 9223372036854775807
ctypes.c_char * int(9223372036854775807)

#64-bit test with max 64bit integer 9223372036854775807 + 1
ctypes.c_char * int(9223372036854775808)

Example of overflow message in a 64bit system:

```python
>>> ctypes.c_char * int(9223372036854775808)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
OverflowError: cannot fit 'long' into an index-sized integer
```
<table>
<thead>
<tr>
<th>ctypes calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_byte</td>
</tr>
<tr>
<td>c_longdouble</td>
</tr>
<tr>
<td>c_longdouble</td>
</tr>
<tr>
<td>c_void_p</td>
</tr>
</tbody>
</table>

- ctypes are not limited to size of memory
- overflow checks are mostly missing.

An overflow will occur in both 32bit and 64bit systems.

**SOLUTION**

Implement Overflow checking and data validation.
Even if this behaviour is "by design" and expected, this condition is not checked resulting in a numeric overflow.

```python
valid = (2 ** 63) - 1
invalid = 2 ** 63
for n in xrange(invalid):
    print n
```

**OverflowError: Python int too large to convert to C long**

Even if this behaviour is "by design" and expected, this condition is not checked resulting in a numeric overflow.

This happens because `xrange` uses "Plain Integer Objects" and cannot accept objects of arbitrary length.

**SOLUTION**

Use python "long integer object“ that will allow numbers of arbitrary length as the limit will be the system's memory.
valid = (2**63)-1
invalid = 2**63

class A(object):
    def __len__(self):
        return invalid

print len(A())

OverflowError: long int too large to convert to int

len() does not check for the length of the object and does not use "python int objects" (unlimited). This can cause an Overflow error as the object may contain a ".length" property.

SOLUTION

Use python “python int objects” that will allow numbers of arbitrary length as the limit will be the system's memory.
from decimal import Decimal

try:
    # DECIMAL '1172837167.27'
    x = Decimal("1172837136.0800")
    # FLOAT '1172837167.27'
    y = 1172837136.0800
    if y > x:
        print("ERROR: FLOAT seems comparable with DECIMAL")
    else:
        print("ERROR: FLOAT seems comparable with DECIMAL")
except Exception as e:
    print("OK: FLOAT is NOT comparable with DECIMAL")

Python 2.6.5, 2.7.4, 2.7.10
ERROR: FLOAT seems comparable with DECIMAL   (WRONG)

Python 3.1.2
OK: FLOAT is NOT comparable with DECIMAL   (CORRECT)
try:
    # STRING 1234567890
    x = "1234567890"
    # FLOAT '1172837167.27'
    y = 1172837136.0800
    if y > x:
        print("ERROR: FLOAT seems comparable with STRING")
    else:
        print("ERROR: FLOAT seems comparable with STRING")
except Exception as e:
    print("OK: FLOAT is NOT comparable with STRING")

Python 2.6.5, 2.7.4, 2.7.10
ERROR: FLOAT seems comparable with STRING  (WRONG)

Python 3.1.2
OK: FLOAT is NOT comparable with STRING  (CORRECT)
Python internal compare function does not verify if a comparison of two objects is done by using object of the same type.

In this case python does not know how to compare STRING and FLOAT and returns a FALSE instead of returning an Error.

Same problem if we try to compare DECIMAL and FLOATS, python does not know how to compare this objects and returns a FALSE instead of returning an Error.

**SOLUTION**
Implement strong type checking and perform data validation.
We will talk about..

1. Date and time —> time, datetime, os

2. Numbers —> ctypes, xrange, len, decimal

3. Strings —> input, eval, codecs, os, ctypes

4. Files —> sys, os, io, pickle, cPickle

5. Protocols —> socket, poplib, urllib, urllib2
import os
try:
    # Linux/Unix
    eval("__import__('os').system('clear')", {})
    # Windows
    #eval("__import__('os').system(cls')", {})
    print "Module OS loaded by eval"
except Exception as e:
    print repr(e)

Any code will be executed without limits in the context of the user that loaded the interpreter.
Secret = "42"

value = input("Answer to everything is ? ")

print "The answer to everything is %s" % (value,)

Answer to everything is ? dir()
The answer to everything is
["Secret", '__builtins__', '__doc__', '__file__', '__name__', '__package__']

The dir() function returns “most” of the attributes of an object, and as a result we obtain the “Secret” object.

Answer to everything is ? Secret
The answer to everything is 42
import codecs
import io

b = b'\x41\xF5\x42\x43\xF4'
print("Correct-String %r") % ((repr(b.decode('utf8', 'replace'))))

with open('temp.bin', 'wb') as fout:
    fout.write(b)
with codecs.open('temp.bin', encoding='utf8', errors='replace') as fin:
    print("CODECS-String %r") % (repr(fin.read()))
with io.open('temp.bin', 'rt', encoding='utf8', errors='replace') as fin:
    print("IO-String %r") % (repr(fin.read()))

The original string will be silently truncated at the first byte.

Correct-String —> “u'A\ufffdBC\ufffd''
CODECS-String —> “u'A\ufffdBC'' (WRONG)
IO-String —> “u'A\ufffdBC\ufffd'' (OK)
When “codecs” reads the string it expects a sequence of 4 bytes and does not decode it because it waits for other 3 bytes, and as a consequence the resulting string is truncated.

A better and safer approach would be to read the entire stream and only then proceed to the decoding phase as done by the “io” module.

**SOLUTION**

Either use the “io” module or implement string recognition and validation to detect malformed characters.
import os
os.environ['a=b'] = 'c'

try:
    os.environ.clear()
    print("PASS => os.environ.clear removed variable 'a=b'")
except:
    print("FAIL => os.environ.clear removed variable 'a=b'")
raise

Names and syntax of environment variables names are also based on the specific rules used in each platform.

Python does not share the same logic and tried to implement a generic interface compatible with most operating systems.

This choice of preferring compatibility over security have allowed the existence of cracks in the logic used for environment variables.
It is **possible** to define an environment variable with an empty key, or a variable that contains "="; but **not to remove it**.

```bash
$ env -i =value python -c 'import pprint, os; pprint.pprint(os.environ); del os.environ[""]'

environ({'': 'value'})
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "Lib/os.py", line 662, in __delitem__
    self.unsetenv(encodedkey)
OSError: [Errno 22] Invalid argument
```

```bash
$ env -i python -c 'import pprint, posix, os; os.environ["a"]="1"; print(os.environ); posix.unsetenv("a=")'

environ({'a': '1'})
Traceback (most recent call last):
  File "<string>", line 1, in <module>
OSError: [Errno 22] Invalid argument
```
Python behaviour changes, depending on the version:

• Python 2.6 —> NO ERRORS, allows invalid operations !

• PYTHON 2.7 —> OSError: [Errno 22] Invalid argument

• PYTHON 3.1 —> NO ERRORS, allows invalid operations !

**SOLUTION**

Implement a solution to detect architecture and OS, then for each case prevent the usage of 'key-value' pairs associated to environment variable that are empty or invalid for several OS.
buffer=ctypes.create_string_buffer(8)
buffer.value='a\0bc1234'

print "Original value    => %r" % (buffer.raw,)
print "Interpreted value => %r" % (buffer.value,)

The ctypes module **truncates** NUL-containing strings.

Original value    => 'a\x00bc1234'
Interpreted value => 'a'

This behaviour is consistent with how C handles string, by considering a NUL character as a line terminator. Python in this case, by using ctypes, is inheriting the same logic therefore the string is silently truncated.

**SOLUTION**

Implement data validation to detect NUL-containing strings to protect them, or avoid using ctypes.
try:
    if 0:
        yield 5
    print("T1-FAIL")
except Exception as e:
    print("T1-PASS")
pass

try:
    if False:
        yield 5
    print("T2-FAIL")
except Exception as e:
    print(repr(e))
pass

**SOLUTION**

Solved in latest Python 2.7.x, avoid constructs like “if 0:”, “if False:”, “while 0:” “while False:”.
We will talk about..

1. Date and time —> time, datetime, os

2. Numbers —> ctypes, xrange, len, decimal

3. Strings —> input, eval, codecs, os, ctypes

4. Files —> sys, os, io, pickle, cPickle

5. Protocols —> socket, poplib, urllib, urllib2
By exploiting the fact that pickle is **NOT** designed to be safe/secure, we can make it execute **whatever we want**.

In this case we are asking to pickle to load a string that is specially formatted that makes it executable by python.

Pickle loads the string and by processing it executes `"ls -la /"`. 
### Result of the command “ls -la /”

<table>
<thead>
<tr>
<th>Permission</th>
<th>User</th>
<th>Group</th>
<th>Size</th>
<th>Date/Time</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>drwxr-xr-x</td>
<td>root</td>
<td>root</td>
<td>4096</td>
<td>Feb 28 01:42</td>
<td>.</td>
</tr>
<tr>
<td>drwxr-xr-x</td>
<td>root</td>
<td>root</td>
<td>4096</td>
<td>Feb 28 01:42</td>
<td>..</td>
</tr>
<tr>
<td>drwxr-xr-x</td>
<td>root</td>
<td>root</td>
<td>4096</td>
<td>Feb 28 01:14</td>
<td>bin</td>
</tr>
<tr>
<td>drwxr-xr-x</td>
<td>root</td>
<td>root</td>
<td>4096</td>
<td>Feb 28 01:57</td>
<td>boot</td>
</tr>
<tr>
<td>drwxr-xr-x</td>
<td>root</td>
<td>root</td>
<td>3680</td>
<td>May  2 14:28</td>
<td>dev</td>
</tr>
<tr>
<td>drwxr-xr-x</td>
<td>root</td>
<td>root</td>
<td>12288</td>
<td>Apr 30 22:16</td>
<td>etc</td>
</tr>
<tr>
<td>drwxr-xr-x</td>
<td>root</td>
<td>root</td>
<td>4096</td>
<td>Feb 28 00:45</td>
<td>home</td>
</tr>
<tr>
<td>lrwxrwxrwx</td>
<td>root</td>
<td>root</td>
<td>30</td>
<td>Feb 27 23:29</td>
<td>initrd.img -&gt; /boot/initrd.img-3.2.0-4-amd64</td>
</tr>
<tr>
<td>drwxr-xr-x</td>
<td>root</td>
<td>root</td>
<td>4096</td>
<td>Feb 28 01:54</td>
<td>lib</td>
</tr>
<tr>
<td>drwxr-xr-x</td>
<td>root</td>
<td>root</td>
<td>4096</td>
<td>Feb 27 23:31</td>
<td>lib64</td>
</tr>
<tr>
<td>drwx------</td>
<td>root</td>
<td>root</td>
<td>16384</td>
<td>Feb 27 23:25</td>
<td>lost+found</td>
</tr>
</tbody>
</table>

### Linux

<table>
<thead>
<tr>
<th>Permission</th>
<th>User</th>
<th>Group</th>
<th>Size</th>
<th>Date/Time</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>drwxr-xr-x</td>
<td>root</td>
<td>wheel</td>
<td>1122</td>
<td>12 Oct 18:58</td>
<td>.</td>
</tr>
<tr>
<td>drwxr-xr-x</td>
<td>root</td>
<td>wheel</td>
<td>1122</td>
<td>12 Oct 18:58</td>
<td>..</td>
</tr>
<tr>
<td>drwxrwxr+x+</td>
<td>root</td>
<td>admin</td>
<td>4148</td>
<td>10 Oct 15:19</td>
<td>Applications</td>
</tr>
<tr>
<td>drwxr-xr-x+</td>
<td>root</td>
<td>wheel</td>
<td>2312</td>
<td>3 Sep 10:47</td>
<td>Library</td>
</tr>
<tr>
<td>drwxr-xr-x+x</td>
<td>root</td>
<td>wheel</td>
<td>68</td>
<td>24 Aug 2013</td>
<td>Network</td>
</tr>
<tr>
<td>drwxr-xr-x+x</td>
<td>root</td>
<td>wheel</td>
<td>136</td>
<td>13 Jul 07:28</td>
<td>System</td>
</tr>
<tr>
<td>drwxr-xr-x+x</td>
<td>root</td>
<td>admin</td>
<td>238</td>
<td>8 Oct 11:23</td>
<td>Users</td>
</tr>
<tr>
<td>drwxrwxrwt@</td>
<td>root</td>
<td>admin</td>
<td>170</td>
<td>14 Oct 10:41</td>
<td>Volumes</td>
</tr>
<tr>
<td>drwxr-xr-x+x</td>
<td>root</td>
<td>wheel</td>
<td>1326</td>
<td>13 Jul 14:14</td>
<td>bin</td>
</tr>
<tr>
<td>drwxrwxr-t@</td>
<td>root</td>
<td>admin</td>
<td>68</td>
<td>24 Aug 2013</td>
<td>cores</td>
</tr>
<tr>
<td>dr-xr-xr-x</td>
<td>root</td>
<td>wheel</td>
<td>7937</td>
<td>12 Oct 18:57</td>
<td>dev</td>
</tr>
</tbody>
</table>

### Mac OS X

<table>
<thead>
<tr>
<th>Permission</th>
<th>User</th>
<th>Group</th>
<th>Size</th>
<th>Date/Time</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>drwxr-xr-x</td>
<td>root</td>
<td>wheel</td>
<td>1122</td>
<td>12 Oct 18:58</td>
<td>.</td>
</tr>
<tr>
<td>drwxr-xr-x</td>
<td>root</td>
<td>wheel</td>
<td>1122</td>
<td>12 Oct 18:58</td>
<td>..</td>
</tr>
<tr>
<td>drwxrwxr+x+</td>
<td>root</td>
<td>admin</td>
<td>4148</td>
<td>10 Oct 15:19</td>
<td>Applications</td>
</tr>
<tr>
<td>drwxr-xr-x+x</td>
<td>root</td>
<td>wheel</td>
<td>2312</td>
<td>3 Sep 10:47</td>
<td>Library</td>
</tr>
<tr>
<td>drwxr-xr-x+x</td>
<td>root</td>
<td>wheel</td>
<td>68</td>
<td>24 Aug 2013</td>
<td>Network</td>
</tr>
<tr>
<td>drwxr-xr-x+x</td>
<td>root</td>
<td>wheel</td>
<td>136</td>
<td>13 Jul 07:28</td>
<td>System</td>
</tr>
<tr>
<td>drwxr-xr-x+x</td>
<td>root</td>
<td>admin</td>
<td>238</td>
<td>8 Oct 11:23</td>
<td>Users</td>
</tr>
<tr>
<td>drwxrwxrwt@</td>
<td>root</td>
<td>admin</td>
<td>170</td>
<td>14 Oct 10:41</td>
<td>Volumes</td>
</tr>
<tr>
<td>drwxr-xr-x+x</td>
<td>root</td>
<td>wheel</td>
<td>1326</td>
<td>13 Jul 14:14</td>
<td>bin</td>
</tr>
<tr>
<td>drwxrwxr-t@</td>
<td>root</td>
<td>admin</td>
<td>68</td>
<td>24 Aug 2013</td>
<td>cores</td>
</tr>
<tr>
<td>dr-xr-xr-x</td>
<td>root</td>
<td>wheel</td>
<td>7937</td>
<td>12 Oct 18:57</td>
<td>dev</td>
</tr>
</tbody>
</table>
import os
import cPickle
import traceback
import sys

# bignum = int((2**31)-1) # 2147483647 -> OK
bignum = int(2**31) # 2147483648 -> Max 32bit -> Crash
random_string = os.urandom(bignum)
print("STRING-LENGTH-1=%r") % (len(random_string))
fout = open('test.pickle', 'wb')
try:
    cPickle.dump(random_string, fout)
except Exception as e:
    print("###### ERROR-WRITE ######")
    print sys.exc_info()[0]
    raise
fout.close()
fin = open('test.pickle', 'rb')
try:
    random_string2 = cPickle.load(fin)
except Exception as e:
    print("###### ERROR-READ ######")
    print sys.exc_info()[0]
    raise
print("STRING-LENGTH-2=%r") % (len(random_string2))
print random_string == random_string2
sys.exit(0)

pickles / cPickles

Depending on the Python version used, pickle or cPickle will either save truncated data without error, or save a portion with a max size limited to 32bit size.

And depending on how Python has been compiled when installed in the system, it may return errors on either the size of random data requested, or report an OS error as invalid argument.
STRING-LENGTH-1=2147483648
############ ERROR-WRITE ############
<type 'exceptions.MemoryError'>
Traceback (most recent call last):
...
    pickle.dump(random_string, fout)
SystemError: error return without exception set

STRING-LENGTH-1=2147483648
############ ERROR-WRITE ############
<type 'exceptions.MemoryError'>
Traceback (most recent call last):
...
File "/usr/lib/python2.7/pickle.py", line 488, in save_string
    self.write(STRING + repr(obj) + '\n')
MemoryError

**SOLUTION**

Implement strong data validation to be sure that nothing dangerous will ever be processed, and limit data size to 32bit sizes even in 64bit systems.
```python
import os
import sys
FPATH = 'bug2091.test'

print 'wa (1)_write1'
with open(FPATH, 'wa') as fp:
    fp.write('test1-')

print repr(fp.read())

print 'wa (2)_write3'
with open(FPATH, 'wa+') as fp:
    fp.write('test3-')

print repr(fp.read())

print 'aw_write4'
with open(FPATH, 'aw') as fp:
    fp.write('test4-')

print repr(fp.read())

print 'rU+_write2'
with open(FPATH, 'rU+') as fp:
    fp.write('test2-')

print repr(fp.read())

print 'rU+_read1'
with open(FPATH, 'rU+') as fp:
    print repr(fp.read())

print 'read_2'
with open(FPATH, 'read') as fp:
    print repr(fp.read())

os.unlink(FPATH)
sys.exit(0)
```

'U' -> universal newline mode (deprecated)
## Testing file operations on Linux and Mac OS X

<table>
<thead>
<tr>
<th>Test String</th>
<th>Flags</th>
<th>Operation</th>
<th>Expected Result</th>
<th>Test Result (LINUX-OS X)</th>
</tr>
</thead>
</table>
| test1-      | wa    | 1. truncate and write  
2. write in append mode | Invalid Mode    | test1-                  |
| test2-      | rU+   | 1. read (Universal Newline)  
2. open file in read and write | test2-          | test2-                  |
| test3-      | wa+   | 1. truncate and write  
2. write in append mode | Invalid Mode    | test3-                  |
| test4-      | aw    | 1. write in append mode  
2. truncate and write | Invalid Mode    | test3-test4-            |
|             | rU+   | 1. read (Universal Newline)  
2. open file in read and write | test2-          | test3-test4-            |
|             |       | read, ?, append, ?                    | Invalid Mode    | test3-test4-            |
Testing file operations on Windows

<table>
<thead>
<tr>
<th>Test String</th>
<th>Flags</th>
<th>Operation</th>
<th>Expected Result</th>
<th>Test Result (WINDOWS)</th>
</tr>
</thead>
</table>
| test1-      | wa    | 1. truncate and write  
2. write in append mode | Invalid Mode | Invalid Mode |
| test2-      | rU+   | 1. read (Universal Newline)  
2. open file in read and write | test2- | test2- |
| test3-      | wa+   | 1. truncate and write  
2. write in append mode | Invalid Mode | Invalid Mode |
| test4-      | aw    | 1. write in append mode  
2. truncate and write | Invalid Mode | Invalid Mode |
| rU+         |       | 1. read (Universal Newline)  
2. open file in read and write | test2- | test2- |
| read        |       | read, ?, append, ? | Invalid Mode | Invalid Mode |
The code is trying to write a non zero amount of output to something that never reads from standard input.

In this case the file descriptor has been closed and nothing can be sent.
import io
import sys

fd = io.open(sys.stdout.fileno(), 'wb')
fd.close()
sys.stdout.write("Crash")

**SOLUTION**

File and stream libraries are **NOT** following OS specifications but instead they use a general logic. It is necessary to implement a library capable of handling, for each OS, the proper set of calls.
import os
import sys
testfile = 'tempA'
with open(testfile, "ab") as f:
    f.write(b"abcd")
    f.write(b"x" * (1024 ** 2))

import io
testfilea = 'tempB'
with io.open(testfilea, "ab") as f:
    f.write(b"abcd")
    f.write(b"x" * (1024 ** 2))

To check Python behaviour with file writes (on Linux):

    strace python -OOBRttu script.py
**PYTHON 2.6**

Amount of data we want to write = 4 + 1.048.576 = **1.048.580**

Results of ‘strace’ with standard ‘open’ call

```python
write(3, "abcdxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"..., 4096) = 4.096
write(3, "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"..., 1044480) = 1.044.480
```

First call is buffered, instead of writing only 4 (abcd) it writes 4.092 ‘x’
Second call writes ‘x’ for a total of 1.044.480.
Checking the total data written something is not right.

- 1044480 + 4096 = 1.048.576 (missing 4, expected **1.048.580**)
Waiting 5 second ‘fix’ the problem as the OS has flushed the cache.

Results of ‘strace’ by using ‘io’ module

```python
write(3, "abcd", 4) = 4
write(3, "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"..., 1048576) = 1.048.576
```
All is fine if we use the ‘io’ module.
Amount of data we want to write = 4 + 1.048.576 = 1.048.580

Results of ‘strace’ with standard ‘open’ call

write(3, "abcdxxxxxxxxxxxxxxxxxxxxxxxxxxxx"..., 4096) = 4.096
write(3, "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"..., 1044480) = 1.044.480
write(3, "xxxx", 4)                     = 4

First call is buffered, instead of writing only 4 (abcd) it writes 4.092 ‘x’
Second call writes ‘x’ for a total of 1.044.480.
Third call will write the remaining ‘x’, and written data is correct.
Only ‘problem’ is that we were expecting ‘2’ calls and NOT ‘3’.

Results of ‘strace’ by using ‘io’ module

write(3, "xxxx", 4)                     = 4
write(3, "abcdxxxxxxxxxxxxxxxxxxxxxxxxxxxx"..., 1048580) = 1048580
All is fine if we use the ‘io’ module.
PYTHON 3.x

Amount of data we want to write = 4 + 1.048.576 = 1.048.580

Results of ‘strace’ with standard ‘open’ call
\[\text{write}(3, "abcd", 4) = 4\]
\[\text{write}(3, "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"..., 1048576) = 1.048.576\]
All is fine if we use the standard ‘open’ call.

Results of ‘strace’ by using ‘io’ module
\[\text{write}(3, "abcd", 4) = 4\]
\[\text{write}(3, "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"..., 1048576) = 1.048.576\]
All is fine if we use the ‘io’ module.

SOLUTION

Atomic operation are NOT guaranteed in Python 2, core library are using the cache to read and write.
The ‘io’ module should be used when possible.
We will talk about..

1. Date and time —> time, datetime, os
2. Numbers —> ctypes, xrange, len, decimal
3. Strings —> input, eval, codecs, os, ctypes
4. Files —> sys, os, io, pickle, cpickle
5. Protocols —> socket, poplib, urllib, urllib2
Core libraries are OS independent, developer must know how to create proper communication channels for each OS, the library will permit to execute operation that are not safe and not correct.

If a client connects to the HTTP server and then we close the server, python will **NOT** release resources, the OS will **NOT** release the socket.
import socket
import SimpleHTTPServer
import SocketServer
PORT = 8080
# ESSENTIAL: socket resuse is setup BEFORE it is bound.
# This will avoid TIME_WAIT issues and socket in use errors
class MyTCPServer(SocketServer.TCPServer):
    def server_bind(self):
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.socket.bind(self.server_address)
    def do_GET(self):
        self.send_response(200)
        self.end_headers()
Handler = SimpleHTTPServer.SimpleHTTPRequestHandler
Handler.do_GET = do_GET
httpd = MyTCPServer(("", PORT), Handler)
httpd.serve_forever()
import socket
HOST = '127.0.0.1'
PORT = 45678
NULLS = '\0' * (1024 * 1024) # 1 MB
try:
    sock = socket.socket()
    sock.bind((HOST, PORT))
    sock.listen(1)
    while 1:
        print "Waiting connection..."
        conn, _ = sock.accept()
        print "Sending welcome..."
        conn.sendall("+OK THIS IS A TEST\n")
        conn.recv(4096)
        DATA = NULLS
        try:
            while 1:
                print "Sending 1 GB..."
                for _ in xrange(1024):
                    conn.sendall(DATA)
                except IOError, ex:
                    print "Error: %r" % str(ex)
        except Exception, ex:
            print "Error: %r" % str(ex)
            print "End session."
        finally:
            sock.close()
print "End server."

import poplib
import sys
HOST = '127.0.0.1'
PORT = 45678
try:
    print "Connecting to %r:%d..." % (HOST, PORT)
    pop = poplib.POP3(HOST, PORT)
    print "Welcome:", repr(pop.welcome)
    print "Listing..."
    reply = pop.list()
    print "LIST:", repr(reply)
except Exception, ex:
    print "Error: %r" % str(ex)
print "End.
sys.exit(0)

Simple test
1. Start a dummy server
2. Use client to connect to server
3. Server sends NULs
4. Client will keep receiving NULs
5. Client memory if full, OS crash!
Server
Waiting connection...
Sending welcome...
Sending 1 GB...
Error: '[Errno 54] Connection reset by peer'
End session.

If using Python >= 2.7.9, 3.3:
Client
Connecting to '127.0.0.1':45678...
Welcome: '+OK THIS IS A TEST'
Listing...
Error: 'line too long'
End.

If using Python < 2.7.9, 3.3:
Client
Connecting to '127.0.0.1':45678...
Welcome: '+OK THIS IS A TEST'
........
Error: 'out of memory'

SOLUTION
Use ‘Python > 2.7.9’ or ‘Python > 3.3’, if not possible implement controls to check for data type and size.
Libraries with “Unlimited data“ issues

<table>
<thead>
<tr>
<th>Library</th>
<th>Link to Python bug</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPLIB</td>
<td><a href="http://bugs.python.org/issue16037">http://bugs.python.org/issue16037</a></td>
</tr>
<tr>
<td>FTPLIB</td>
<td><a href="http://bugs.python.org/issue16038">http://bugs.python.org/issue16038</a></td>
</tr>
<tr>
<td>IMAPLIB</td>
<td><a href="http://bugs.python.org/issue16039">http://bugs.python.org/issue16039</a></td>
</tr>
<tr>
<td>NNTPLIB</td>
<td><a href="http://bugs.python.org/issue16040">http://bugs.python.org/issue16040</a></td>
</tr>
<tr>
<td>POPLIB</td>
<td><a href="http://bugs.python.org/issue16041">http://bugs.python.org/issue16041</a></td>
</tr>
<tr>
<td>SMTPLIB</td>
<td><a href="http://bugs.python.org/issue16042">http://bugs.python.org/issue16042</a></td>
</tr>
<tr>
<td>XMLRPC</td>
<td><a href="http://bugs.python.org/issue16043">http://bugs.python.org/issue16043</a></td>
</tr>
</tbody>
</table>
```python
import os
import io
import urllib2
#but all fine with urllib
domain = 'ftp://ftp.ripe.net'
location = '/pub/stats/ripencc/
file = 'delegated-ripencc-extended-latest'
url = domain + location + file
data = urllib2.urlopen(url).read()
with io.open(file, 'wb') as w:
    w.write(data)
file_size = os.stat(file).st_size
print "Filesize: %s" % (file_size)
```

urllib2 does NOT have proper logic to handle data streams and fails silently.

**Wrong file sizes**
- Filesize: 65536
- Filesize: 32768
- Filesize: 49152

**Every proper size**
- Filesize: 6598450
- Filesize: 6598450
- Filesize: 6598450

**SOLUTION**
Make use of the OS.
## Known Unsafe Libs

<table>
<thead>
<tr>
<th>ast</th>
<th>multiprocessing</th>
<th>reexec</th>
</tr>
</thead>
<tbody>
<tr>
<td>bastion</td>
<td>os.exec</td>
<td>shelve</td>
</tr>
<tr>
<td>commands</td>
<td>os.popen</td>
<td>subprocess</td>
</tr>
<tr>
<td>cookie</td>
<td>os.spawn</td>
<td>tarfile</td>
</tr>
<tr>
<td>cPickle / pickle</td>
<td>os.system</td>
<td>urllib2</td>
</tr>
<tr>
<td>eval</td>
<td>parser</td>
<td>urlparse</td>
</tr>
<tr>
<td>marshal</td>
<td>pipes</td>
<td>yaml</td>
</tr>
<tr>
<td>mktemp</td>
<td>pty</td>
<td>zipfile</td>
</tr>
</tbody>
</table>
Closing comments:

- Python is a great language, we like it very much and we will keep using it.

- Everything used to make this slides has been in the public domain for years, is just difficult to find.

- Do NOT assume something is working as it should just because millions of people are using it, and definitely do NOT assume is doing it safely.
Thank You

Enrico Branca
Security Researcher
enrico.branca@awebof.info
enrico.branca@owasp.org

OWASP Python Security project
https://github.com/ebranca/owasp-pysec/wiki