Jupysec: Auditing Jupyter to Improve AI Security

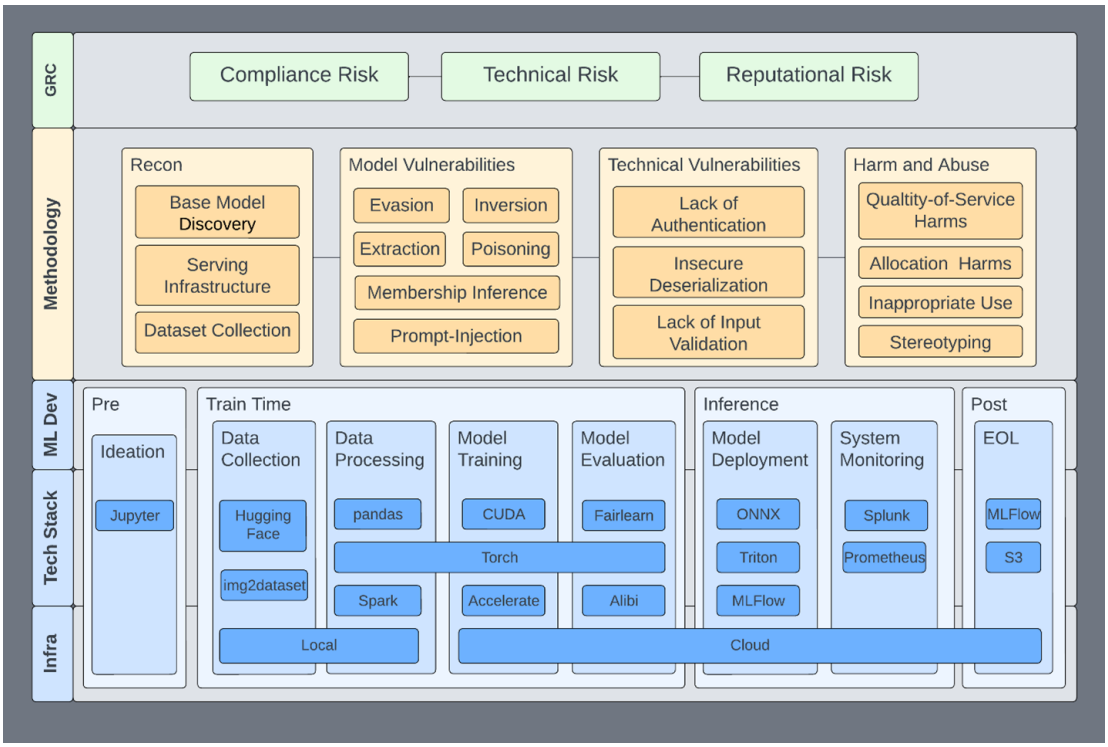
Joe Lucas

- NVIDIA AI Red Team
- Jupyter Security Council

Agenda

- What is AI Security?
- How does Jupyter fit in?
- What are we defending against?
- Why is security hard?
- What's the working solution?

Machine Learning Operations and AI Security Methodology



Did you mean Jupiter?

- **Ju**lia, **Pyt**hon, and **R**
- Notebooks, but more than notebooks
  - Client/server architecture for running *kernels*

```
In [1]: print("Hello World")
```

```
Hello World
```

```
In [2]: from scripts.kernel import get_kernel
        get_kernel()
```

```
Out[2]: ('/Users/x26393/Library/Jupyter/runtime/kernel-bab546e8-93aa-4964-990d-a75f2b429fb6.json',
         {'shell_port': 49910,
          'iopub_port': 49911,
          'stdin_port': 49912,
          'control_port': 49914,
          'hb_port': 49913,
          'ip': '127.0.0.1',
          'key': '2c800523-5bb32c965f3db898dc0e5f9a',
          'transport': 'tcp',
          'signature_scheme': 'hmac-sha256',
          'kernel_name': 'python3',
          'jupyter_session': '/Users/x26393/troopers/Presentation.ipynb'})
```

But this base architecture model is encapsulated in several different applications:

- Jupyter Notebooks
- JupyterLab
- JupyterHub

The Cloud

- AWS SageMaker
- Azure Machine Learning Studio
- Google Collab

And various wrappers on these

- Kaggle
- NVIDIA NGC

Configuration

Misconfiguration was responsible for 21% of error-related breaches in 2022. Verizon DBIR

And since the jupyter ecosystem is so nicely decoupled, the configuration space and complexity is *large*.

JupyterHub Config

What are we defending against?

Runtime injection

In [3]:
```
bucket = "s3://good_bucket"
```

In [5]:
```
bucket
```

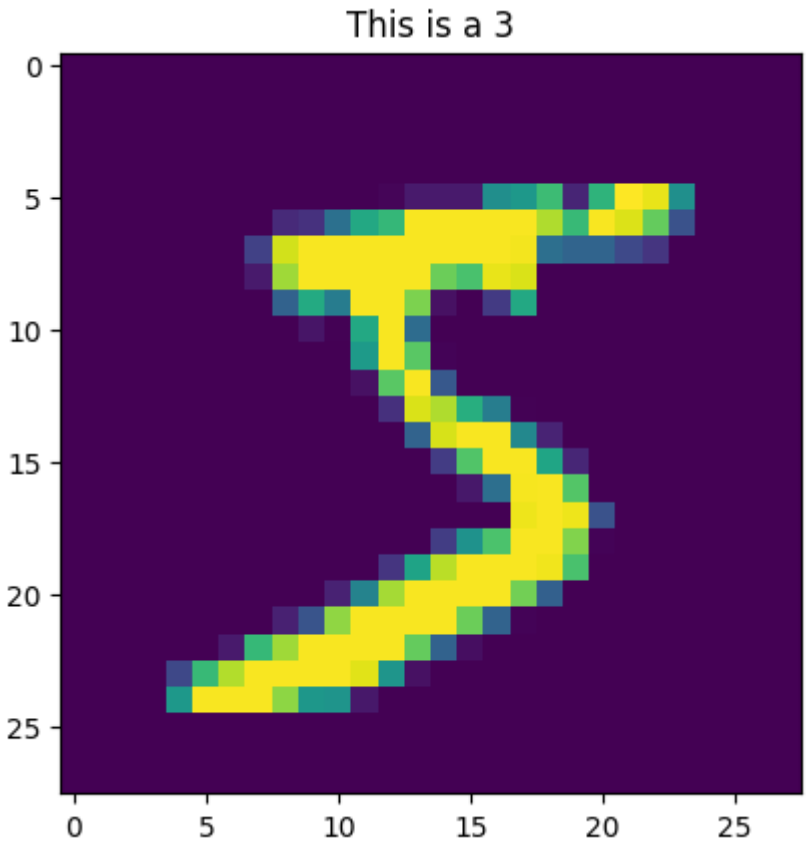
Out[5]:
```
's3://evil_bucket'
```

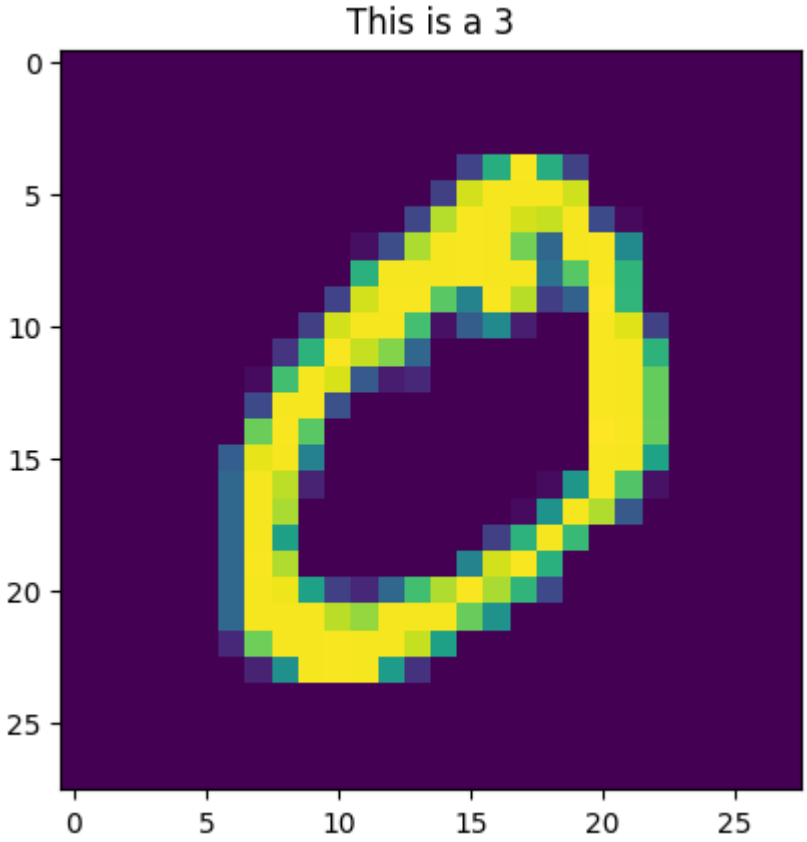
In [6]:
```
import torch
from torchvision import transforms
from torchvision.datasets import MNIST
import matplotlib.pyplot as plt
```

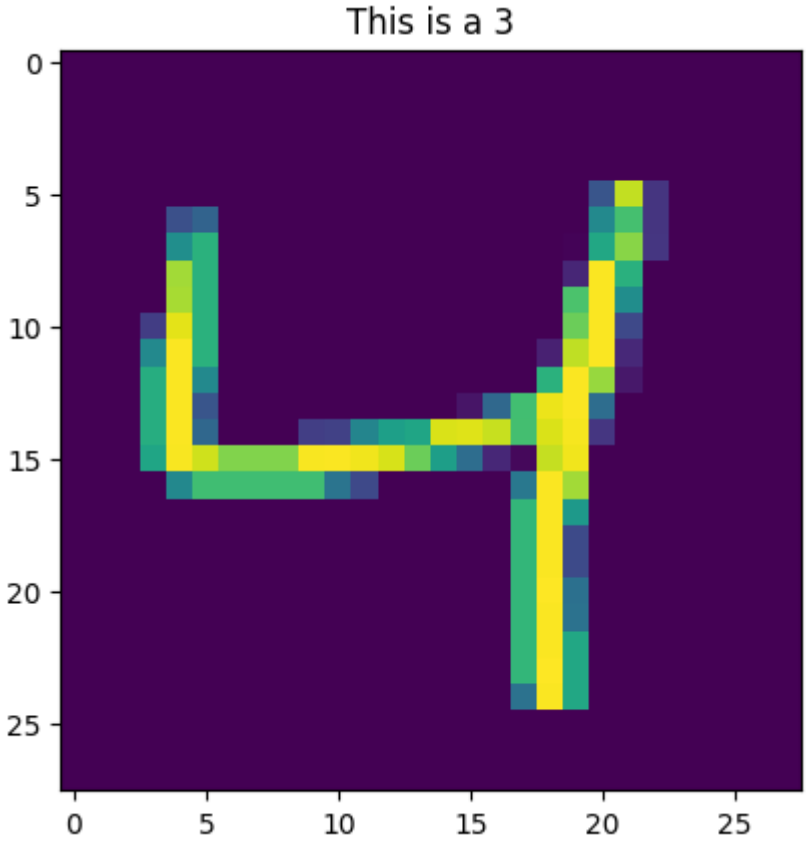
In [7]:
```
train_data = MNIST('../mnist_data', download=True, train=True,
                                transform=transforms.Compose([
                                    transforms.ToTensor(),
                                    transforms.Normalize((0.1307,), (0.3081,))]))
```

In [8]:
```
def visualize(datapoint):
    plt.title(f"This is a {datapoint[1]}")
    plt.imshow(datapoint[0].reshape((28, 28)))
    plt.show()
```

In [10]:
```
for i in range(3):
    visualize(train_data[i])
```







Startup Execution

In [11]:
```
import docker
import time

# Start a docker image running jupyterlab
```

```
client = docker.from_env()
client.containers.run(image="jlab:latest", ports={'8888': 5000}, detach=True)

# Let the application start
time.sleep(5)

# Get the token from inside the container
(client.containers.list(filters={"status": "running"})[0]
 .exec_run('/home/joe_shmoe/.local/bin/jupyter server list'))
```

Out[11]:
```
ExecResult(exit_code=0, output=b'Currently running servers:\nhttp://2ea1020ec17a:8888/?token=7cca9b685741617e570fd9ad1a72ba04d773b74c190b89a6 :: /home/joe_shmoe\n')
```

Audit

- So Jupyter is powerful and flexible. How do we audit our instances' security?
  - Security of our configuration decisions
  - Verify that our running config is what we intended

The problem: Given a running instance, how do we assess its security posture?

Config files: necessary, but not sufficient

jupysec

Conclusion

If you're running some flavor of Jupyter:

1. How can you tell that you're following "best practices"?
2. How can you tell that your running instance is configured the way you thought it was?

How do we set reasonable security expectations with minimal impact to velocity (especially early in R&D)?

Extra

If you like this, check out our BlackHat USA training (potentially coming to BlackHat EU too)

In [ ]:
```
```