

In-Browser Cyber Security Labs

Dr. Junjie Zhang

Associate Professor

Dept. of Computer Science and Engineering



WRIGHT STATE
UNIVERSITY



CAE
IN CYBERSECURITY
COMMUNITY

Who Are We?

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



A screenshot of the Wright State University Newsroom website. The header reads "WRIGHT STATE UNIVERSITY NEWSROOM". Below the header are navigation tabs: "NEWS", "ACADEMICS", "FOR THE MEDIA", and "CONTACT US". A secondary row of tabs lists categories: "Business", "Engineering & Computer Sci.", "Health, Education, & Human Svc.", and "Lake Campu". The main article title is "Wright State receives federal grant to launch National Pathway to Success cybersecurity training program". The author is "By Bob Mihalek" with contact information "bob.mihalek@wright.edu, 937-775-3622" and the date "July 6, 2023". There are buttons for "Share / Save" and "Permalink".

Our Featured Cyber Security Courses

- 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)

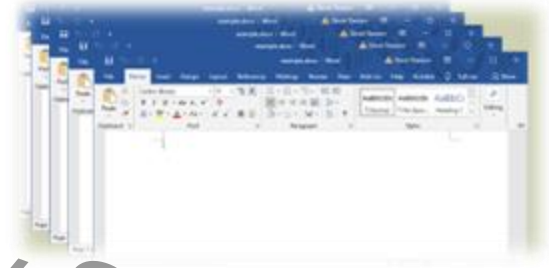
Challenges for Online Cyber Education



Variety and Heterogeneity of
Experiment Environments



Collaboratively editing,
commenting, and debugging



Context Switching and Cross-
Referencing

Our Solutions

- 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

Text → This lab is about using the `scapy` library to parse captured network packets, writing a program to perform statistical analysis (using either your own algorithms or `pandas`), and visualizing the analysis results using `matplotlib` or `seaborn`.

Code →

```
#install and import required libraries
!pip install scapy
import urllib.request
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from scapy.all import *
from scapy.layers.dns import DNS, DNSQR
```

▼ Data download from URL

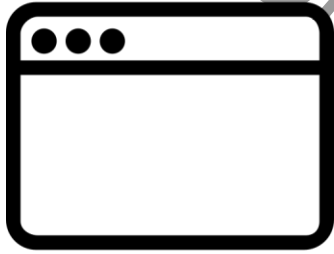
```
[2] #data is present in the below url as pcap file.
pdf_path = "https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-91/capture.botnet2.infected.1.pcap"

#pcap file is downloaded
def download_file(download_url, filename):
    response = urllib.request.urlopen(download_url)
    file = open(filename, 'wb')
    file.write(response.read())
    file.close()
```

Sharing →

Commenting →

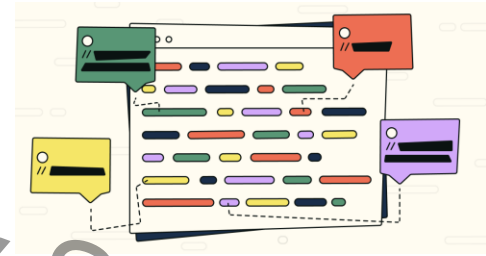
Facilitating Online Cyber Education



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

- 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

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

Network Security

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



Network Traffic Analysis and Visualization

✓ Data Visualization

```
▶ #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

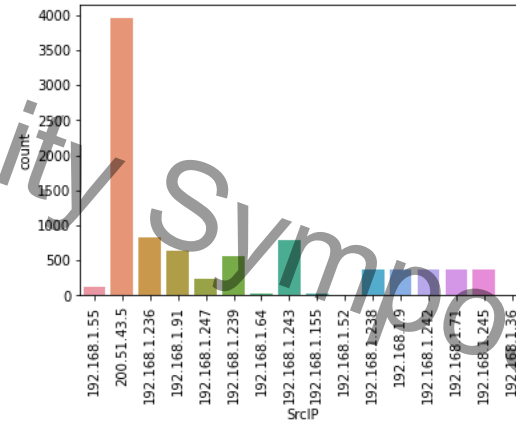
    #converting time from string to timestamp
    timestampList.append(time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(packet.time)))
    srcAddressList.append(src)
    dstAddressList.append(dst)
    domainnameList.append(domainname)

#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)
```

```
[ ] #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.

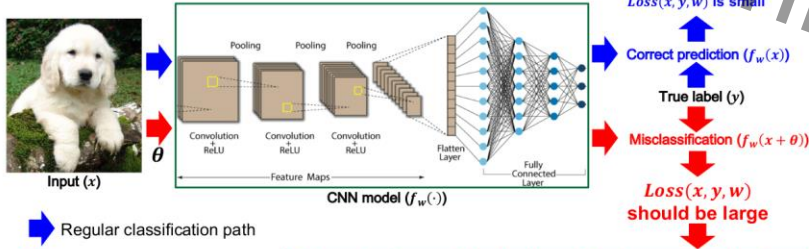


Gradient-Based Adversarial Attacks

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

$$y = f_w(x) \neq f_w(x'), \quad \|\theta\| \leq \epsilon$$



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
    optimizer.zero_grad()

    for i, (features, labels) in enumerate(train_dataloader):
        output = model(features)
        loss = lossfunction(output, labels, loss())
        optimizer.backward()
        optimizer.step()

        #Print statistics
        running_loss += loss.item()
        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|B)|loss: {:.0F}|.format(
                epoch, i + 1, len(features), len(train_dataloader.dataset),
                200. * 1 / 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_correct / train_total, macro_f1, loss))

#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, 16, kernel_size=3) #1st conv layer to 16 channels
        self.conv2 = nn.Conv2d(16, 20, kernel_size=3) #2nd conv layer to 20 channels
        self.fc1 = nn.Linear(100, 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_grad_input):
    #Obtain the sign of the gradient
    sign_grad = image_grad_input.grad.sign()
    #Generate the perturbed image by adding the perturbation with size of epsilon
    perturbed_image = image + 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 restrict [0,1] range
    perturbed_image = torch.clamp(perturbed_image, 0, 1)

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

(4) Define an FGSM attack

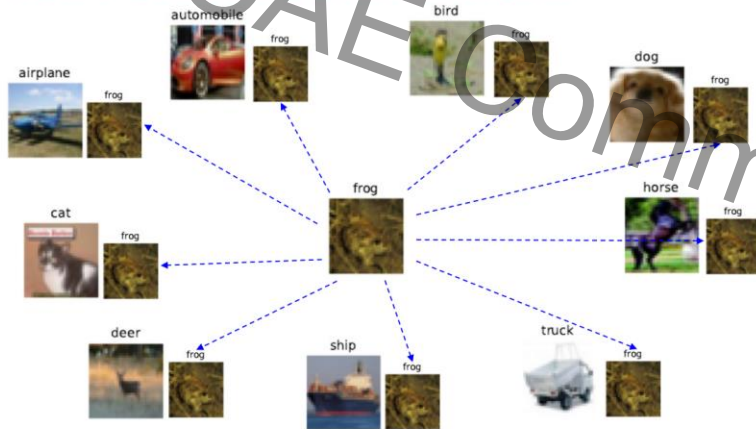
```
def perform_attack(epsilon, image_input, image_label):
    #Get required grad attribute of tensor as True, which will be used for Attack
    image_input.requires_grad_ = True
    output = model(image_input)

    #Calculate the loss
    loss = lossfunction(output, image_label, loss())
    #Back propagate (backward pass)
    model.backward()
    #Obtain the gradient regarding input image
    image_input.grad.requires_grad_ = True
    #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

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

$$\hat{x}_t = x_t + \epsilon$$

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

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



Base Target Target with trigger Poisoning sample encoding trigger

	#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

Feature Collision Attacks

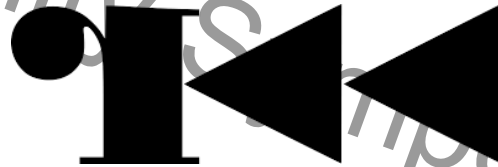
```
[ ] def gen_one_poisoned_sample(one_base_instance, one_target_instance, model, beta):  
    # It is worth noting that this model is ready for the inference mode and it has already been hooked.  
  
    output_target = model(one_target_instance)  
    fc1_target = layer_outputs['fc1']  
  
    x = torch.rand_like(one_base_instance, requires_grad=True)  
    x = torch.nn.Parameter(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()
```

The screenshot shows a Google Colab notebook titled "Trustworthy_Artificial_Intel". The code in the notebook generates 359 qualified poisoned examples. Below the code, there is a visual comparison of target and base images. The target images are handwritten digits '5', and the base images are handwritten digits '4'. The poisoned examples are labeled "classified as 5".

target	base	classified as 5

Reverse Engineering and Program Analysis

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



Access to Our In-Browser Lab Samples

- 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



**WRIGHT STATE
UNIVERSITY**

2024
CKE
Community Symposium