Prison Snake

A broad dive into the object structure of Python and the functionality behind PyJail solutions

Charles Averill

Computer Security Group
The University of Texas at Dallas

February 2022

What is a PyJail?

- A common CTF problem in which a Python interpreter with limited functionality is provided to the user

What is a PyJail?

- A common CTF problem in which a Python interpreter with limited functionality is provided to the user
- Goal is typically to call `os.system()`, `open()`, or another similar function that provides access to file-reading abilities

What is a PyJail?

- A common CTF problem in which a Python interpreter with limited functionality is provided to the user
- Goal is typically to call `os.system()`, `open()`, or another similar function that provides access to file-reading abilities
- Common restrictions involve removing keywords such as `import`, blocking any input containing the text `open`, preventing any function calls outside of `print`, etc.

What is a PyJail?

- A common CTF problem in which a Python interpreter with limited functionality is provided to the user
- Goal is typically to call `os.system()`, `open()`, or another similar function that provides access to file-reading abilities
- Common restrictions involve removing keywords such as `import`, blocking any input containing the text `open`, preventing any function calls outside of `print`, etc.
- As a result of these limitations, solutions usually look something like

```
().__class__.__base__.__subclasses__()[134].__init__.__globals__["__builtins__"]["open"]("./key", "r").read()
```

What is a PyJail?

- A common CTF problem in which a Python interpreter with limited functionality is provided to the user
- Goal is typically to call `os.system()`, `open()`, or another similar function that provides access to file-reading abilities
- Common restrictions involve removing keywords such as `import`, blocking any input containing the text `open`, preventing any function calls outside of `print`, etc.
- As a result of these limitations, solutions usually look something like

```
().__class__.__base__.__subclasses__()[134].__init__.__globals__["__builtins__"]["open"]("./key", "r").read()
```

- This is crazy! The solution seems arbitrary but there is logic behind each attribute chosen in this solution. We will discover why solutions like this work, and why this functionality exists.

What is CPython?

- The official reference implementation of the Python programming language

What is CPython?

- The official reference implementation of the Python programming language
- Written in a combo of C and Python

What is CPython?

- The official reference implementation of the Python programming language
- Written in a combo of C and Python
- Compiles Python to bytecode (.pyc files) to be interpreted later, so technically a compiler and interpreter

What is CPython?

- The official reference implementation of the Python programming language
- Written in a combo of C and Python
- Compiles Python to bytecode (.pyc files) to be interpreted later, so technically a compiler and interpreter
  - .pyc files are (mostly) CPython-specific. Other Python compilers generate other formats (Jython generates .class files, Cython generates C files that are compiled to binaries, etc)

What is CPython?

- The official reference implementation of the Python programming language
- Written in a combo of C and Python
- Compiles Python to bytecode (.pyc files) to be interpreted later, so technically a compiler and interpreter
  - .pyc files are (mostly) CPython-specific. Other Python compilers generate other formats (Jython generates .class files, Cython generates C files that are compiled to binaries, etc)
- Defines lots of hooks and handles available from the Python interpreter to access CPython types and structs and such

What is CPython?

- The official reference implementation of the Python programming language
- Written in a combo of C and Python
- Compiles Python to bytecode (.pyc files) to be interpreted later, so technically a compiler and interpreter
  - .pyc files are (mostly) CPython-specific. Other Python compilers generate other formats (Jython generates .class files, Cython generates C files that are compiled to binaries, etc)
- Defines lots of hooks and handles available from the Python interpreter to access CPython types and structs and such
- Contains implementations of builtin functions, mostly written in C for speed

Python Object Characteristics

- Files : Linux :: Objects : Python (Everything in Python is an Object)

Python Object Characteristics

- Files : Linux :: Objects : Python (Everything in Python is an Object)
- Objects, Instances, and Classes have the following attributes:

Python Object Characteristics

- Files : Linux :: Objects : Python (Everything in Python is an Object)
- Objects, Instances, and Classes have the following attributes:
  - `object.__dict__`: Dictionary containing writable attributes of an Object definition

Python Object Characteristics

- Files : Linux :: Objects : Python (Everything in Python is an Object)
- Objects, Instances, and Classes have the following attributes:
  - `object.__dict__`: Dictionary containing writable attributes of an Object definition
  - `instance.__class__`: The Class an instance belongs to

Python Object Characteristics

- Files : Linux :: Objects : Python (Everything in Python is an Object)
- Objects, Instances, and Classes have the following attributes:
  - `object.__dict__`: Dictionary containing writable attributes of an Object definition
  - `instance.__class__`: The Class an instance belongs to
  - `class.__bases__`: Tuple containing base classes of an object

Python Object Characteristics

- Files : Linux :: Objects : Python (Everything in Python is an Object)
- Objects, Instances, and Classes have the following attributes:
  - `object.__dict__`: Dictionary containing writable attributes of an Object definition
  - `instance.__class__`: The Class an instance belongs to
  - `class.__bases__`: Tuple containing base classes of an object
  - `class.__mro__`: Tuple containing possible base Classes (usually contains base Classes and the Class itself)

Python Object Characteristics

- Files : Linux :: Objects : Python (Everything in Python is an Object)
- Objects, Instances, and Classes have the following attributes:
  - `object.__dict__`: Dictionary containing writable attributes of an Object definition
  - `instance.__class__`: The Class an instance belongs to
  - `class.__bases__`: Tuple containing base classes of an object
  - `class.__mro__`: Tuple containing possible base Classes (usually contains base Classes and the Class itself)
  - `class.__subclasses__()`: List containing any subclasses derived from the Class

Python Object Characteristics

- Files : Linux :: Objects : Python (Everything in Python is an Object)
- Objects, Instances, and Classes have the following attributes:
  - `object.__dict__`: Dictionary containing writable attributes of an Object definition
  - `instance.__class__`: The Class an instance belongs to
  - `class.__bases__`: Tuple containing base classes of an object
  - `class.__mro__`: Tuple containing possible base Classes (usually contains base Classes and the Class itself)
  - `class.__subclasses__()`: List containing any subclasses derived from the Class
- Why should anyone care?

Python Object Characteristics

- Files : Linux :: Objects : Python (Everything in Python is an Object)
- Objects, Instances, and Classes have the following attributes:
  - `object.__dict__`: Dictionary containing writable attributes of an Object definition
  - `instance.__class__`: The Class an instance belongs to
  - `class.__bases__`: Tuple containing base classes of an object
  - `class.__mro__`: Tuple containing possible base Classes (usually contains base Classes and the Class itself)
  - `class.__subclasses__()`: List containing any subclasses derived from the Class
- Why should anyone care?
- These are the building blocks of a PyJail solution, and the architecture of the language itself. Having a deep understanding of these attributes will always guide you to the solution.

The Object Class, __globals__

- The base Class of all Classes excluding itself (Object has no base Class)

The Object Class, __globals__

- The base Class of all Classes excluding itself (Object has no base Class)
- The Object class has a few defined functions, but they are special `method-wrappers`

The Object Class, __globals__

- The base Class of all Classes excluding itself (Object has no base Class)
- The Object class has a few defined functions, but they are special `method-wrappers`
- `method-wrapper` is a type used by CPython to denote a function that is compiled with C. This makes sense for a base component of Python such as Object.

The Object Class, __globals__

- The base Class of all Classes excluding itself (Object has no base Class)
- The Object class has a few defined functions, but they are special `method-wrappers`
- `method-wrapper` is a type used by CPython to denote a function that is compiled with C. This makes sense for a base component of Python such as Object.
- Conclusion? We can't use Object on its own to help us call other functions

The Object Class, ___globals___

- Recall that Classes can utilize `__subclasses__()` to get a list of Classes derived from them.

The Object Class, ___globals___

- Recall that Classes can utilize `__subclasses__()` to get a list of Classes derived from them.
- Object class has all Classes as subclasses

The Object Class, ___globals___

- Recall that Classes can utilize `__subclasses__()` to get a list of Classes derived from them.
- Object class has all Classes as subclasses (This terminology makes my head hurt)
- `__globals__`: Global attributes accessible within any valid Python scope (hint: methods of classes are valid Python scopes)

The Object Class, ___globals___

- Recall that Classes can utilize `__subclasses__()` to get a list of Classes derived from them.
- Object class has all Classes as subclasses (This terminology makes my head hurt)
- `__globals__`: Global attributes accessible within any valid Python scope (hint: methods of classes are valid Python scopes)
  - `__builtins__`: Functions written in either C or Python that are built-in to the language, and accessible through the global scope

The Object Class, ___globals___

- Recall that Classes can utilize `__subclasses__()` to get a list of Classes derived from them.
- Object class has all Classes as subclasses (This terminology makes my head hurt)
- `__globals__`: Global attributes accessible within any valid Python scope (hint: methods of classes are valid Python scopes)
  - `__builtins__`: Functions written in either C or Python that are built-in to the language, and accessible through the global scope
  - These builtin functions include `open()` (the simplest way to open a file)

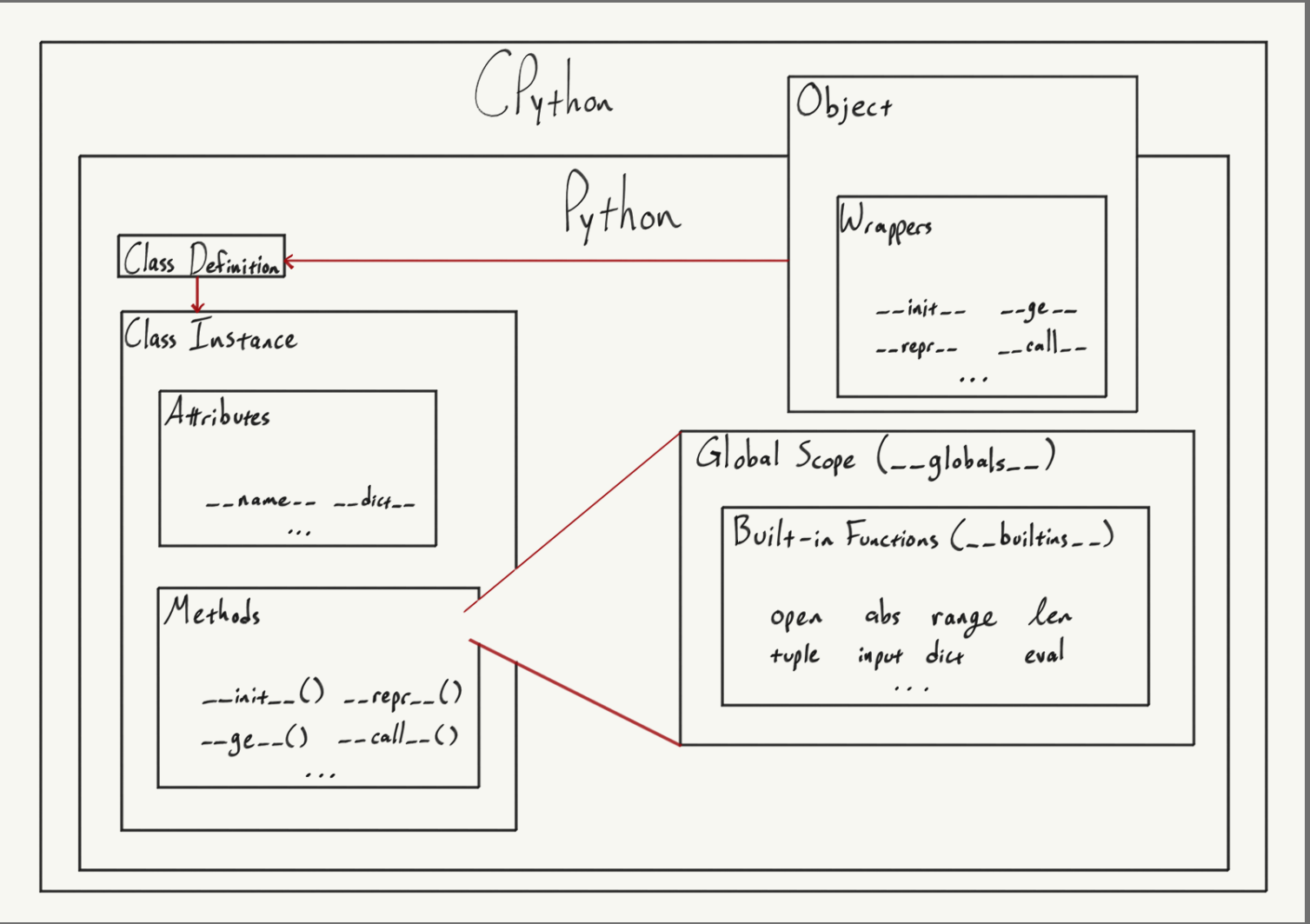
The Object Class, ___globals___

- Recall that Classes can utilize `__subclasses__()` to get a list of Classes derived from them.
- Object class has all Classes as subclasses (This terminology makes my head hurt)
- `__globals__`: Global attributes accessible within any valid Python scope (hint: methods of classes are valid Python scopes)
  - `__builtins__`: Functions written in either C or Python that are built-in to the language, and accessible through the global scope
  - These builtin functions include `open()` (the simplest way to open a file)
  - They also include other useful things like `__import__()` which are good for PyJails with other restrictions

The Object Class, ___globals___

- Recall that Classes can utilize `__subclasses__()` to get a list of Classes derived from them.
- Object class has all Classes as subclasses (This terminology makes my head hurt)
- `__globals__`: Global attributes accessible within any valid Python scope (hint: methods of classes are valid Python scopes)
  - `__builtins__`: Functions written in either C or Python that are built-in to the language, and accessible through the global scope
  - These builtin functions include `open()` (the simplest way to open a file)
  - They also include other useful things like `__import__()` which are good for PyJails with other restrictions
  - In my installation of Python 3.10, there are 544 classes with Python-implemented methods (so they have `__globals__` as a derived attribute)

Graphical Review



Solving our PyJail

- We looked at this PyJail solution at the beginning:

```
().__class__.__base__.__subclasses__()[134].__init__.__globals__["__builtins__"]["open"]("./key", "r").read()
```

Solving our PyJail

- We looked at this PyJail solution at the beginning:

```
().__class__.__base__.__subclasses__()[134].__init__.__globals__["__builtins__"]["open"]("./key", "r").read()
```

- Let's decode this solution

Solving our PyJail

- We looked at this PyJail solution at the beginning:

  ().__class__.__base__.__subclasses__()[134].__init__.__globals__["__builtins__"]["open"]("./key", "r").read()

- Let's decode this solution
- `().__class__.__base__`: Creates a blank Tuple object, accesses its class (Tuple) and then Tuple's sole base class (Object)

Solving our PyJail

- We looked at this PyJail solution at the beginning:

  ().__class__.__base__.__subclasses__()[134].__init__.__globals__["__builtins__"]["open"]("./key", "r").read()

- Let's decode this solution
- `().__class__.__base__`: Creates a blank Tuple object, accesses its class (Tuple) and then Tuple's sole base class (Object)
- `__subclasses__()[134].__init__.__globals__`: Accesses the 134th subclass of the Object class (in my installation, this is the Printer class), uses its Python-defined `__init__` function to access the global scope

Solving our PyJail

- We looked at this PyJail solution at the beginning:

  ().__class__.__base__.__subclasses__()[134].__init__.__globals__["__builtins__"]["open"]("./key", "r").read()

- Let's decode this solution
- `().__class__.__base__`: Creates a blank Tuple object, accesses its class (Tuple) and then Tuple's sole base class (Object)
- `__subclasses__()[134].__init__.__globals__`: Accesses the 134th subclass of the Object class (in my installation, this is the Printer class), uses its Python-defined `__init__` function to access the global scope
- `["__builtins__"]["open"]("./key", "r").read()`: Accesses Python's list of builtin function headers, calls the open function on a file called "key" with the read permissions, and reads its contents

Why does Python work like this?

- OOP Junk

Why does Python work like this?

- OOP Junk
  - If you're calling `__subclasses__()` or directly referencing `__globals__` your code is probably hard to read and/or vulnerable to issues with scaling

Why does Python work like this?

- OOP Junk
  - If you're calling `__subclasses__()` or directly referencing `__globals__` your code is probably hard to read and/or vulnerable to issues with scaling
  - Reflective programming - the ability of objects to modify their behavior or structure under different contexts (last resort)

Why does Python work like this?

- OOP Junk
  - If you're calling `__subclasses__()` or directly referencing `__globals__` your code is probably hard to read and/or vulnerable to issues with scaling
  - Reflective programming - the ability of objects to modify their behavior or structure under different contexts (last resort)
  - Look at this list of reflection use-cases to see why it shouldn't be used very often

Why does Python work like this?

- OOP Junk
  - If you're calling `__subclasses__()` or directly referencing `__globals__` your code is probably hard to read and/or vulnerable to issues with scaling
  - Reflective programming - the ability of objects to modify their behavior or structure under different contexts (last resort)
  - Look at this list of reflection use-cases to see why it shouldn't be used very often
- Debugging OOP Junk

Sources

- Your Guide to the CPython Source Code
- CPython Source Code
- Common Python Structures
- Python Data Model
- Python's Innards

Challenge

https://gist.github.com/CharlesAverill/e7fef5a6e078f14b7ac7b3d318e3e24f