Information Security and Python

OWASP Chapter meeting #10

7th August 2014, Bucureşti, România

OWASP
The Open Web Application Security Project

OWASP
The Open Web Application Security Project

• PYTHON

“Python is Now (July 2014) the Most Popular Introductory Teaching Language at Top U.S. Universities, eight of the top 10 CS departments (80%) and 27 of the top 39 (69%) teach Python in introductory courses.”

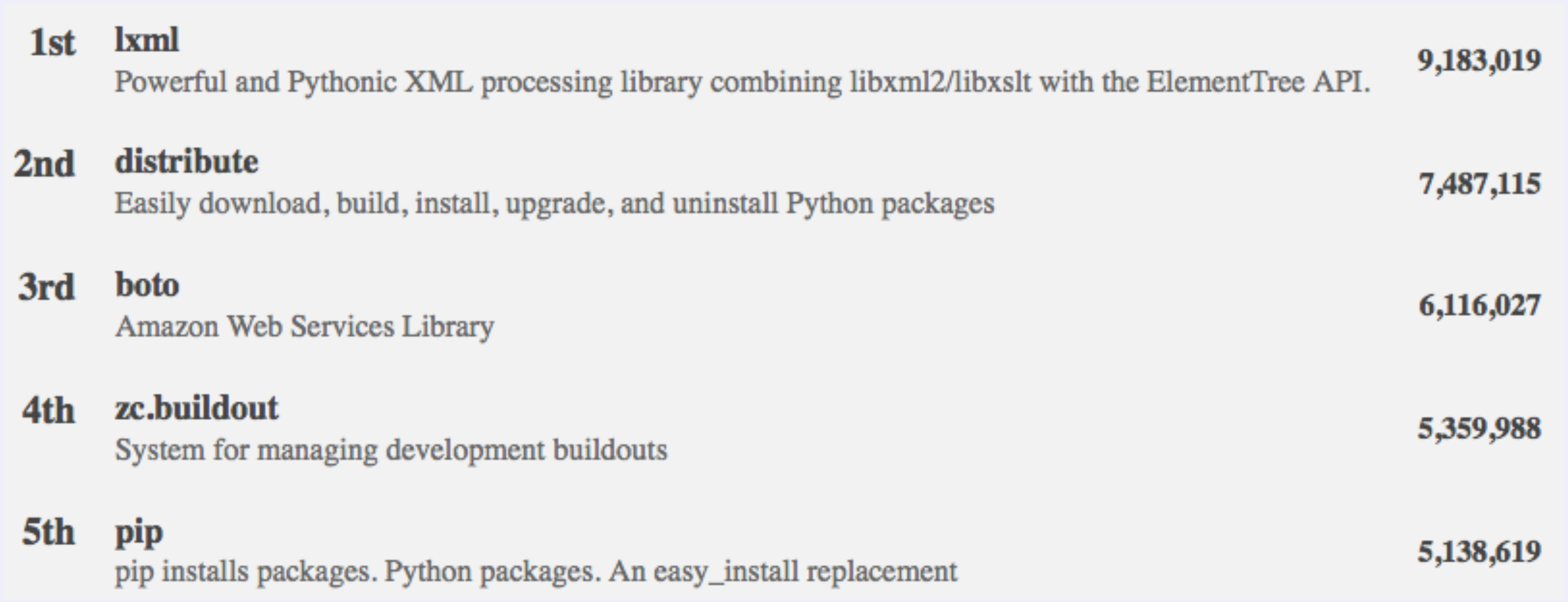
Any University in Europe teaching Python?

University of Banja Luka	**Universitat Politècnica de Catalunya**
University of Southampton	**University of Évora**
Coventry University	**Université Paris Sud**
University of Oxford	**Université Paris-Dauphine**

OWASP
The Open Web Application Security Project

• Top used python packages?

(http://pypi-ranking.info/alltime)



1st	**lxml** Powerful and Pythonic XML processing library combining libxml2/libxslt with the ElementTree API.	9,183,019
2nd	**distribute** Easily download, build, install, upgrade, and uninstall Python packages	7,487,115
3rd	**boto** Amazon Web Services Library	6,116,027
4th	**zc.buildout** System for managing development buildouts	5,359,988
5th	**pip** pip installs packages. Python packages. An easy_install replacement	5,138,619

OWASP
The Open Web Application Security Project

• PYTHON for reverse engineering?

Androguard	*IDAPython*	*pyasm2*	*pype32*
apkjet	*libdisassemble*	*PyBFD*	*python-adb*
AsmJit-Python	*llvmpy*	*PyCodin*	*python-ptrace*
BeaEnginePython	*Miasm*	*pydasm*	*PythonGdb*
Binwalk	*ollydbg2-python*	*PyDBG*	*PyVEX*
Buggery	*OllyPython*	*pydbgr*	*pywindbg*
cuckoo	*PDBparse*	*PyELF*	*Rekall*
Disass	*pefile*	*pyew*	*Vivisect*
ElfParserLib	*PIDA*	*pygdb2*	*Volatility*
Frida	*PyADB*	*pyMem*	*WinAppDbg*

OWASP
The Open Web Application Security Project

• PYTHON for networking?

Scapy	*libdnet*	*dpkt*	*Impacket*
pypcap	*pynids*	*Dirtbags py-pcap*	*flowgrep*
Mallory	*Pytbull*	*0trace*	

• PYTHON for fuzzing?

Sulley	*Peach Fuzzing*	*antiparser*	*TAOF*
untidy	*Powerfuzzer*	*Mistress*	*Fuzzbox*
WSBang	*Construct*	*Fusil*	*SMUDGE*

OWASP
The Open Web Application Security Project

- **PYTHON for the web?**

Requests	*HTTPie*	*ProxMon*	*WSMap*
Twill	*Ghost*	*Windmill*	*FunkLoad*
spynner	*mitmproxy*	*pathod / pathoc*	*scrapy*

- **PYTHON for offensive actions?**

Plenty of ***dangerous*** python tools in “packet storm security” website:

- http://packetstormsecurity.com/files/tags/python/

More general tools:

- http://pythonsource.com/

OWASP
The Open Web Application Security Project

- **PYTHON → use with moderation**

We have seen some powerful tools written in python but what about the security of python itself?

- Are there operations to avoid?
- Any module or core library to use with caution?
- Something to know before writing code for security?

OWASP
The Open Web Application Security Project

• OWASP Python Security Project

Python Security is a free, open source, OWASP Project that aims at creating a hardened version of python that makes it easier for security professionals and developers to write applications more resilient to attacks and manipulations.

Our code in GITHUB:

- https://github.com/ebranca/owasp-pysec/

Known Issues in python modules concerning software security:

- https://github.com/ebranca/owasp-pysec/wiki/Security-Concerns-in-modules-and-functions

OWASP
The Open Web Application Security Project

• OWASP Python Security Project

A new ambitious project that aims at making python more secure and viable for usage in sensitive environments.

- We have started a full security review of python by checking core modules written in both C and python
- First goal is to have a secure layer of modules for LINUX

The security review takes a lot of time and we are slowly publishing libraries and tools, documentation will follow ☺

OWASP
The Open Web Application Security Project

• EXAMPLE – numeric overflow

```
N = 2 ** 63
for n in xrange(N):
    print n
```

<u>RESULT (debian 7 x64)</u>

```
Traceback (most recent call last):
  File "xrange_overflow.py", line 5, in <module>
    for n in xrange(N):
OverflowError: Python int too large to convert to C long
```

PROBLEM: xrange uses "Plain Integer Objects" created by the OS
SOLUTION: Use python "long integer object“ that will allow numbers of arbitrary length as the limit will be the system's memory.

• EXAMPLE – operations with file descriptors

```
import sys
import io

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

try:
    sys.stdout.write("test for error")
except Exception:
    raise
```

RESULT

```
close failed in file object destructor:
sys.excepthook is missing
lost sys.stderr
```

This is happening because the code is trying to write a non zero amount of data to something that does not exist. In this case the file descriptor has been closed and nothing can be sent, but python has no controls for this kind of actions and wrongly returns a system error.

OWASP
The Open Web Application Security Project

• EXAMPLE – object evaluation in JSON

According to "http://www.ietf.org/rfc/rfc4627.txt" a JSON object is either a list or a dictionary with other elements in it. At the moment Python's implementation does not follow standards and took liberty to parse also many object types.

<u>CORRECT BEHAVIOUR</u>

JSON LOADS LIST! '[[3], [5], [7]]' -- JSON LOADS DICTIONARY! '{"abc": 456}'
JSON UNABLE TO LOAD COMPLEX -- JSON UNABLE TO LOAD BYTEARRAY

<u>WRONG BEHAVIOUR (parsed withour error or warning)</u>

JSON LOADS STRING 42 -- JSON LOADS INTEGER '3'
JSON LOADS FLOAT '3.141592' -- JSON LOADS TUPLE! '[1, 2, 3]'

OWASP
The Open Web Application Security Project

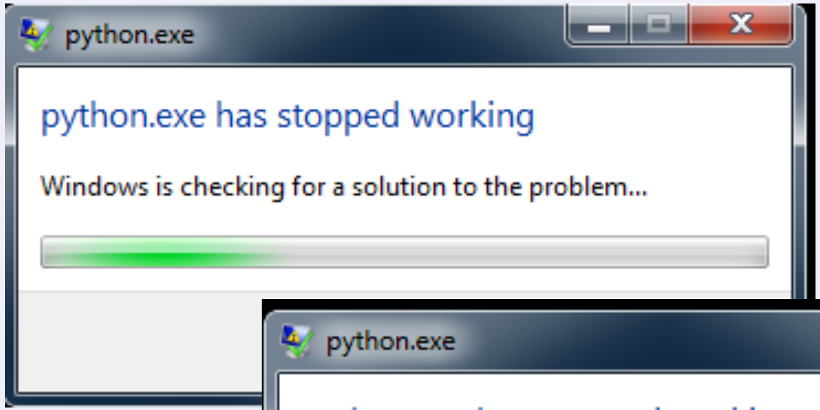
```
import sys
import json
try:
   b = json.loads('42')
   print ("JSON LOADS STRING %r") % (b,)
except Exception:
   print "JSON UNABLE TO LOAD STRING"
try:
   c = json.dumps(int(3.141592))
   print ("JSON LOADS INTEGER %r") % (c,)
except Exception:
   print "JSON UNABLE TO LOAD INTEGER"
try:
   d = json.dumps(float(3.141592))
   print ("JSON LOADS FLOAT %r") % (d,)
except Exception:
   print "JSON UNABLE TO LOAD FLOAT"
try:
   e = json.dumps(complex(3.141592))
   print ("JSON LOADS COMPLEX! %r") % (e,)
except Exception:
   print "JSON UNABLE TO LOAD COMPLEX"
```

```
try:
   f = json.dumps([[3], [5], [7]])
   print ("JSON LOADS LIST! %r") % (f,)
except Exception:
   print "JSON UNABLE TO LOAD LIST"
try:
   f = json.dumps((1, 2, 3))
   print ("JSON LOADS TUPLE! %r") % (f,)
except Exception:
   print "JSON UNABLE TO LOAD TUPLE"
try:
   g = json.dumps({ 'abc': 456 })
   print ("JSON LOADS DICTIONARY! %r") % (g,)
except Exception:
   print "JSON UNABLE TO LOAD DICTIONARY"
try:
   h = json.dumps(bytearray("hello"))
   print ("JSON LOADS BYTEARRAY! %r") % (h,)
except Exception:
   print "JSON UNABLE TO LOAD BYTEARRAY"
```

OWASP
The Open Web Application Security Project

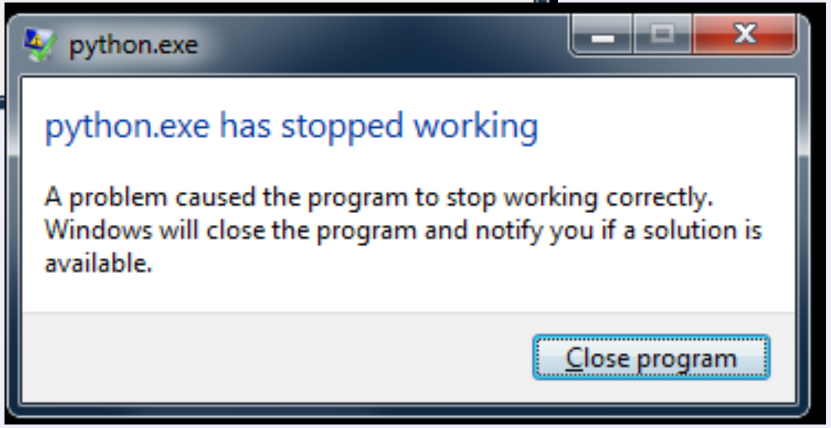
• EXAMPLE - File descriptors in Windows

```
C:\Python27>python.exe -V
Python 2.7.6

python.exe -OOBtt “winfd_1.py”

############################
import io
import sys
fd = io.open(sys.stdout.fileno(), 'wb')
fd.close()
sys.stdout.write(“Now writing to stdout closed FD will cause a crash")
############################
```


python.exe

python.exe has stopped working

Windows is checking for a solution to the problem...


python.exe

python.exe has stopped working

A problem caused the program to stop working correctly. Windows will close the program and notify you if a solution is available.

• EXAMPLE - File operations with “tar”

If a malicious malicious .tar file is created with entries containing absolute or relative paths the tarfile module happily uses them as is without sanity checking.

Problem in “***TarFile”*** libray at line 2088:

“self._extract_member(tarinfo, os.path.join(path, tarinfo.name))”

When a join operation between two strings takes place, **if the second string has "/" as first character then the first string is ignored.**

os.path.join('test.txt', '/usr/lib/zlib.so') → **'/usr/lib/zlib.so'**

Furthermore **"join" does not sanitize the file path allowing names with '../'** in fact allowing malicious operation to be conducted, all without any error raised.

• EXAMPLE – instruction validation

```
import sys

((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((
((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((
1
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))

sys.exit(0)
```

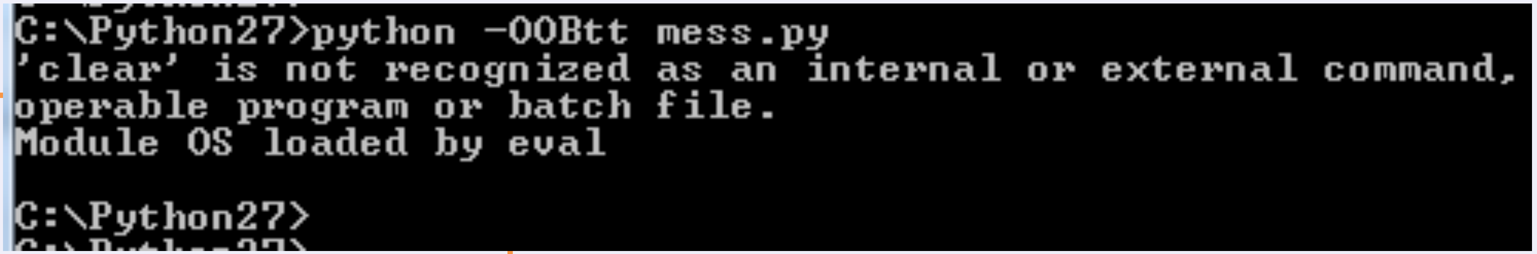
<u>RESULTS</u>

```
python -OOBRtt test.py
s_push: parser stack overflow
MemoryError
```

Python has internal recursion limits but is lacking controls on code recursion, hence the MemoryError where in fact is a recursion error generated by the interpreter.

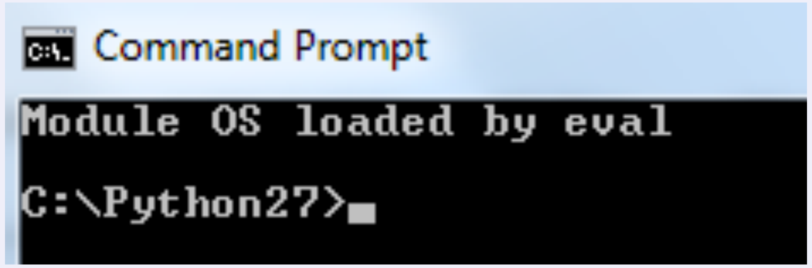
OWASP
The Open Web Application Security Project

• EXAMPLE – string evaluation



```
C:\Python27>python -OOBtt mess.py
'clear' is not recognized as an internal or external command,
operable program or batch file.
Module OS loaded by eval

C:\Python27>
```

```
import sys
import os
try:
    eval("__import__('os').system('clear')", {})
    #eval("__import__('os').system(cls')", {})
    print "Module OS loaded by eval"
except Exception as e:
    print repr(e)
```



```
Module OS loaded by eval

C:\Python27>
```

In python the function "eval" executes a string but is not possible to any control to the operation. Any malicious code will be executed without limits in the context of the user that loaded the interpreter. This function is **REALLY DANGEROUS** if used to parse user input or strings from untrusted sources.

OWASP
The Open Web Application Security Project

• EXAMPLE – input evaluation

```
Secret = "A SECRET DATA"
Public = "a BANANA"
value = input("Please enter your age ")
print "There are",value,
print "monkeys looking for",Public
```

What you type as input is interpreted through an expression and the result is saved into your target variable with no control or limits.

```
python -OOBRtt input_1.py
Please enter your age 32
There are 32 monkeys looking for a BANANA

python -OOBRtt input_1.py
Please enter your age dir()
There are ['Public', 'Secret', '__builtins__', '__doc__', '__file__', '__name__', '__package__'] monkeys
looking for a BANANA

python -OOBRtt input_1.py
Please enter your age Secret
There are A SECRET DATA monkeys looking for a BANANA
```

The dir() function returns "most" of the attributes of an object.

• EXAMPLE – variable manipulation

```
def test(first_arg, second_arg):
   return (first_arg + second_arg)

def main():
   if len(sys.argv) > 3:
      return test(1, test)
   else:
      return test(1, 2)
```

```
python -W error -OOBRtt operator_1.py 0
python -W error -OOBRtt operator_1.py 0 1
python -W error -OOBRtt operator_1.py 0 1 2
Traceback (most recent call last):
File "operator_1.py", line 5, in test
   return first_arg + second_arg
TypeError: unsupported operand type(s) for +: 'int'
and 'function'
```

In python variables are not "statically typed" therefore is possible to have a valid part code pointing to another section of the code only under certain conditions, thus escaping normal testing procedures. A possible solution would be to implement a module to check object type, length and reference, and to raise an exception (TypeError or Value Error) avoiding intermediate operations.

- Unicode string encode/decode

<u>**RESULT**</u>

Correct-String "u'A\\ufffdBC\\ufffd'"	→	KNOWN GOOD STRING
CODECS-String "u'A\\ufffdBC'"	→	WRONG
IO-String "u'A\\ufffdBC\\ufffd'"	→	OK

The problem is due to a bug in the "codec" library that detects the character "F4" and assumes this is the first character of a sequence of characters therefore and wait to receive the remaining 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.

- CODE – Unicode string encode/decode

```
import codecs
import io
try:
   ascii
except NameError:
   ascii = repr
b = b'\x41\xF5\x42\x43\xF4'
print("Correct-String %r") % ((ascii(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: ←    ISSUE HERE
   print("CODECS-String %r") % (ascii(fin.read()))
with io.open('temp.bin', 'rt', encoding='utf8', errors='replace') as fin:
   print("IO-String %r") % (ascii(fin.read()))
```

OWASP
The Open Web Application Security Project

• EXAMPLE – data corruption with “cPickle”

```
import os
import cPickle
import traceback
random_string = os.urandom(int(2147483648))
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
```

pickle/CPICKLE (debian 7 x64)

LIMIT = 2147483648 -1 = 2147483647

(32bit integer object)

TEST WITH STRING SIZE "2147483647"

ALL OK

TEST using cPickle (data corruption)

```
TEST WITH STRING SIZE "2147483648"
###### ERROR-WRITE ######
<type 'exceptions.SystemError'>
Traceback (most recent call last):
  File "pickle_2.py", line 18, in <module>
    pickle.dump(random_string, fout)
SystemError: error return without exception set
```

EXAMPLE – data corruption with “pickle”

```
import os
import pickle
import traceback
random_string = os.urandom(int(2147483648))
print ("STRING-LENGTH-1=%r") % (len(random_string))
fout = open('test.pickle', 'wb')
try:
   pickle.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 = pickle.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
```

pickle/CPICKLE (debian 7 x64)

LIMIT = 2147483648 -1 = 2147483647

(32bit integer object)

TEST WITH STRING SIZE "2147483647"

ALL OK

TEST using pickle (data corruption)

```
TEST WITH STRING SIZE "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
```

OWASP
The Open Web Application Security Project

• EXAMPLE – unrestricted code in “pickle”

```
import pickle
obj = pickle.load(open('./bug.pickle'))
print "== Object =="
print repr(obj)
```

bug.pickle

```
cos
system
(S'ls -al /'
tR.
```

```
drwxr-xr-x  24 root root  4096 Feb 28 01:42 .
drwxr-xr-x  24 root root  4096 Feb 28 01:42 ..
drwxr-xr-x   2 root root  4096 Feb 28 01:14 bin
drwxr-xr-x 158 root root 12288 Apr 30 22:16 etc
drwxr-xr-x   3 root root  4096 Feb 28 00:45 home
drwx------   2 root root 16384 Feb 27 23:25 lost+found
drwxr-xr-x   3 root root  4096 May  2 09:18 media
drwxr-xr-x   2 root root  4096 Dec  4 12:31 mnt
drwxr-xr-x   2 root root  4096 Feb 27 23:26 opt
dr-xr-xr-x 316 root root     0 Apr 16 12:21 proc
drwx------   7 root root  4096 Mar  7 23:09 root
drwxr-xr-x   2 root root  4096 Feb 28 01:55 sbin
drwxr-xr-x   2 root root  4096 Feb 27 23:26 srv
drwxr-xr-x  13 root root     0 Apr 16 12:21 sys
drwxrwxrwt  13 root root  4096 May  2 14:57 tmp
drwxr-xr-x  10 root root  4096 Feb 27 23:26 usr
drwxr-xr-x  13 root root  4096 Feb 28 07:21 var
```

NEVER use ANYTHING that uses pickle or cPickle as is NOT designed as safe/secure solution for serialization

OWASP
The Open Web Application Security Project

- EXAMPLE – inconsistent “pickle” serialization

RESULT

```
b'cUserList\ndefaultdict\nq\x00)Rq\x01.'
b'ccollections\ndefaultdict\nq\x00)Rq\x01.'
b'\x80\x02cUserList\ndefaultdict\nq\x00)Rq\x01.'
b'\x80\x02ccollections\ndefaultdict\nq\x00)Rq\x01.‘
```

(http://hg.python.org/cpython/file/7272ef213b7c/Lib/_compat_pickle.py at line 80)

If there's a “collections.defaultdict” in the pickle dump, python 3 pickles it to “UserString.defaultdict” instead of “collections.defaultdict” **even if python 2.7 and 2.6 do not have a “defaultdict” class in “UserString”.**

```
# python 3
import pickle
import collections
dct = collections.defaultdict()
f = pickle.dumps(dct, protocol=1)
print (repr(f))
g = pickle.dumps(dct, protocol=1,
fix_imports=False)
print (repr(g))
h = pickle.dumps(dct, protocol=2)
print (repr(h))
i = pickle.dumps(dct, protocol=2,
fix_imports=False)
print (repr(i))
```

- EXAMPLE – review of pickle/cPickle
  - **Main problems: code injection, data corruption**

- cPickle: severe errors as exceptions are "lost" even if an error is generated and signalled by the O.S.
- pickle: no controls on data/object integrity
- pickle: no control on data size or system limitations
- pickle: code evaluated without security controls
- pickle: string encoded/decoded without verification

• EXAMPLE – leaks in poplib/urllib/smtplib …

```
python -OOBRtt pop3_server.py
Traceback (most recent call last):
  File "pop3_server.py", line 12, in <module>
    sock.bind((HOST, PORT))
  File "/usr/lib/python2.7/socket.py", line 224, in meth
    return getattr(self._sock,name)(*args)
socket.error: [Errno 98] Address already in use
```

If python process has an error the **exception** will not reliably close all file and socket file descriptors (handles) leading to **leaks** and **uncontrollable** background processes

```
ps aux | grep pop3
user01   30574  0.0  0.0  33256  6052 ?      S    19:34   0:00 /usr/bin/python –OOBRtt pop3_server.py

lsof -P | grep python | grep pop3
pop3_serv 30574          user01  txt       /usr/bin/python2.7
pop3_serv 30574          user01  mem       REG       /usr/lib/python2.7/lib-dynload/_ssl.so
```

OWASP
The Open Web Application Security Project

• EXAMPLE – socket remains open after error ..

OPEN IN TERMINAL 1 (one line):

```
python -m smtpd -n -c DebuggingServer
localhost:45678
```

OPEN IN TERMINAL 2:

```
python -OOBRtt smtplib_1.py
```

smtplib_1.py

```
import smtplib
try:
    s = smtplib.SMTP_SSL("localhost", 45678)
except Exception:
    raise
```

RESULT:

```
ssl.SSLError: [Errno 1] _ssl.c:504: error:140770FC:SSL
routines:SSL23_GET_SERVER_HELLO:unknown protocol

lsof -P | grep python | grep ":45678"
python   16725   user01   3u   IPv4   31510356   0t0   TCP localhost:45678 (LISTEN)
```

The underlying socket connection remains open, but you can't access it or close it.

OWASP
The Open Web Application Security Project

• EXAMPLE – “unlimited data” in POP3

```
python -OOBRtt pop3_client.py
Connecting to '127.0.0.1':45678...
Welcome: '+OK THIS IS A TEST'
Error: 'out of memory‘
```

CLIENT

```
import poplib
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."
```

SERVER

```
import socket
HOST = '127.0.0.1'
PORT = 45678
NULLS = '\0' * (1024 * 1024) # 1 MB
sock = socket.socket()
sock.bind((HOST, PORT))
sock.listen(1)
while 1:
conn, _ = sock.accept()
conn.sendall("+OK THIS IS A TEST\r\n")
   conn.recv(4096)
   DATA = NULLS
   try:
      while 1:
         for _ in xrange(1024):
            conn.sendall(DATA)
   except IOError, ex:
      print "Error: %r" % str(ex)
```

- EXAMPLE – libs with “unlimited data“ issues

HTTPLIB →	http://bugs.python.org/issue16037
FTPLIB →	http://bugs.python.org/issue16038
IMAPLIB →	http://bugs.python.org/issue16039
NNTPLIB →	http://bugs.python.org/issue16040
POPLIB →	http://bugs.python.org/issue16041
SMTPLIB →	http://bugs.python.org/issue16042
XMLRPC →	http://bugs.python.org/issue16043

OWASP
The Open Web Application Security Project

- Small list of **KNOWN UNSAFE** python components

ast	multiprocessing	pty
bastion	os.exec	rexec
commands	os.popen	shelve
cookie	os.spawn	subprocess
cPickle	os.system	tarfile
eval	parser	yaml
marshal	pickle	zipfile
mktemp	pipes	

- **Closing Summary**

- Python is easy to learn and very powerful **BUT** (like cookies) has to be used with care and moderation.
- There are *no limits* or controls in the language, is *responsibility of the coder to know what can be done and what to avoid.*

OWASP
The Open Web Application Security Project

Secure Coding Review

Server Issues
Misconfiguration
Application headers
Application Errors
Default files
Default Locations
Traffic in clear text
Vulnerable to DoS
Vulnerable to MITM

Crypto Issues
Weak ciphers
Small keys
Invalid SSL certs

Access class to Monitor
Local network
Local access only
Remote Network Access

Vulnerabilities to Check
Format String
Buffer Errors
Credentials Management
Cryptographic Issues
Information Leak
Input Validation
OS Command Injections
SQL Injection

Architectural Aspects
Kernel Architecture
Data write policy
NIC configuration
Entropy pool

Language Issues
File operations
Object evaluations
Instruction Validation
Variable Manipulation
String/Input Evaluation
Unicode encode/decode
Serialization
Data limits

OWASP
The Open Web Application Security Project

Contact

Enrico Branca

“OWASP Python Security Project”

http://www.pythonsecurity.org/

Email: enrico.branca@owasp.org
Linkedin: http://fr.linkedin.com/in/ebranca