PyCon Israel 2018

Escaping the Python Sandbox

PRESENTED BY:

Tomer Zait

Security Researcher, F5 Networks

$ whoami

- Security Researcher at F5 Networks
- Practical Software Engineer, OSCP, OSCE
- 4 Times Winner Of The Israeli Cyber Challenge (CTF)
- Open Source Security Projects: x64dbgpy, PyMultitor, phantom-requests, SubDomain-Analyzer, AutoBrowser

- Twitter: @realgam3
- Linkedin: https://linkedin.com/in/realgam3
- Github: https://github.com/realgam3

PySandbox, Is It Possible?

Lets Ask Google About PySandbox?

Lets Ask Google About PySandbox?

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!! WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING !!!
!!!                                                                 !!!
!!! pysandbox is BROKEN BY DESIGN, please move to a new sandboxing  !!!
!!! solution: run python in a sandbox, not the opposite!            !!!
!!!                                                                 !!!
!!! Learn more about pysandbox failure:                             !!!
!!! https://lwn.net/Articles/574215/                                !!!
!!!                                                                 !!!
!!! WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING !!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

About Python

Objects In Python

Objects In Python

```
In [1]: import os

In [2]: isinstance(os, object)
Out[2]: True
```

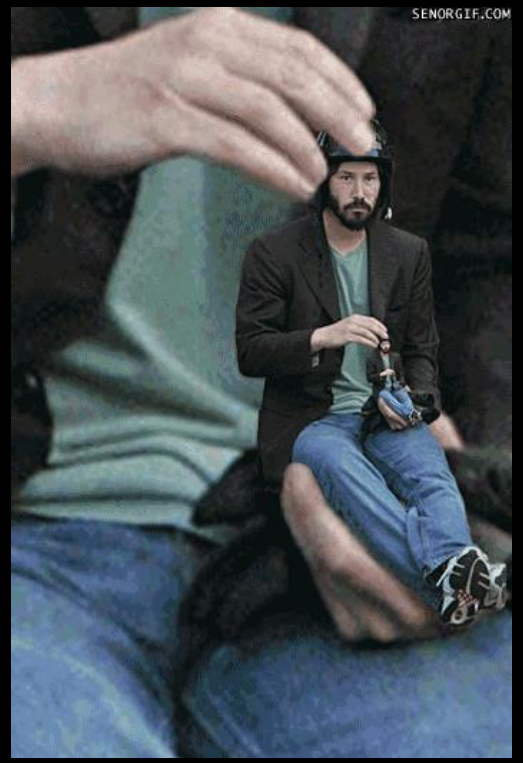
Objects In Python

```
In [1]: def func():
   ...:     pass
   ...:

In [2]: isinstance(func, object)
Out[2]: True
```

Objects In Python

```
In [1]: isinstance(object, object)
Out[1]: True
```



~~Objects~~ In Python Types

```
In [1]: isinstance(object, type)
Out[1]: True

In [2]: type(object)
Out[2]: type
```

Special Attributes

5.13. Special Attributes

The implementation adds a few special read-only attributes to several object types, where they are relevant. Some of these are not reported by the `dir()` built-in function.

`object.`**`__dict__`**
A dictionary or other mapping object used to store an object's (writable) attributes.

`instance.`**`__class__`**
The class to which a class instance belongs.

`class.`**`__bases__`**
The tuple of base classes of a class object.

`definition.`**`__name__`**
The name of the class, type, function, method, descriptor, or generator instance.

The following attributes are only supported by new-style classes.

`class.`**`__mro__`**
This attribute is a tuple of classes that are considered when looking for base classes during method resolution.

`class.`**`mro`**()
This method can be overridden by a metaclass to customize the method resolution order for its instances. It is called at class instantiation, and its result is stored in `__mro__`.

`class.`**`__subclasses__`**()
Each new-style class keeps a list of weak references to its immediate subclasses. This method returns a list of all those references still alive. Example:

Subclasses

```
In [1]: class C1(object): pass

In [2]: class C2(C1): pass

In [3]: class C3(C1): pass

In [4]: C1.__subclasses__()
Out[4]: [__main__.C2, __main__.C3]
```

Method Resolution Order

```
In [1]: class A(object): pass

In [2]: class B(object): pass

In [3]: class AB(A,B): pass

In [4]: class BA(B,A): pass

In [5]: AB.__mro__
Out[5]: (__main__.AB, __main__.A, __main__.B, object)

In [6]: BA.__mro__
Out[6]: (__main__.BA, __main__.B, __main__.A, object)
```

Method Resolution Order

```
In [7]: class ABBA(AB,BA): pass
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-29f9253959b0> in <module>()
----> 1 class ABBA(AB,BA): pass

TypeError: Error when calling the metaclass bases
    Cannot create a consistent method resolution
order (MRO) for bases B, A
```

Built-in

```
In [1]: __builtin__
Out[1]: <module '__builtin__' (built-in)>

In [2]: __builtin__.
```

__builtin__.abs	__builtin__.chr	__builtin__.EOFError
__builtin__.all	__builtin__.classmethod	__builtin__.eval
__builtin__.any	__builtin__.cmp	__builtin__.Exception
__builtin__.apply	__builtin__.coerce	__builtin__.execfile
__builtin__.ArithmeticError	__builtin__.compile	__builtin__.exit
__builtin__.AssertionError	__builtin__.complex	__builtin__.False
__builtin__.AttributeError	__builtin__.copyright	__builtin__.file
__builtin__.BaseException	__builtin__.credits	__builtin__.filter
__builtin__.basestring	__builtin__.delattr	__builtin__.float
__builtin__.bin	__builtin__.DeprecationWarning	__builtin__.FloatingPointError
__builtin__.bool	__builtin__.dict	__builtin__.format
__builtin__.buffer	__builtin__.dir	__builtin__.frozenset
__builtin__.BufferError	__builtin__.display	__builtin__.FutureWarning
__builtin__.bytearray	__builtin__.divmod	__builtin__.GeneratorExit
__builtin__.bytes	__builtin__.Ellipsis	__builtin__.getattr
__builtin__.BytesWarning	__builtin__.enumerate	__builtin__.globals
__builtin__.callable	__builtin__.EnvironmentError	__builtin__.hasattr

Function Attributes

User-defined functions

A user-defined function object is created by a function definition (see section Function definitions). It should be called with an argument list containing the same number of items as the function's formal parameter list.

Special attributes:

Attribute	Meaning	
`__doc__` `func_doc`	The function's documentation string, or `None` if unavailable.	Writable
`__name__` `func_name`	The function's name	Writable
`__module__`	The name of the module the function was defined in, or `None` if unavailable.	Writable
`__defaults__` `func_defaults`	A tuple containing default argument values for those arguments that have defaults, or `None` if no arguments have a default value.	Writable
`__code__` `func_code`	The code object representing the compiled function body.	Writable
`__globals__` `func_globals`	A reference to the dictionary that holds the function's global variables — the global namespace of the module in which the function was defined.	Read-only
`__dict__` `func_dict`	The namespace supporting arbitrary function attributes.	Writable
`__closure__` `func_closure`	`None` or a tuple of cells that contain bindings for the function's free variables.	Read-only

The Challenges

Level #1 - Secure Pyshell

Secure Pyshell - 100 Pwning - Solved

Solve Hint Review

A friend of mine is die hard fan of python . He created a python interpreter of his own And claims to be very secure , prove him he is wrong. He loves Trump, btw.

nc 139.59.61.220 22345

Submit!

Level #2 - Zumbo 3

Zumbo 3 250 ×

And the final stage, with real hacking included!

Welcome to ZUMBOCOM....you can do anything at ZUMBOCOM.

Three flags await. Can you find them?

http://zumbo-8ac445b1.ctf.bsidessf.net

Solved

Solved By:

hanto	7 days ago
israelites	11 days ago
anaconda	12 days ago
nocommnt	12 days ago

Show 105 others

Level #3 - PyBabbies

pybabbies

200

443 solves

so secure it hurts

nc 54.165.210.171 12345

Written by ColdHeat

pyshell.py

Key

Submit

Level #3 - PyBabbies

```
from __future__ import print_function

targets = __builtins__.__dict__.keys()
targets.remove('raw_input')
targets.remove('print')
for x in targets:
    del __builtins__.__dict__[x]
```

Level #3 - PyBabbies

```
banned = [
    "import",
    "exec",
    "eval",
    "pickle",
    "os",
    "subprocess",
    "kevin sucks",
    "input",
    "banned",
    "cry sum more",
    "sys"
]
```

https://Links

- http://pyconil2018.realgame.co.il
- https://www.digitalwhisper.co.il/files/Zines/0x5A/DW90-5-PySandbox.pdf
- https://github.com/vstinner/pysandbox
- https://nvisium.com/blog/2016/03/09/exploring-ssti-in-flask-jinja2.html

If You Really Like CTF Challenges

Questions?

SOLUTIONS FOR AN APPLICATION WORLD