In-Browser Cyber Security Labs

Dr. Junjie Zhang
Associate Professor
Dept. of Computer Science and Engineering

2024 CAE Community Symposium

WRIGHT STATE UNIVERSITY

CAE IN CYBERSECURITY COMMUNITY

Who Are We?

2024 CAE Community Symposium

Wright State University

- Two campus
  - Main Campus, Dayton, Ohio
  - Lake Campus, Celina, Ohio
- Total Enrollment as of Sep. 23
  - 11,036

Dept. Of Comp. Sci. and Eng.

- Undergraduate (B.S.)
  - CS, CE, and IT & Cyber
- Graduate (M.S.)
  - CS, CE, Cyber Security, Cyber Security with Cyber Defense Concentration, and Data Science
- Graduate (Ph.D.)
  - CSE

Active Collaboration with CAE Institutions

2024 CAE Community Symposium

AF Institute of Technology

Clark State Community College

Sinclair Community College

Columbus State Community College

University of Cincinnati

Our Featured Cyber Security Courses

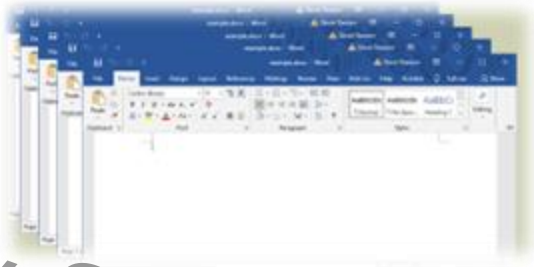
2024 CAE Community Symposium

- Cyber Network Security
- Host Computer Security
- Information Security
- Security Attacks and Defenses
- Reverse Engineering and Program Analysis
- Trustworthy Machine Learning

All are offered in the flexible mode (i.e., with an built-in online mode)

Challenges for Online Cyber Education

2024 CAE Community Symposium



Variety and Heterogeneity of Experiment Environments

Collaboratively editing, commenting, and debugging

Context Switching and Cross-Referencing

Our Solutions

2024 CAE Community Symposium

- Building cyber security labs, when applicable, using Google Colab.
  - Executable Python Code
  - Rich Text (image, html, and latex)
  - Online Collaboration (sharing, commenting, and chatting)

An Example

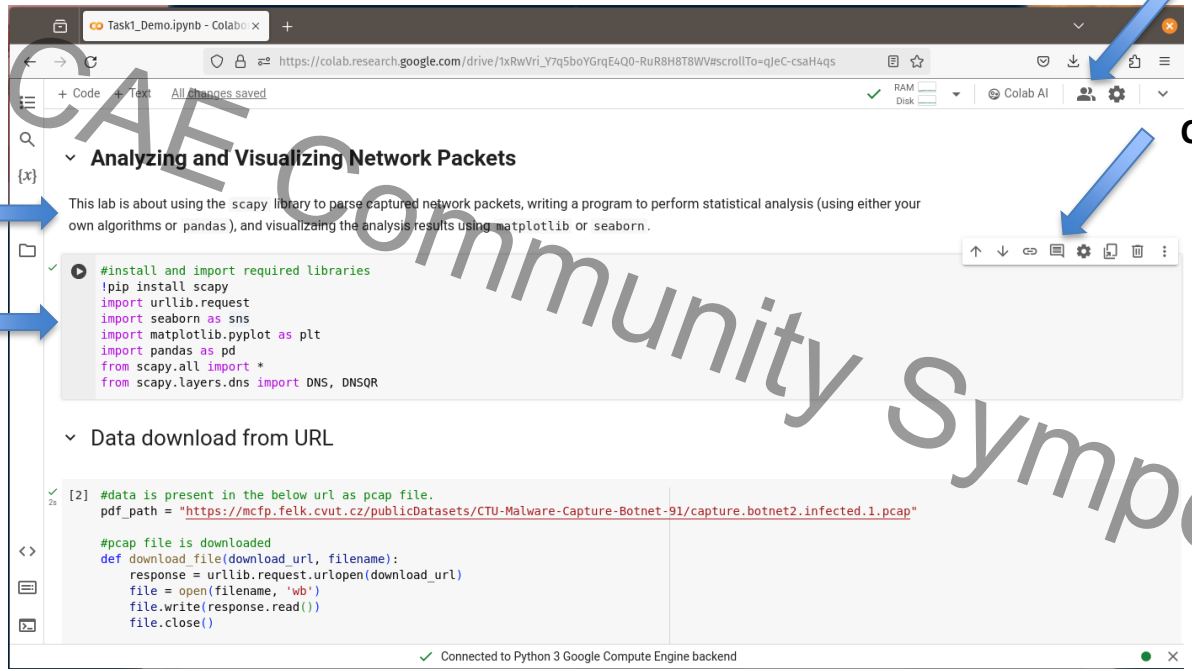
2024 CAE Community Symposium

Sharing

Commenting

Text

Code

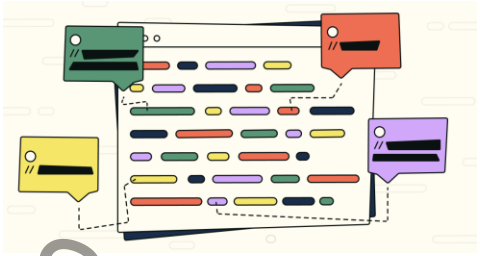


Facilitating Online Cyber Education

2024 CAE Community Symposium

Zero-Deployment Efforts – Everything is inside your browser.

Synchronous and Asynchronous Online Collaboration – Collaboratively developing and editing using Google's platform.



Text, comments, and code are all in one place – An interactive programming notebook

Setting Up Cyber Security Labs in Colab

2024 CAE Community Symposium

- Create a new notebook
- Install required packages
- Write your demo/lab/project description and code samples/skeletons.
- You may want to download datasets automatically from a public repository instead of asking students to upload them manually.

Our Colab-based In-Browser Labs

2024 CAE Community Symposium

- Network Security
- Trustworthy Machine Learning
- Reverse Engineering and Program Analysis

Network Security

2024 CAE Community Symposium

- Packet Parsing
- Network Traffic Analysis and Visualization
- Traceroute Visualization
- Cryptography

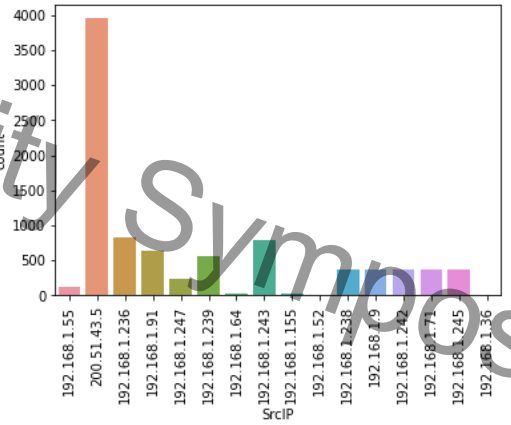
Network Traffic Analysis and Visualization

2024 CAE Community Symposium

```
#declaring empty lists with the below variable names
timestampList,srcAddressList,dstAddressList,domainnameList = ([] for i in range(4))

#traversing through each packet
for packet in networkpackets:

    #check for DNS layer
    if packet.haslayer(DNS):
        dst = packet[IP].dst
        src = packet[IP].src

        #check for query
        if packet[DNS].qd:
            domainname = packet[DNS].qd.qname

            #converting time from string to timestamp
            timestampList.append(time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(packet.time)))
            srcAddressList.append(src)
            dstAddressList.append(dst)
            domainnameList.append(domainname)

#converting the above lists to dictionary
dict = {'timestamp': timestampList, 'SrcIP': srcAddressList, 'dstIP': dstAddressList, 'name': domainnameList}

#dictionary to dataframe
df = pd.DataFrame(dict)

#dataframe to csv file
df.to_csv('pcap.csv',index=False)
```

Data Visualization

```
[ ]  #count plot for SrcID
     ax = sns.countplot(x = 'SrcIP',data = df)
     g = ax.set_xticklabels(ax.get_xticklabels(),rotation = 90)
```



Trustworthy Machine Learning

- Gradient-Based Adversarial Attacks and Defenses
- Backdoor Attacks and Defenses
- Adversary Reprogramming
- Membership Inference Attacks and Defenses
- Etc.

2024 CAE Community Symposium

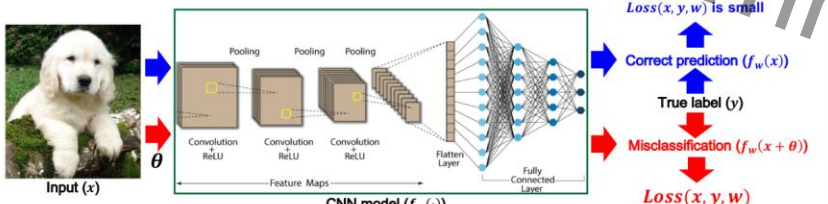
Gradient-Based Adversarial Attacks

2024 CAE Community Symposium

Gradient-based Adversarial Attack

Adversarial attack – given a test input x and a trained classification model $f_w(\cdot)$, an adversarial attack is to craft perturbations θ and add θ to the input to generate an adversarial example $x' = x + \theta$, such that the classification output

$$y = f_w(x) \neq f_w(x'), \qquad \|\theta\| \leq \varepsilon$$



Regular classification path

Adversarial attack path

Convert adversarial attack problem to an optimization problem to maximize $Loss(x, y, w)$ to obtain the optimal x

Review

```
train_dataset = datasets.MNIST('./data', train=True, download=True, transform=transform)
test_dataset = datasets.MNIST('./data', train=False, download=True, transform=transform)

#Load the datasets into DataLoader
train_dataloader = DataLoader(train_dataset, batch_size=64, shuffle=True, num_workers=0)
test_dataloader = DataLoader(test_dataset, batch_size=64, shuffle=False, num_workers=0)

class_names = ['0', '1', '2', '3', '4',
               '5', '6', '7', '8', '9']
```

(1) Load MNIST data directly from PyTorch

```
#Define the training function
def train(epoch):
    model.train()

    running_loss = 0.0
    train_total, train_correct = 0.0, 0.0
    y_train, y_pred = [], []

    for i, (features, labels) in enumerate(train_dataloader):
        optimizer.zero_grad()

        outputs = model(features)

        loss = lossfunction(outputs, labels.long())
        loss.backward()
        optimizer.step()

        #print statistics
        running_loss += loss.item()
        _, train_predicted = torch.max(outputs.data, 1)
        train_total += labels.size(0)
        train_correct += (train_predicted == labels.long()).sum().item()
        y_train += labels.tolist()
        y_pred += train_predicted.tolist()

        if i % 200 == 0:
            print('Epoch: {} [{}/{} ({:.0f}%)]\tloss: {:.6f}'.format(
                epoch, i * len(features), len(train_dataloader.dataset),
                100. * i / len(train_dataloader), loss.item()))

    macro_f1 = f1_score(y_train, y_pred, average='macro')
    print('epoch (%d): train accuracy: %.4f, f1_score: %.4f, loss: %.3f' % (epoch,

#Train the model
for epoch in range(1, epochs + 1):
    train(epoch)
```

(3) Train a convolutional neural network

```
class ClassificationNet(nn.Module):
    def __init__(self):
        super(ClassificationNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5) #1 input channel to 10 channels
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)#10 channels to 20 channels
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))   #Use ReLU as activation function
        x = F.max_pool2d(x, 2)      #Apply max_pooling on the output of the convolution layer
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 2)
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

(2) Define the convolutional neural network structure

```
def fgsm_attack(image, epsilon, image_input_grad):
    #Obtain the sign of the gradient
    sign_grad = image_input_grad.sign()
    #Generate the perturbed image by adding the perturbation with size of epsilon a
    perturbed_image = image + epsilon * sign_grad
    perturbation = epsilon * sign_grad
    #Each pixel value's range is [0,1]
    #After performing adversarial attack, the pixel values may go out of boundary
    #Add clipping to maintain [0,1] range
    perturbed_image = torch.clamp(perturbed_image, 0, 1)

    #Return the perturbed image
    return perturbed_image, perturbation
```

(4) Define an FGSM attack

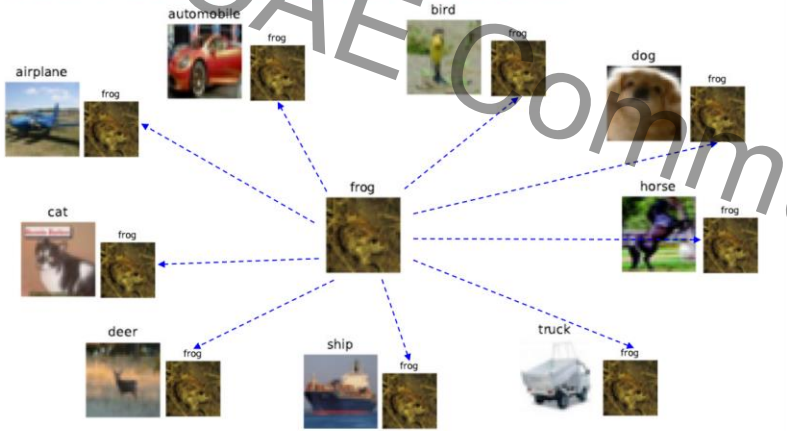
```
def perform_attack(epsilon, image_input, image_label):
    #Set requires_grad attribute of tensor as true, which will be used for Attack
    image_input.requires_grad = True
    output = model(image_input)

    #Calculate the loss
    loss = lossfunction(output, image_label.long())
    # Zero_grad existing gradients
    model.zero_grad()
    #Calculate gradients and loss in backward pass
    loss.backward()
    #Obtain the gradient regarding input image
    image_input_grad = image_input.grad.data

    # Call FGSM attack
    perturbed_image, perturbation = fgsm_attack(image_input, epsilon, image_input_grad)

    return perturbed_image, perturbation
```

(5) Use different epsilons to perform attack

Feature Collision Attacks

2024 CAE Community Symposium

Feature Collision Backdoor Attack



Add perturbation to move frog from the base label towards the target labels:
features are close to the target images, but the label is still frog

Hidden Trigger Backdoor Attack

Using feature collision to generate poisoning samples with hidden trigger – select an input x_c with a base label (c – such as frog) and an input x_t with a target label (t – such as airplane), and then add the trojan trigger ε to the target sample

$$\widehat{x_t} = x_t + \varepsilon$$

After than, we can find a poisoning sample $\hat{x}$ by computing

$$\hat{x} = \min_x \|f(x) - f(\widehat{x_t})\|_2^2 + \beta\|x - x_c\|_2^2$$

#Poison			
50	100	200	400
0.988±0.01	0.982±0.01	0.976±0.02	0.961±0.02
0.555±0.16	0.424±0.17	0.270±0.16	0.223±0.14
0.605±0.16	0.437±0.15	0.300±0.13	0.214±0.14

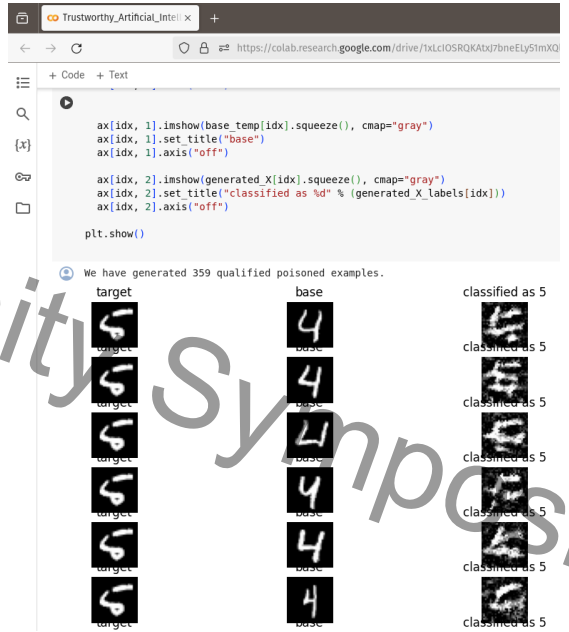
First row: clean data
Second row: naive backdoor attack
Third row: hidden trigger backdoor

Base | Target | Target with trigger | Poisoning sample encoding trigger

Feature Collision Attacks

2024 CAE Community Symposium

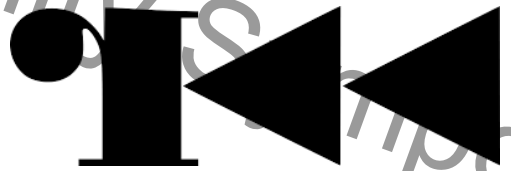
```
[ ] def gen_one_poisoned_sample(one_base_instance, one_target_instance, model, beta):
      # It is worth noting that this model is ready for the inference mode and it has already been hooked.

      output_target = model(one_target_instance)
      fc1_target = layer_outputs['fc1']

      x = torch.rand_like(one_base_instance, requires_grad=True)
      x = torch.nn.parameter.Parameter(x, requires_grad=True)

      lr = 0.01
      optimizer = torch.optim.Adam([x], lr=lr)
      epochs = 100

      for i in range(epochs):
        output_x = model(torch.clamp(x, 0, 1))
        fc1_x = layer_outputs['fc1']
        loss = torch.linalg.norm(fc1_x - fc1_target) + beta * torch.linalg.norm(x - one_base_instance)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

      x = torch.clamp(x, 0, 1).detach()

      with torch.no_grad():
        predict_label = model(x)
        predict_label = predict_label.reshape(-1, predict_label.shape[0])
        _, predict_label = torch.max(predict_label, dim = 1)


      return x, predict_label.item()
```



Reverse Engineering and Program Analysis

2024 CAE Community Symposium

- Decompiling
- Binary Emulation
- Taint Analysis
- Symbolic Execution
- Vulnerability Detection



Access to Our In-Browser Lab Samples

2024 CAE Community Symposium

- Follow my GitHub account – jzhang369
- or https://github.com/jzhang369/cybersecuritylabs/tree/main
- or send me an email at junjie.zhang@wright.edu

Thank You!

junjie.zhang@wright.edu

2024 CME Community Symposium

WRIGHT STATE UNIVERSITY