Secure Coding with Python

OWASP Romania Conference 2014

24th October 2014, Bucureşti, România

OWASP

The Open Web Application Security Project

OWASP
The Open Web Application Security Project

- About Me

Started to work in IT in **1997**, moved to information security in **2001**. Working in information security for over a decade with experience in software security, information security management, and information security R&D.

Worked in many roles like Senior Security Engineer, Security Architect, Disaster Recovery Specialist, Microsoft Security Specialist, etc… etc...

Leader of **"OWASP Python Security" Project**

- http://www.pythonsecurity.org/

Co-Leader of **"OWASP Project Metrics" Project**

- https://github.com/OWASP/OWASP-Project-Metrics

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

- **OWASP Python Security Project**

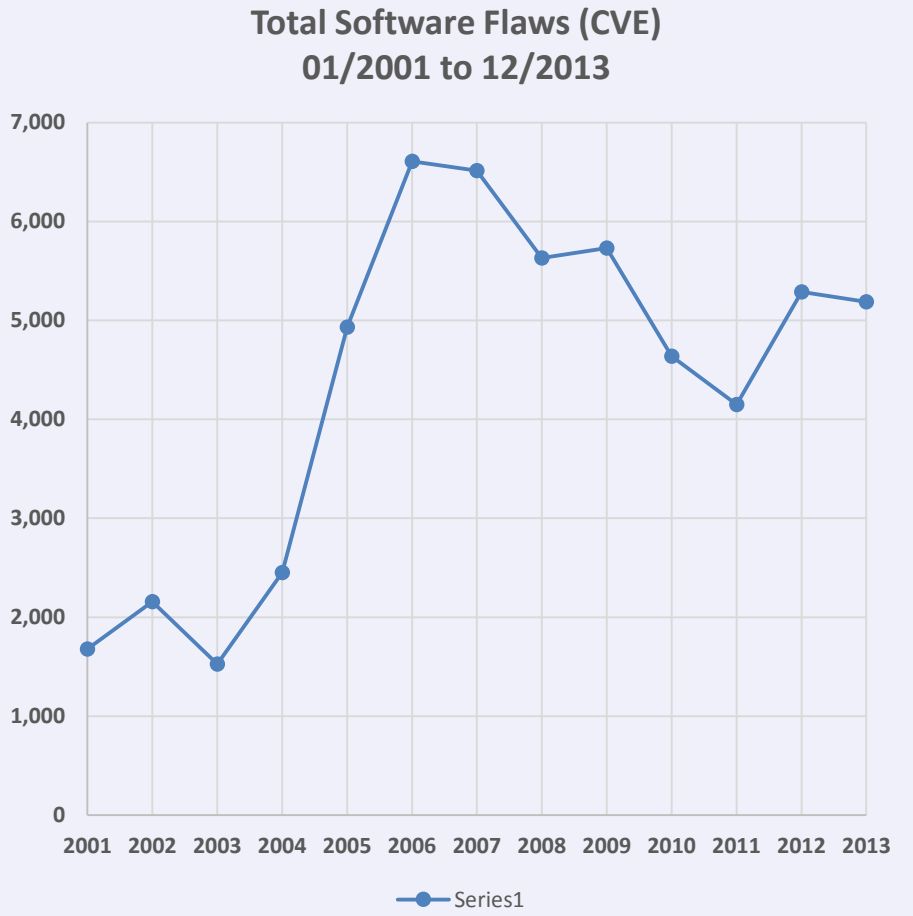
Python Security is a free, open source, OWASP Project that aims at creating a hardened version of python that makes it easier for security professionals and developers to write applications more resilient to attacks and manipulations.

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



http://web.nvd.nist.gov/view/vuln/statistics

After checking statistics generated from vendors we have to also check data generated by the community at large.

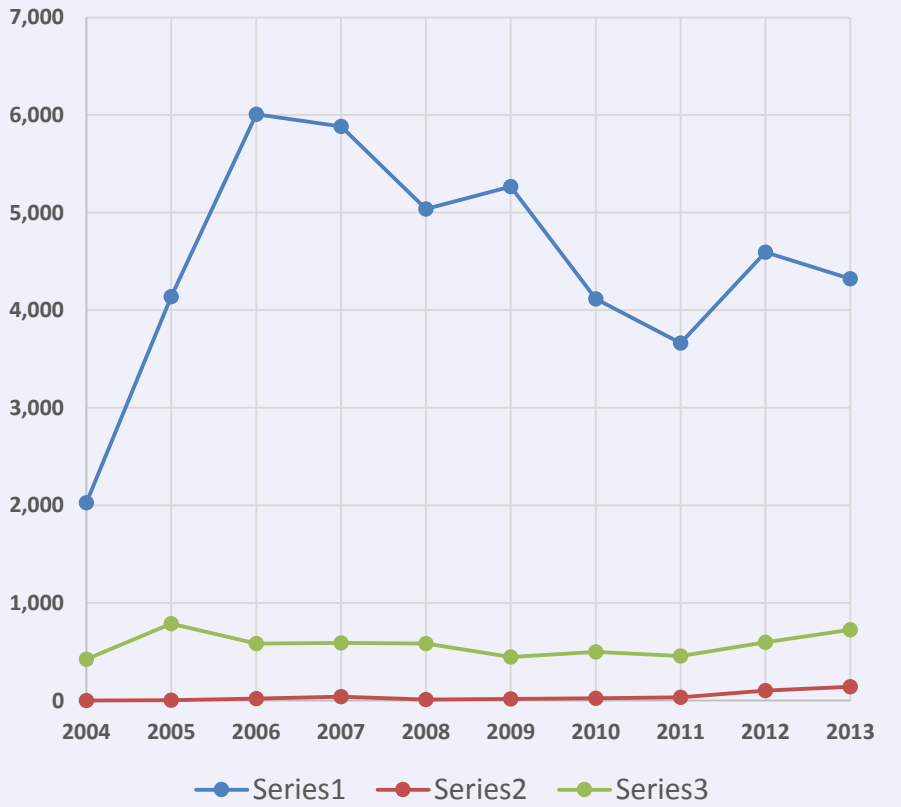
Statistics on publicly disclosed vulnerabilities are available at the site "NIST.gov" under the name "**National Vulnerability Database"**

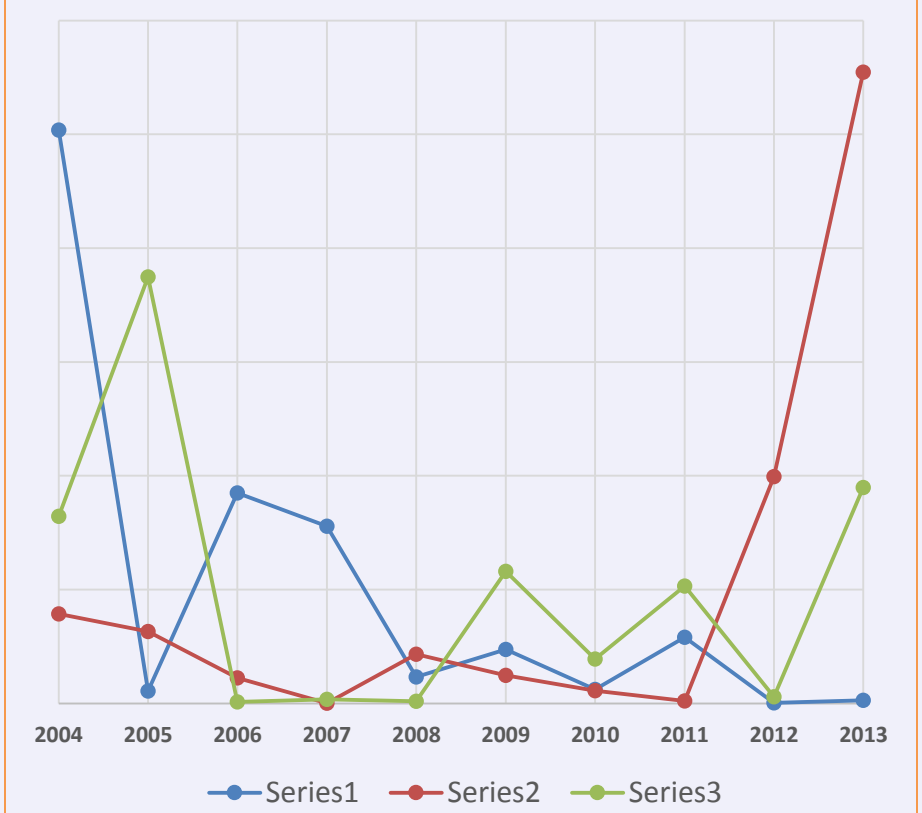
http://web.nvd.nist.gov/view/vuln/statistics

We will review vulnerability stats:

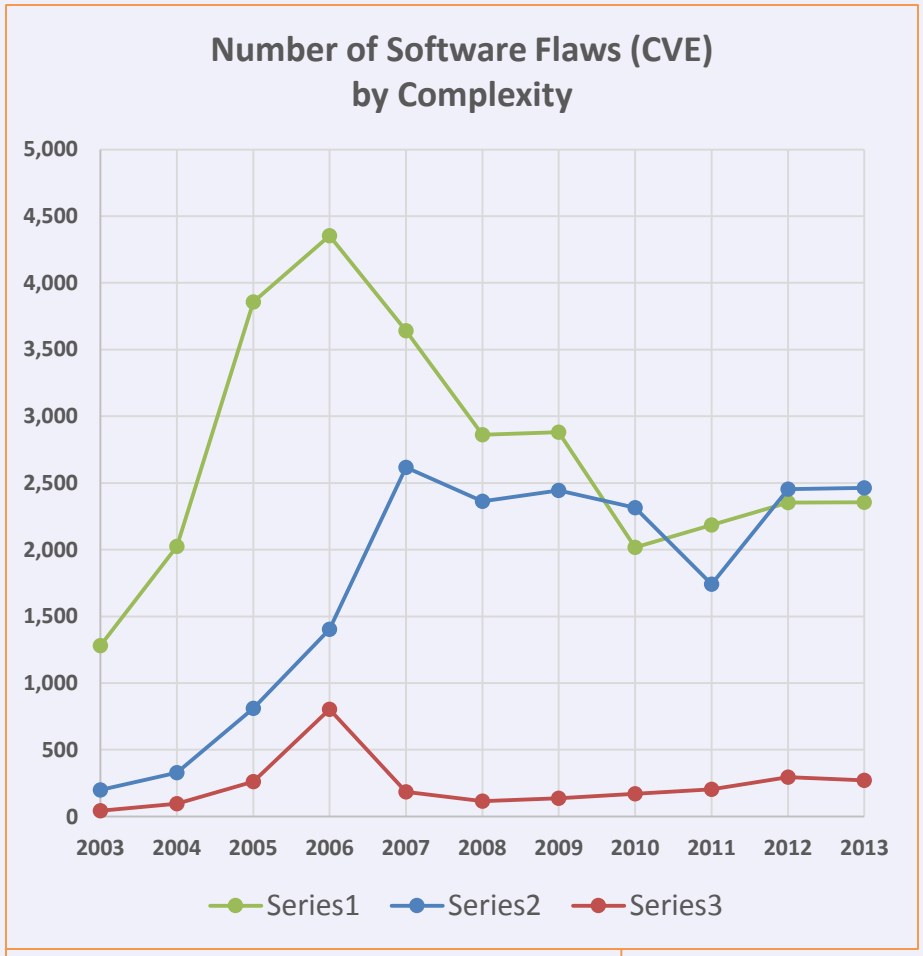
- By Access vector
- By Complexity
- By Severity
- By Category

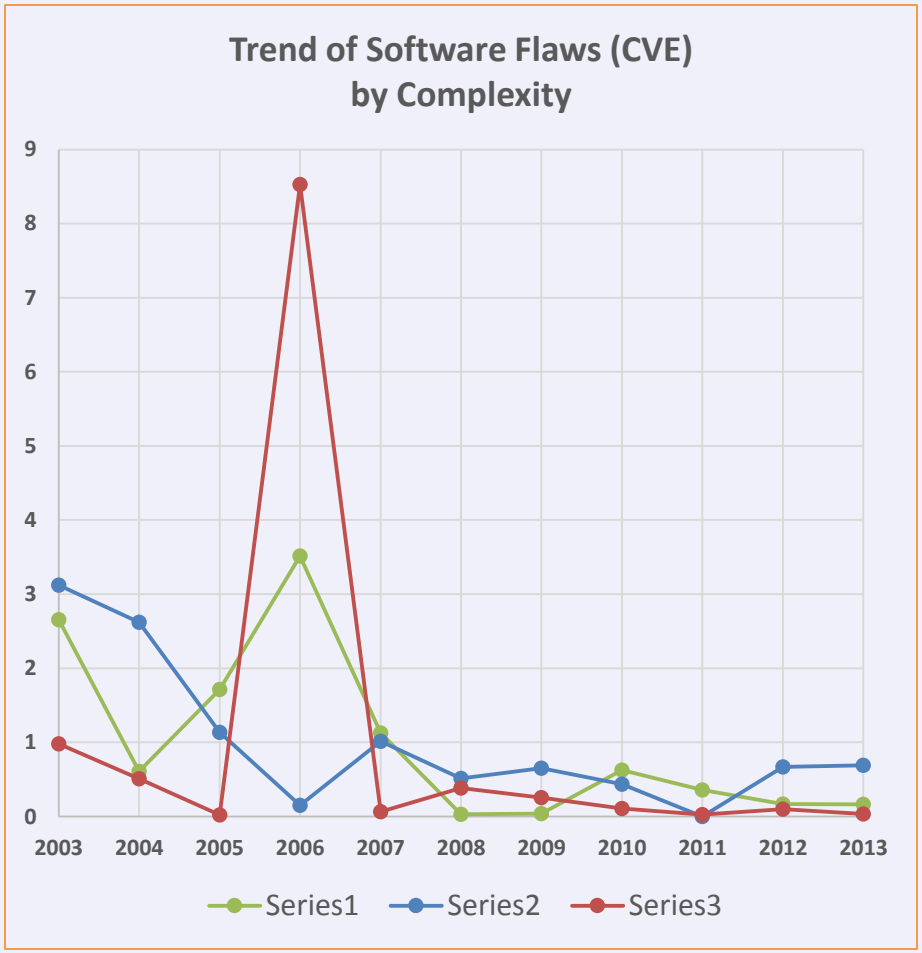
Then we will formulate some conclusions.





http://web.nvd.nist.gov/view/vuln/statistics





http://web.nvd.nist.gov/view/vuln/statistics

- Initial review of "**National Vulnerability Database**" statistics revealed:
  - Number of public vulnerabilities relaying on "network" is decreasing
  - Number of public vulnerabilities relaying on "local network" access (adjacent networks) in increasing
  - Number of public vulnerabilities relaying on "local access only" access in increasing
  - Medium or low complexity Vulnerabilities are preferred

- Analysis of the "**Web Application Vulnerability Statistics 2013**" report revealed:
  - Rate of server misconfigurations is increasing
  - Authentication issues are increasingly not checked
  - Authorization issues are increasingly not checked
  - Server application headers are not sanitized
  - Server application error are not filtered
  - Server default files/dirs are left accessible

- How network configurations can impact internal code operations?

- IP Fragmentation
  - https://isc.sans.edu/forums/diary/IP+Fragmentation+Attacks/13282
  - http://www.snort.org/assets/165/target_based_frag.pdf
  - http://www.icir.org/vern/papers/activemap-oak03.pdf

- Depending on the system reading the fragmented packets arriving at the NIC, the reassembly process can either DESTROY or REASSEMBLE the original stream, as an application may have sent valid data but the receiving end may see only random data.

```
def genjudyfrags():
    pkts=scapy.plist.PacketList()
    pkts.append(IP(flags="MF",frag=0)/("1"*24))
    pkts.append(IP(flags="MF",frag=4)/("2"*16))
    pkts.append(IP(flags="MF",frag=6)/("3"*24))
    pkts.append(IP(flags="MF",frag=1)/("4"*32))
    pkts.append(IP(flags="MF",frag=6)/("5"*24))
    pkts.append(IP(frag=9)/("6"*24))
return pkts
```

This section of code will generate six packet fragments as outlined in "IP Fragment Reassembly with scapy" with the offsets specified in the Shankar/Paxson and Novak papers.

The picture is taken from the Novak paper and represent the final packet order per each reassembly policy.

http://www.snort.org/assets/165/target_based_frag.pdf

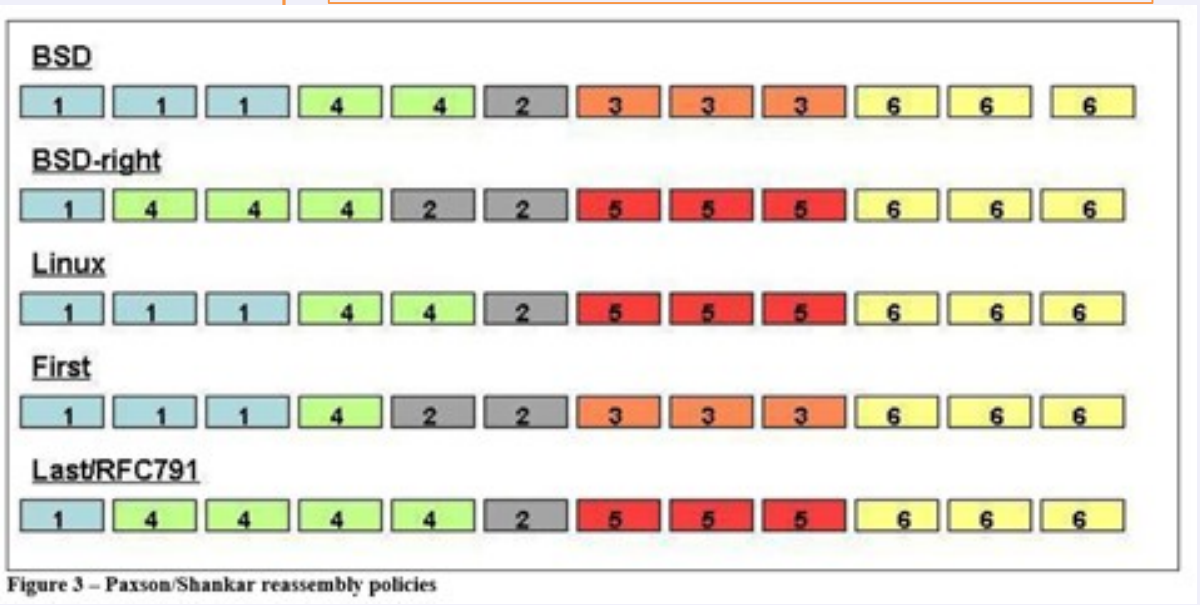


Figure 3 – Paxson/Shankar reassembly policies

python -OOBR reassembler.py –demo

Reassembled using policy: First (Windows, SUN, MacOS, HPUX)

1111111111111111111111144444442222222222222223333333333333333333333366666666666666666666

Reassembled using policy: Last/RFC791 (Cisco)

1111111444444444444444444444444444442222222555555555555555555555566666666666666666666

Reassembled using policy: Linux (Umm.. Linux)

1111111111111111111111144444444444444422222225555555555555555555555566666666666666666666

Reassembled using policy: BSD (AIX, FreeBSD, HPUX, VMS)

1111111111111111111111144444444444444422222223333333333333333333333366666666666666666666

Reassembled using policy: BSD-Right (HP Jet Direct)

1111111444444444444444444444422222222222222255555555555555555555556666666666666666666

- What about numeric operations?

As an example we will take in consideration LINUX.

Many security operations are based on random numbers and every linux system using any cryptographic function can be impacted by the lack of good entropy.

What is generally overlooked is that under linux almost every process uses entropy when is created and even the network stack uses entropy to generate the "*TCP-syn cookies*".

- This is an expected behavior and is working as designed.
- Spawning a process uses (on average) 16 bytes of entropy per "*exec()*", therefore when server load spikes entropy is quickly depleted as the kernel is not generating entropy fast enough.
- Also when a system is built to use "*Stack Smashing Protector*" (SSP) by default it uses "*/dev/urandom*" directly, this tends to consume all the kernel entropy.
- Almost all modern Linux systems use "*Address space layout randomization*" (ASLR) and stack protections that need a small amount of entropy per process. Since "/dev/*urandom*" always remixes, it doesn't strictly run out, but the entropy drops.

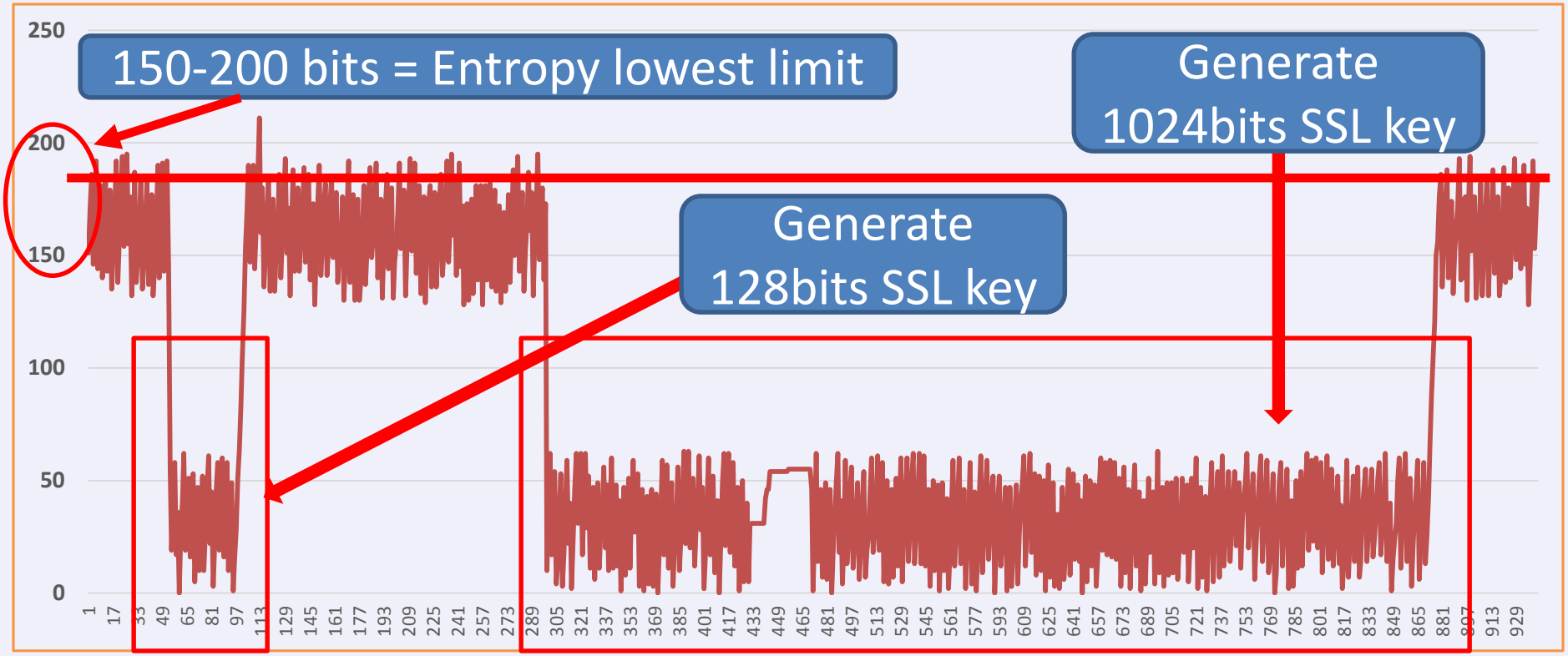
- In fact many linux command used to check the amount of entropy are “consuming” it and may lead to it’s depletion.

- For example this command will “consume” entropy
  - *watch cat /proc/sys/kernel/random/entropy_avail*

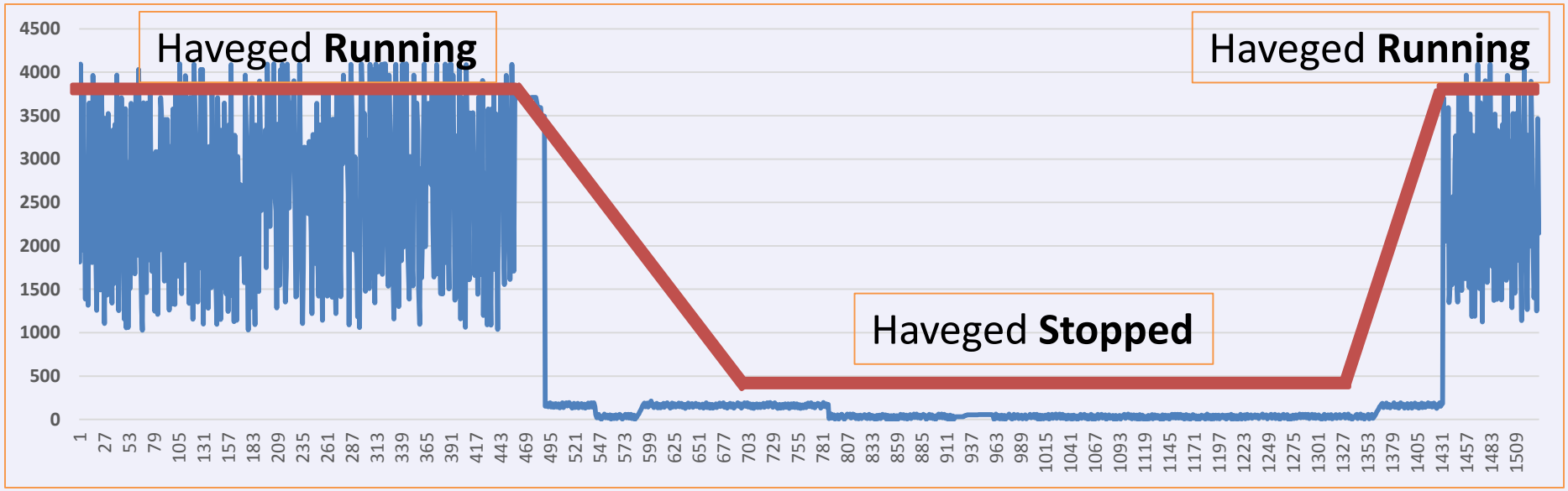
- But this python one-line script will NOT use entropy:
  - *python -c "$(echo -e "import time\nwhile True:\n time.sleep(0.5)\n print open('/proc/sys/kernel/random/entropy_avail', 'rb').read(),")"*

- Also the command “*inotifywatch -v -t 60 /dev/random*” will monitor the access to “/dev/random” without using entropy

- What happens to the entropy level in a working linux server under average load?



Under linux every process uses entropy and every server “should” not have less than 200 bits. It Is possible to increase the entropy level using entropy deamons like the package “haveged”. (http://www.issihosts.com/haveged/)



- **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 Secure Coding Principles

1. Minimize attack surface area
2. Establish secure defaults
3. Principle of Least privilege
4. Principle of Defence in depth
5. Fail securely

6. Don’t trust services
7. Separation of duties
8. Avoid security by obscurity
9. Keep security simple
10. Fix security issues correctly

- In reality “Secure coding” is a **PRACTICE**

Practice: “the actual application or use of an idea, belief, or method, as opposed to theories relating to it”

The definition of “secure coding” **changes over time** as each person/company has different ideas.

- **Is about how to DESIGN code to be inherently secure and NOT on how to write secure code**

- As a PRACTICE secure coding includes but is not limited to:

  - Definition of areas of interest
  - Analysis of architectures involved
  - Review of implementation details
  - Verification of code logic and syntax
  - Operational testing (unit testing, white-box)
  - Functional testing (black-box)

- Secure coding depends on “functional testing”
  - Functional testing: “verifies a program by checking it against ... design document(s) or specification(s)”
  - System testing: “validate[s] a program by checking it against the published user or system requirements”

(Kaner, Falk, Nguyen. *Testing Computer Software*. Wiley Computer Publishing, 1999)

- Operational testing = white-box testing → unit-test
  - (http://en.wikipedia.org/wiki/Operational_acceptance_testing)
- Functional testing = black-box testing
  - (http://en.wikipedia.org/wiki/Functional_testing)

- **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?

- EXAMPLE – numeric overflow

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

RESULT (debian 7 x64)

```
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

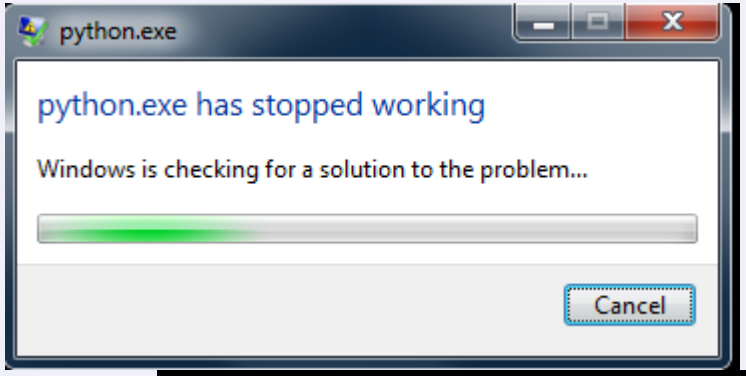
close failed in file object destructor:
sys.excepthook is missing
lost sys.stderr

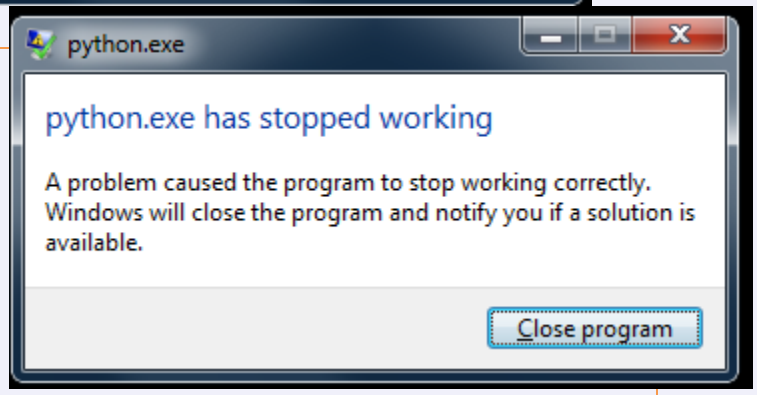
Code is trying to write a non-zero amount of data to something that does not exists.
The file descriptor has been closed and nothing can be sent, but python has no control over it and returns a system error.

- EXAMPLE - File descriptors in Windows

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

python.exe -OOBtt “winfd_1.py”



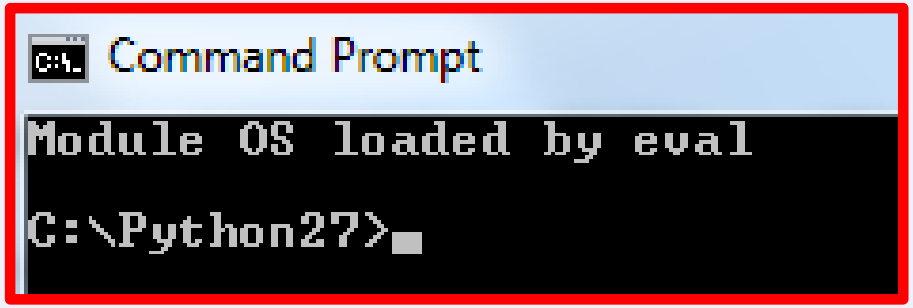



```
import io
import sys

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

sys.stdout.write(“Now writing to stdout closed FD will cause a crash")
```

- EXAMPLE – string evaluation

```
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)
```

```
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>
```



The function "eval" executes a string but is not possible to any control to the operation. Malicious code is executed without limits in the context of the user that loaded the interpreter.

REALLY DANGEROUS

- EXAMPLE – input evaluation

```
Secret = "A SECRET DATA"
Public = "a COCONUT"
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 COCONUT

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

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

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

- Unicode string encode/decode

RESULT

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 and wait to receive the remaining 3 bytes, and 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:
    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()))
```

← **ISSUE HERE**

- 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'>

....

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'>

....

File **"/usr/lib/python2.7/pickle.py**", line 488, in save_string self.write(STRING + repr(obj) + '\n')

MemoryError

- 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
```

WARNING: pickle or cPickle are NOT designed as safe/secure solution for serialization

- 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 – 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.

- 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 – 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
```

- EXAMPLE – libs with “unlimited data“ issues

HTTPLIB → http://bugs.python.org/issue16037 (fixed)

FTPLIB → http://bugs.python.org/issue16038 (fixed)

IMAPLIB → http://bugs.python.org/issue16039 (fixed)

NNTPLIB → http://bugs.python.org/issue16040 (fixed)

POPLIB → http://bugs.python.org/issue16041

SMTPLIB → http://bugs.python.org/issue16042

XMLRPC → http://bugs.python.org/issue16043

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

ast
bastion
commands
cookie
cPickle
eval
marshal
mktemp

multiprocessing
os.exec
os.popen
os.spawn
os.system
parser
pickle
pipes

pty
rexec
shelve
subprocess
tarfile
yaml
zipfile

- **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/

- **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*

- **Closing Summary**

- Python is a *powerful* and *easy to learn* language **BUT** has to be used with care.

- There are *no limits* or controls in the language, is *responsibility of the coder to know what can be done and what to avoid.*

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