Sandbox for Python

网络技术组
黄俊文
huangjumwen@corp.netease.com

Keywords

- Sandbox
- Virtualization
- isolation

Agenda

- Goal (igor)
- A survey of possible solutions
- Current solution
- Other's solution
- Better solution :)
- Q&A&Discuss

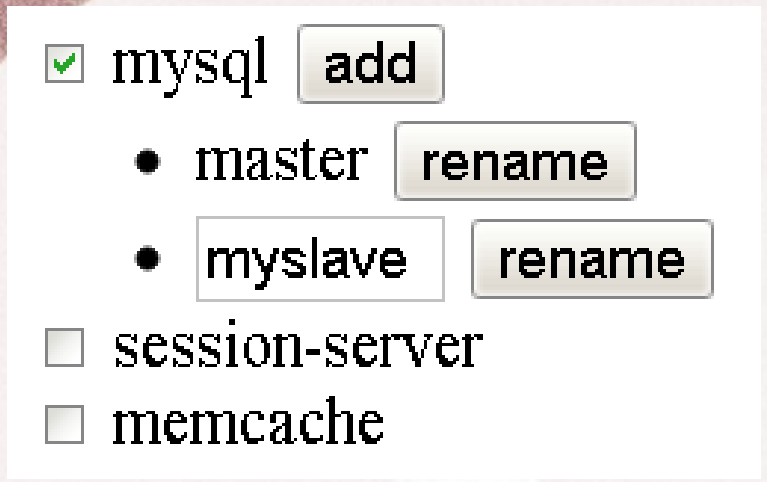
Goal

- 给在 igor 上托管的项目提供受限的环境
  - 防止用户代码的越权行为
    - reboot
    - 修改其他用户的代码
    - 访问到内部接口

Goal

- 给在 igor 上托管的项目提供受限的环境
  - 提供可用的服务
    - Mysql/memcache/session/fetchURL..
    - 通行证接口 ,pay 接口 ...
    - Igor 本身的接口？
  - 用户代码只需要饭来伸手即可

Goal




```
from igor.api import mysql

conn = mysql.get_conn("myslave")
conn.execute("select * from mytable")
```

Goal

- 给在 igor 上托管的项目提供受限的环境
  - 限制资源的使用
    - cpu
    - memory
    - io
    - fd
    - ...

Goal

- 给在 igor 上托管的项目提供受限的环境
  - Lightweight (数十上百 ?)
  - Easy to deploy
  - Low performance penalty
  - Maintainable

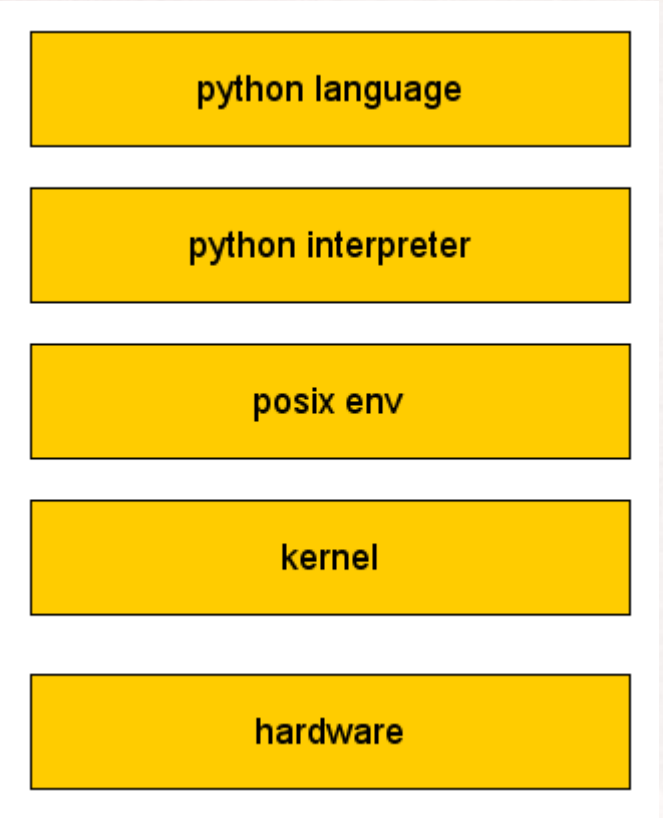
Goal

- 给在 igor 上托管的项目提供受限的环境
  - Pure python guest code
  - Single thread
  - Embed in a standalone server

A survey of possible solutions

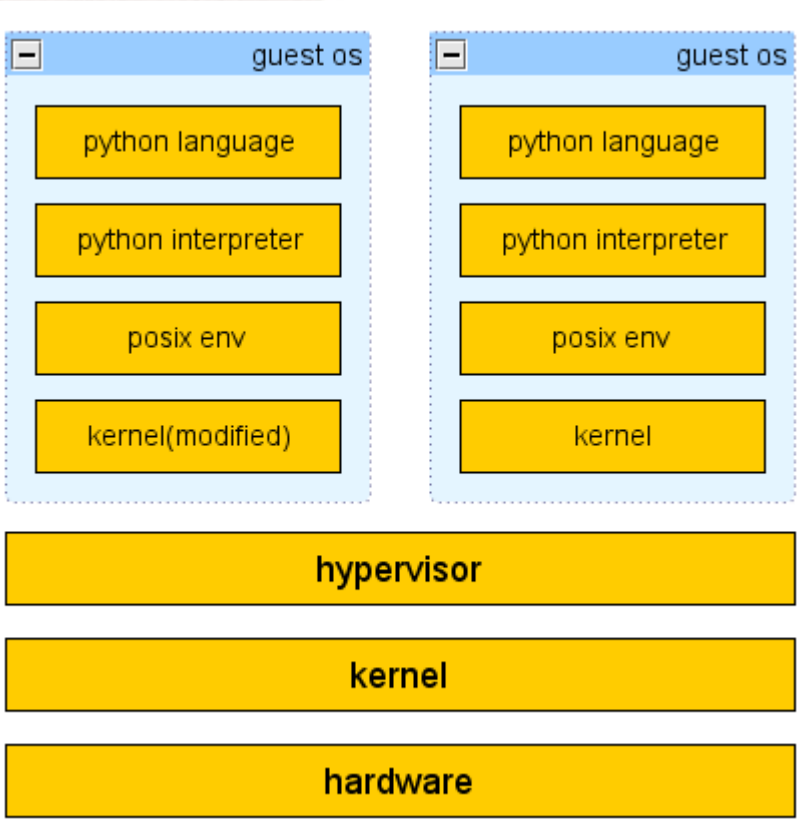
- Google app engine
  http://code.google.com/appengine/
  - Virtualization can implement sandbox
  - Sandbox != virtualization
- Heroku
  http://www.heroku.com/

A survey of possible solutions



A survey of possible solutions

- Full/para virtualization/emulation
  - Vmware
  - Xen
  - Qemu

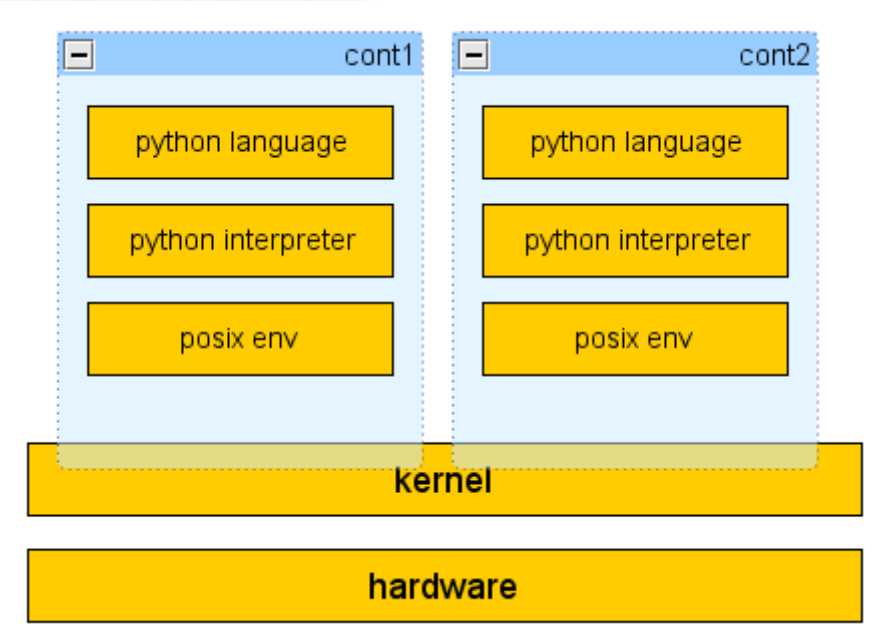


A survey of possible solutions

- Full/para virtualization/emulation
  - Pros
    - Stable enough
    - Should be easy enough to deploy
  - Cons
    - Heavyweight
    - High overhead

A survey of possible solutions

- Container virtualization (os-level virtualization)
  - Chroot?
  - Linux-vserver
  - Openvz
  - **Lxc**



A survey of possible solutions

- Container virtualization (os-level virtualization)
  - Pros
    - Low overhead
    - lightweight(lxc)
  - Cons
    - May need patch to kernel
    - May need higher kernel version(it is really not a problem)
    - Coarse-grained?

A survey of possible solutions

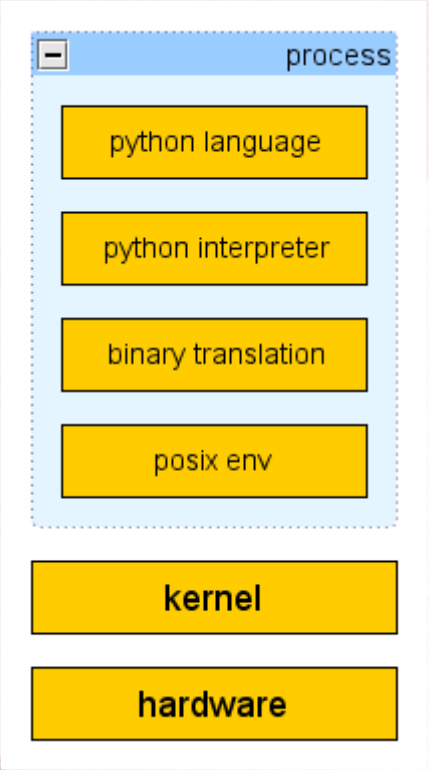
- System call virtualization
  - Intercepting syscalls and emulate a 'view' for the process
    - Ptrace
    - Binary translation
    - LD_PRELOAD
    - Modified glibc

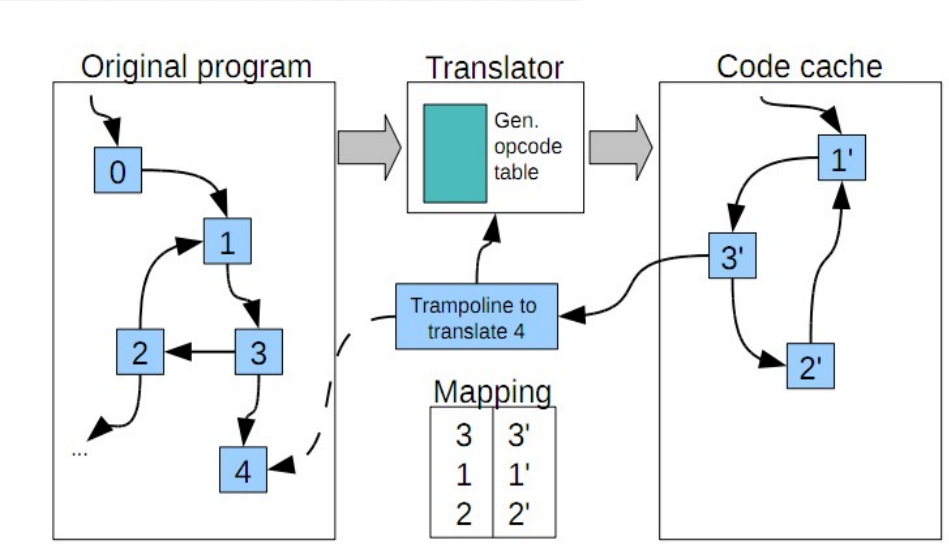
A survey of possible solutions

- System call virtualization
  - Plash
  - Google's seccomp-sandbox
  - Libdetox
  - secomp-nurse

A survey of possible solutions

- FastBT(libdetox)



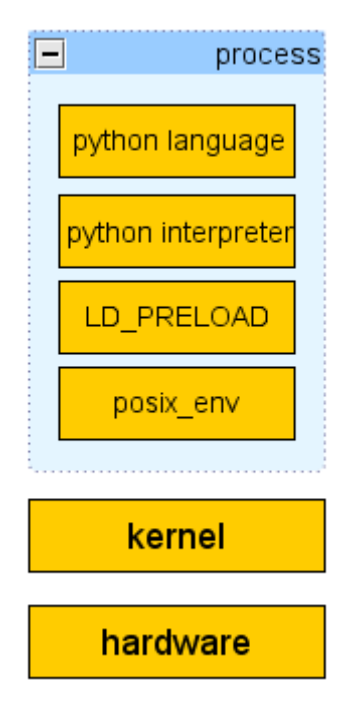


A survey of possible solutions

- FastBT(libdetox)
  - Pros
    - Lightweight
    - Easy to use and understand
  - Cons
    - High overhead
    - Too thin

A survey of possible solutions

- purelibc



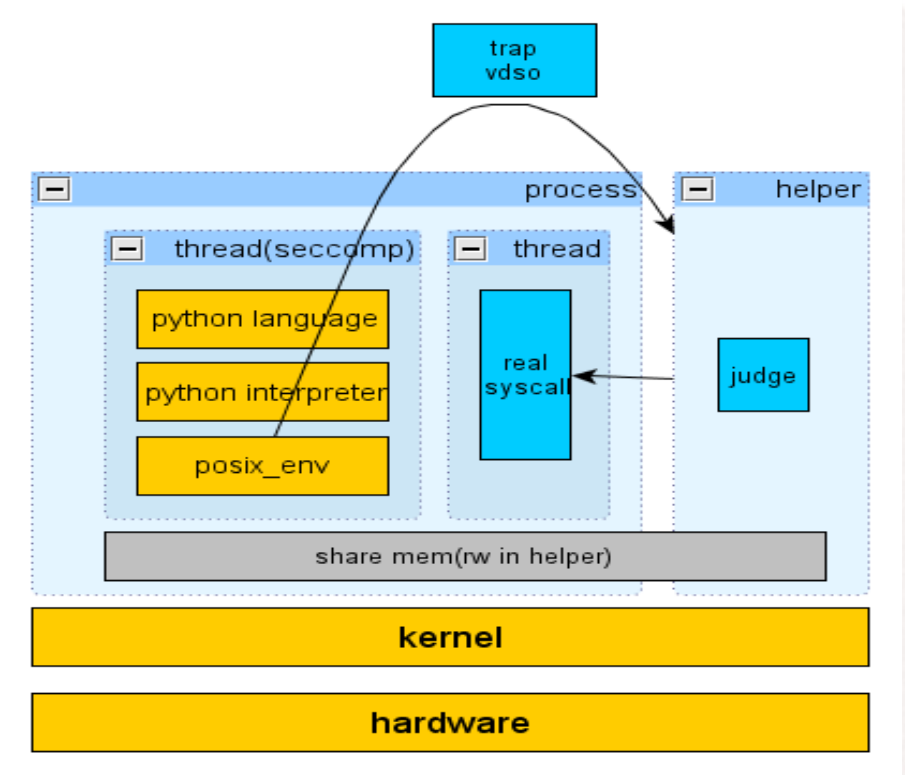

```
#define _GNU_SOURCE
#include <stdio.h>
#include <string.h>
#include <stdarg.h>
#include <sys/syscall.h>
#include <unistd.h>
#include <stdlib.h>
#include <purelibc.h>
#include <dlfcn.h>

static sfun _native_syscall;
static char hosts[]="/etc/hosts";

static char buf[128];
static long int mysc(long int sysno, ...){
  va_list ap;
  long int a1,a2,a3,a4,a5,a6;
  va_start (ap, sysno);
  a1=va_arg(ap,long int);
  a2=va_arg(ap,long int);
  a3=va_arg(ap,long int);
  a4=va_arg(ap,long int);
  a5=va_arg(ap,long int);
  a6=va_arg(ap,long int);
  va_end(ap);
  if (sysno == __NR_open) {
    char *path=(char *)a1;
    if (a1 && strcmp(path,"/etc/passwd")==0)
      a1=(long int) hosts;
  }
  return _native_syscall(sysno,a1,a2,a3,a4,a5,a6);
}

void
__attribute ((constructor))
init_test (void)
{
  _native_syscall=_pure_start(mysc,NULL,PUREFLAG_STDALL);
}
```

A survey of possible solutions

- purelibc
  - Pros
    - Very low overhead
    - Lightweight
  - Cons
    - Not all syscalls are covered by purelibc
    - Not very secure

A survey of possible solutions

- Seccomp-nurse



A survey of possible solutions

- Seccomp-nurse
  - Pros
    - ...
  - Cons
    - Slow
    - Too complex

A survey of possible solutions

- Python language level sandbox
  - Rexec (deprecated)
  - Object-capability
    - Ref cannot be forged
    - Immutable shared state
    - Private namespace
  - Object-capabilities' security based on the controlling of references to objects

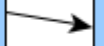
A survey of possible solutions

- Pypy
  - Pros
    - No os support needed
    - Faster than cpython
  - Cons
    - Two processes
    - Long time to compile
    - No libpypy.so ?
    - C extension for cpython

A survey of possible solutions

- RestirctedPython
  - Binary translation in python

```
def f(x):
    x.foo = x.foo + x[0]
    print x
    return printed
```



```
def f(x):
    # Make local variables from globals.
    _print = _print_()
    _write = _write_
    _getattr = _getattr_
    _getitem = _getitem_

    # Translation of f(x) above
    _write(x).foo = _getattr(x, 'foo') + _getitem(x, 0)
    print >>_print, x
    return _print()
```

A survey of possible solutions

- RestirctedPython
  - Pros
    - Very easy to use
    - Normal cpython
    - Fine grained control
  - Cons
    - High overhead

Current solution

- Patch python c source
- Two modes: normal vs jail
- Python 2.5.2: as in GAE
- Mysql/session server/memcache/urllib currently

Current solution

```
 1 import _gateway
 2 import memcache
 3 import sys, datetime
 4
 5 mcuri = _gateway.gatewayuri("mc", "memcache:tcp://0.0.0.0:11211")
 6
 7 sys.enter_jail()
 8
 9 def test1():
10     # should be ok
11     mc = memcache.Client([mcuri,])
12     mc.set("a", datetime.datetime.now().strftime("%F %X"))
13     print "test1: ", mc.get("a")
14
15 def test2():
16     # should failed
17     mc = memcache.Client(["0.0.0.0:11211",])
18     mc.set("a", datetime.datetime.now().strftime("%F %X"))
19     print "test2: ", mc.get("a")
20
21 test1()
22 test2()
```

Current solution

- Sys.enter_jail: enter jail mode
- Fixmods: a chance to change c modules' content
- _gateway: opaque token
- PY_JAIL_CFUNCTION: macro to deny a PyCFunction in jail mode

Current solution

- In jail, you can't:
  - Write files
  - Read dso (binary files)
  - Signal module → only handle keyboard interrupt
  - Posix module → only a few functions are allowed
  - _socket → can create a socket object by gateway uri
  - ……
  - Ref: [Python252_patch_detal] and [Third_party_modules_patch] on wiki

Current solution

- Chroot env:
  - Virtual env
  - Partialfs
- Embedded server

Current solution

- Pros
  - Low overhead
  - Mostly a normal cpython
  - Fine grained control
- Cons
  - High maintain cost
  - Not proven

Others' solution

- ItownSDK
- GAE

Better solution

- Modified python + linux container ?
  - Resource isolated by container:
    - Cpu/mem/eth/io/freezer...
  - Finer grain control by modified python
    - Don't interference with the server
    - Information hidden: mysql's user/passwd or private key, etc.

references

plash's wiki:
http://plash.beasts.org/wiki/HowPlashWorks

intercepting syscalls:
http://plash.beasts.org/wiki/InterceptingSystemCalls

google native client:
http://www.chromium.org/nativeclient/reference/research-papers

http://src.chromium.org/viewvc/native_client/data/docs_tarball/nacl/googleclient/native_client/documentation/nacl_paper.pdf

fastBT:
http://nebelwelt.net/fileadmin/downloads/ETH/Conferences/AMAS-BT09/fastbt_payerm-paper.pdf

python:
http://wiki.python.org/moin/SandboxedPython
http://plash.beasts.org/wiki/CapPython
http://www.cs.ubc.ca/~drifty/papers/python_security.pdf

lxc:
http://lxc.sourceforge.net/
http://lxc.sourceforge.net/index.php/about/kernel-namespaces/
http://www.ibm.com/developerworks/linux/library/l-lxc-containers/
http://www.ibm.com/developerworks/linux/library/l-lxc-security/index.html
Igor's wiki:
[Python-2.5.2 patch detail]
[Third_party_modules_patch]