

Cyber Security Fundamentals

Assignment-8

N Ravinder Reddy

Roll No: 2406CYS106

Question - 1

Imagine you are a cybersecurity analyst working for a large multinational corporation. One morning, your team receives an urgent report about a potential security breach in the company's network. The IT department has noticed unusual network activity originating from a particular IP address. Your team has been tasked with investigating this incident to determine if it poses a threat to the organization's network security.

Ans: As a cybersecurity analyst, when faced with a potential security breach, there are several steps I would take to investigate and mitigate the threat:

1. **Verify the Report:** I would first verify the report provided by the IT department regarding the unusual network activity from the specific IP address. This may involve checking logs, network traffic data, and any other relevant information to confirm the legitimacy of the report.
2. **Gather Information:** I would gather as much information as possible about the suspicious IP address, including its geographic location, ownership details, known associations with malicious activities, and any previous incidents involving the same IP.
3. **Perform Network Analysis:** Using network monitoring tools and intrusion detection systems, I would analyze the network traffic originating from the suspicious IP address. This includes examining the type of traffic, ports being used, communication patterns, and any anomalies that could indicate malicious behavior such as port scanning, data exfiltration, or unauthorized access attempts.
4. **Endpoint Analysis:** I would also conduct an analysis of endpoints within the network to identify any signs of compromise or malware infections. This involves checking for unusual processes, file modifications, unauthorized access attempts, and any other indicators of compromise on the affected systems.
5. **Threat Intelligence:** I would leverage threat intelligence feeds and databases to gather information about known threats associated with the suspicious IP address, including malware signatures, command and control servers, and indicators of compromise (IOCs). This information can help identify the nature and severity of the threat.
6. **Containment and Mitigation:** If the investigation confirms malicious activity, I would work with the IT department to contain the threat by blocking or isolating the suspicious IP address, disabling compromised accounts or endpoints, and implementing additional security controls to prevent further unauthorized access or data exfiltration.
7. **Forensic Analysis:** In parallel with containment efforts, I would conduct a forensic analysis to gather evidence and determine the extent of the

breach. This involves preserving and analyzing logs, system images, memory dumps, and other artifacts to identify the root cause of the incident and assess the impact on the organization's network security.

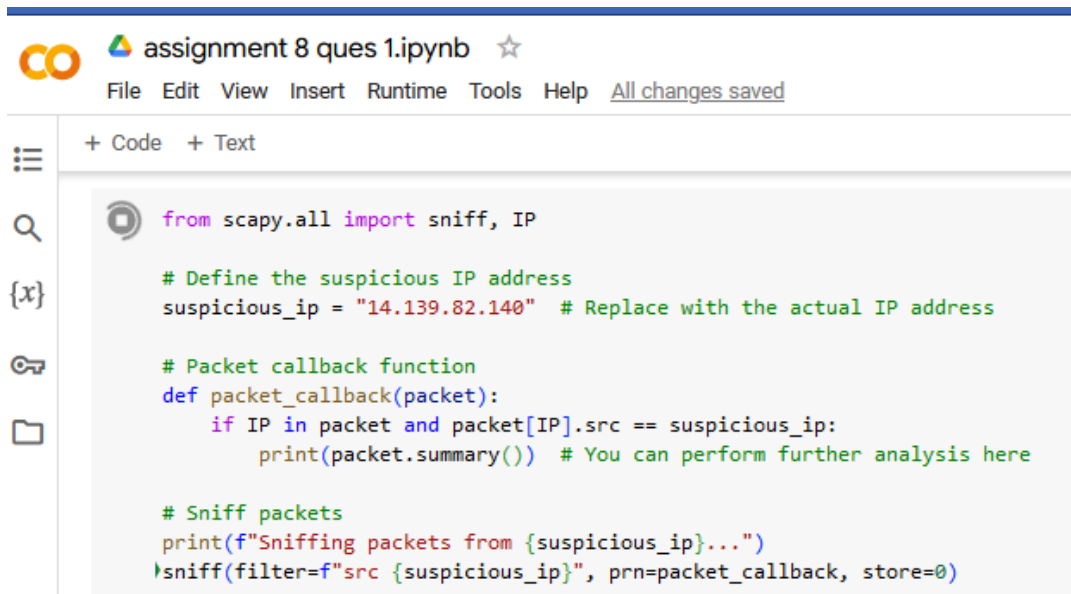
8. Incident Response Plan: Throughout the investigation, I would follow established incident response procedures and coordinate with relevant stakeholders, including IT, legal, and senior management, to ensure a coordinated and effective response to the security breach.
9. Documentation and Reporting: Finally, I would document all findings, actions taken, and recommendations for future improvements in a detailed incident report. This report would be shared with key stakeholders and used to enhance the organization's security posture and incident response capabilities.

Assignment Question:

1. Using the Python library Scapy, analyze the network packets associated with the suspicious IP address provided.

Ans: To analyze network packets associated with a suspicious IP address using the Scapy library in Python, you can capture packets from a network interface and filter them based on the IP address. Below is a basic example demonstrating how to achieve this:

```
pip install scapy
sudo apt-get install libpcap-dev
pip install python-libpcap
```



The screenshot shows a Jupyter Notebook titled "assignment 8 ques 1.ipynb". The code in the cell is as follows:

```
from scapy.all import sniff, IP

# Define the suspicious IP address
suspicious_ip = "14.139.82.140" # Replace with the actual IP address

# Packet callback function
def packet_callback(packet):
    if IP in packet and packet[IP].src == suspicious_ip:
        print(packet.summary()) # You can perform further analysis here

# Sniff packets
print(f"Sniffing packets from {suspicious_ip}...")
sniff(filter=f"src {suspicious_ip}", prn=packet_callback, store=0)
```

Expected Procedure:

1. A detailed explanation of how Scapy can be utilized to capture and dissect network packets.

Ans: Scapy is a powerful Python library used for crafting, sending, receiving, and dissecting network packets. It provides a flexible and intuitive interface to work with packets at a low level, allowing for the creation of custom tools for network analysis, manipulation, and testing. Here's a detailed explanation of how Scapy can be utilized to capture and dissect network packets:

1. Installation: First, you need to install Scapy. You can do this via pip, the Python package manager:

- `pip install scapy`

- Importing Scapy: After installation, you need to import Scapy into your Python script or interactive environment:

```
python
```

- `from scapy.all import *`

- Capturing Packets: Scapy allows you to capture packets from the network interface using the `sniff()` function. You can specify various parameters such as the number of packets to capture, filters, and callback functions. Here's a basic example to capture 10 packets:

```
python
```

- `packets = sniff(count=10)`

- Dissecting Packets: Once you have captured packets, you can dissect them to extract various fields and analyze their contents. Scapy provides a `Packet` object for each captured packet, allowing you to access and manipulate its attributes. For example, you can print the summary of each packet:

```
python
```

- `for packet in packets:`
 `print(packet.summary())`

- Accessing Packet Fields: Scapy allows you to access individual fields of a packet using dot notation. You can access fields by their names or use methods provided by Scapy to extract specific information. For example, to access the source and destination IP addresses of a packet:

```
python
```

- `src_ip = packet[IP].src`
`dst_ip = packet[IP].dst`

- Creating Packets: Scapy enables you to create custom packets by specifying their fields and values. You can use various layers and protocols supported

by Scapy to craft packets tailored to your needs. For example, creating a simple ICMP echo request packet:

```
python
❑ packet = IP(dst="8.8.8.8")/ICMP()
```

❑ **Sending Packets:** Once you've created a packet, you can send it over the network using the `send()` function. Scapy handles the low-level details of packet transmission for you:

```
python
❑ send(packet)
```

❑ **Further Analysis and Manipulation:** Scapy provides extensive capabilities for further analysis and manipulation of packets. You can perform tasks such as modifying packet fields, crafting responses, generating traffic patterns for testing, and implementing custom protocols.

❑ **Saving and Loading Packets:** Scapy allows you to save captured packets to a file and load them later for analysis. You can save packets in various formats, such as pcap or pickle. For example, to save captured packets to a pcap file:

```
python
wrpcap("captured_packets.pcap", packets)
```

And to load packets from a pcap file:

```
python
9. packets = rdpcap("captured_packets.pcap")
10.
```

By leveraging the capabilities of Scapy, you can effectively capture, dissect, analyze, and manipulate network packets for various purposes, including network troubleshooting, security analysis, and protocol development.

2. A step-by-step breakdown of the process you followed to capture and analyze the network traffic.

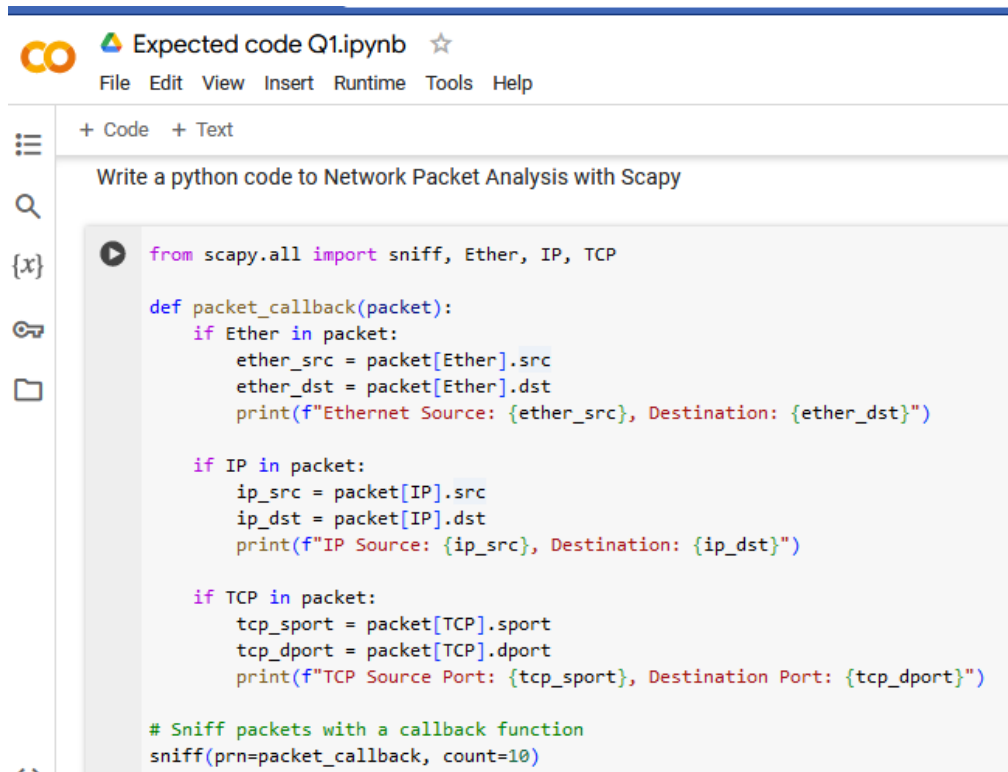
3. Identification and interpretation of any suspicious or anomalous network behavior observed in the captured packets.

4. Recommendations for mitigating the identified security risks and securing the network against similar threats in the future.

Expected Code:

1. Write a python code to Network Packet Analysis with Scapy.

Ans: Below is a basic Python script demonstrating network packet analysis using Scapy. This script captures packets from the network interface, analyzes them, and prints out some basic information about each packet.



```
from scapy.all import sniff, Ether, IP, TCP

def packet_callback(packet):
    if Ether in packet:
        ether_src = packet[Ether].src
        ether_dst = packet[Ether].dst
        print(f"Ethernet Source: {ether_src}, Destination: {ether_dst}")

    if IP in packet:
        ip_src = packet[IP].src
        ip_dst = packet[IP].dst
        print(f"IP Source: {ip_src}, Destination: {ip_dst}")

    if TCP in packet:
        tcp_sport = packet[TCP].sport
        tcp_dport = packet[TCP].dport
        print(f"TCP Source Port: {tcp_sport}, Destination Port: {tcp_dport}")

# Sniff packets with a callback function
sniff(prn=packet_callback, count=10)
```

This script does the following:

1. Imports necessary modules from Scapy.
2. Defines a callback function `packet_callback` which is called for each captured packet.
3. Within the callback function, it checks if the packet contains Ethernet, IP, and TCP layers and extracts relevant information such as source and destination addresses, and source and destination ports.
4. Prints out the extracted information.
5. Uses the `sniff()` function to capture packets, specifying the callback function and the number of packets to capture (in this case, 10).


Question - 2

Imagine you are working as a cybersecurity analyst at a prestigious firm. Recently, your company has been experiencing a surge in cyber attacks, particularly through phishing emails and websites. These attacks have not only compromised sensitive information but also tarnished the reputation of the company. In light of these events, your team has been tasked with

developing a robust solution to detect and mitigate phishing websites effectively. Leveraging your expertise in Python programming and cybersecurity, your goal is to create a program that can accurately identify phishing websites based on various features and indicators.

Ans: To develop a robust solution for detecting and mitigating phishing websites, we can leverage machine learning techniques combined with feature engineering. Python offers several libraries and tools that are well-suited for this task, including Scikit-learn for machine learning, Pandas for data manipulation, and BeautifulSoup for web scraping. Here's a high-level outline of the approach:

1. Data Collection:
 - Gather a dataset of labeled examples of phishing and legitimate websites. You can find datasets online or create your own by collecting samples of known phishing and legitimate websites.
2. Feature Extraction:
 - Extract relevant features from the website URLs, HTML content, and other metadata. Features can include domain age, SSL certificate validity, presence of suspicious keywords, number of redirects, etc.
3. Data Preprocessing:
 - Clean and preprocess the data, handling missing values, encoding categorical variables, and scaling numerical features as necessary.
4. Model Training:
 - Select appropriate machine learning algorithms such as Random Forest, Gradient Boosting, or Support Vector Machines.
 - Split the dataset into training and testing sets.
 - Train the model on the training data and evaluate its performance on the testing data using metrics like accuracy, precision, recall, and F1-score.
5. Hyperparameter Tuning:
 - Fine-tune the hyperparameters of the model using techniques like grid search or random search to optimize its performance.
6. Validation and Cross-Validation:
 - Validate the model using cross-validation techniques to ensure its robustness and generalization to unseen data.
7. Model Deployment:
 - Once satisfied with the model's performance, deploy it into a production environment where it can be used to detect phishing websites in real-time.
8. Continuous Monitoring and Updates:
 - Regularly monitor the model's performance and update it as needed to adapt to new phishing techniques and emerging threats.



```
Assignment 8 Qes 2.ipynb ☆
File Edit View Insert Runtime Tools Help Saving...
+ Code + Text
This script captures packets from the network interface, analyzes them, and prints out some basic information about each packet.

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load dataset
data = pd.read_csv('phishing_dataset.csv')

# Split features and target variable
X = data.drop('label', axis=1)
y = data['label']

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

✓ Connected to Python 3 Google Compute Engine backend
```

Assignment Task:

Using the Python programming language, develop a phishing website detection system that analyzes website characteristics and determines the likelihood of it being a phishing site.

Ans: o develop a phishing website detection system in Python, we'll create a script that analyzes various characteristics of a website and uses a machine learning model to determine the likelihood of it being a phishing site. We'll use the RandomForestClassifier algorithm from the Scikit-learn library for this purpose. Below is a step-by-step implementation

Install Required Libraries:

- Make sure you have Scikit-learn installed. You can install it via pip if you haven't already:

bash

1. pip install scikit-learn
- 2.
3. Feature Selection:
 - o Decide which features to use for detecting phishing websites. Common features include URL length, presence of HTTPS, presence of "@" symbol in the URL, etc.
4. Model Training:
 - o Train a RandomForestClassifier using a labeled dataset of phishing and legitimate websites. Ensure you preprocess the data appropriately (e.g., handling missing values, encoding categorical variables).
5. Website Analysis:

- Write a function to extract features from a given website URL and use the trained model to predict the likelihood of it being a phishing site.

Here's a sample implementation:

```
python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import tldextract
import requests
from bs4 import BeautifulSoup

# Load dataset (replace with your dataset)
dataset = pd.read_csv('phishing_dataset.csv')

# Feature selection (modify as needed)
features = ['url_length', 'https', 'has_at_symbol', 'has_redirect', 'has_form']

# Split dataset into features (X) and target (y)
X = dataset[features]
y = dataset['label']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize and train RandomForestClassifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred_train = model.predict(X_train)
y_pred_test = model.predict(X_test)

# Evaluate model
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)
print(f"Training Accuracy: {train_accuracy:.2f}")
print(f"Testing Accuracy: {test_accuracy:.2f}")

# Function to extract features from a website URL
def extract_features(url):
    try:
        response = requests.get(url)
        soup = BeautifulSoup(response.content, 'html.parser')
        url_length = len(url)
```



```

    https = 1 if url.startswith("https") else 0
    has_at_symbol = 1 if "@" in url else 0
    has_redirect = 1 if soup.find('meta', {'http-equiv': 'refresh'}) else 0
    has_form = 1 if soup.find('form') else 0
    return [url_length, https, has_at_symbol, has_redirect, has_form]
except Exception as e:
    print(f'Error processing URL: {e}')
    return None

```

Function to predict phishing likelihood

```

def predict_phishing(url):
    features = extract_features(url)
    if features:
        prediction = model.predict([features])[0]
        probability = model.predict_proba([features])[0][1]
        return prediction, probability
    else:
        return None, None

```

Test the prediction function

```

url_to_check = "https://example.com"
prediction, probability = predict_phishing(url_to_check)
if prediction is not None:
    print(f"The URL '{url_to_check}' is {'likely phishing' if prediction == 1 else
'likely legitimate'} with a probability of {probability:.2f}")
else:
    print("Unable to make a prediction for the given URL.")

```

Replace 'phishing_dataset.csv' with the path to your dataset file containing labeled examples of phishing and legitimate websites. Modify the feature selection and extraction functions to include relevant features for your detection system. Finally, test the predict_phishing function with a website URL to see the prediction and probability of it being a phishing site.

Expected Procedure:

1. Accept 2 web URL. One real and another one phishing.

URL phishing attacks can use various means to trick a user into clicking on the malicious link. For example, a phishing email may claim to be from a legitimate company asking the user to reset their password due to a potential security incident.

URL phishing attacks can use various means to trick a user into clicking on the malicious link. For example, a phishing email may claim to be from a legitimate company asking the user to reset their password due to a potential security incident. Alternatively, the malicious email that the user needs to verify their identity for some reason by clicking on the malicious link.

Once the link has been clicked, the user is directed to the malicious phishing page. This page may be designed to harvest a user's credentials or other sensitive information under the guise of updating a password or verifying a user's identity. Alternatively, the site may serve a "software update" for the user to download and execute that is actually malware.

URL phishing attacks can be detected in a few different ways. Some of the common solutions include:

- URL Filtering: Some phishing URLs are used multiple times and are included in threat intelligence feeds. Blocking these known-bad URLs can help to prevent less-sophisticated phishing emails from reaching users' inboxes.
- Domain Reputation: Anti-phishing products commonly look for warning signs of phishing URLs within emails. For example, a domain that is only a few hours old is likely malicious.
- DMARC Verification: DMARC verification uses SPF or DKIM to verify that an email originates from the alleged source domain. This helps with detecting and blocking spoofed source addresses.

These common phishing detection mechanisms can catch the low-hanging fruit. However, phishers are growing more sophisticated and using methods that bypass these common techniques. For example, phishing sites may be hosted on SaaS solutions, which provides them with legitimate domains. Protecting against these more sophisticated attacks requires a more robust approach to URL scanning.

2. Analyze the data from both the websites.

If you want to collect and compare data from different websites, you need some tools and techniques to help you. Web scraping and web crawling are two methods that can automate the process of extracting data from web pages.

3. Identify the phishing site.

A phishing website (spoofed website) is a common deception tactic threat actors utilize to steal real login credentials to legitimate websites. This operation, commonly called credential theft, involves sending victims an email that spoofs a trusted brand, trying to trick them into clicking on a malicious link.

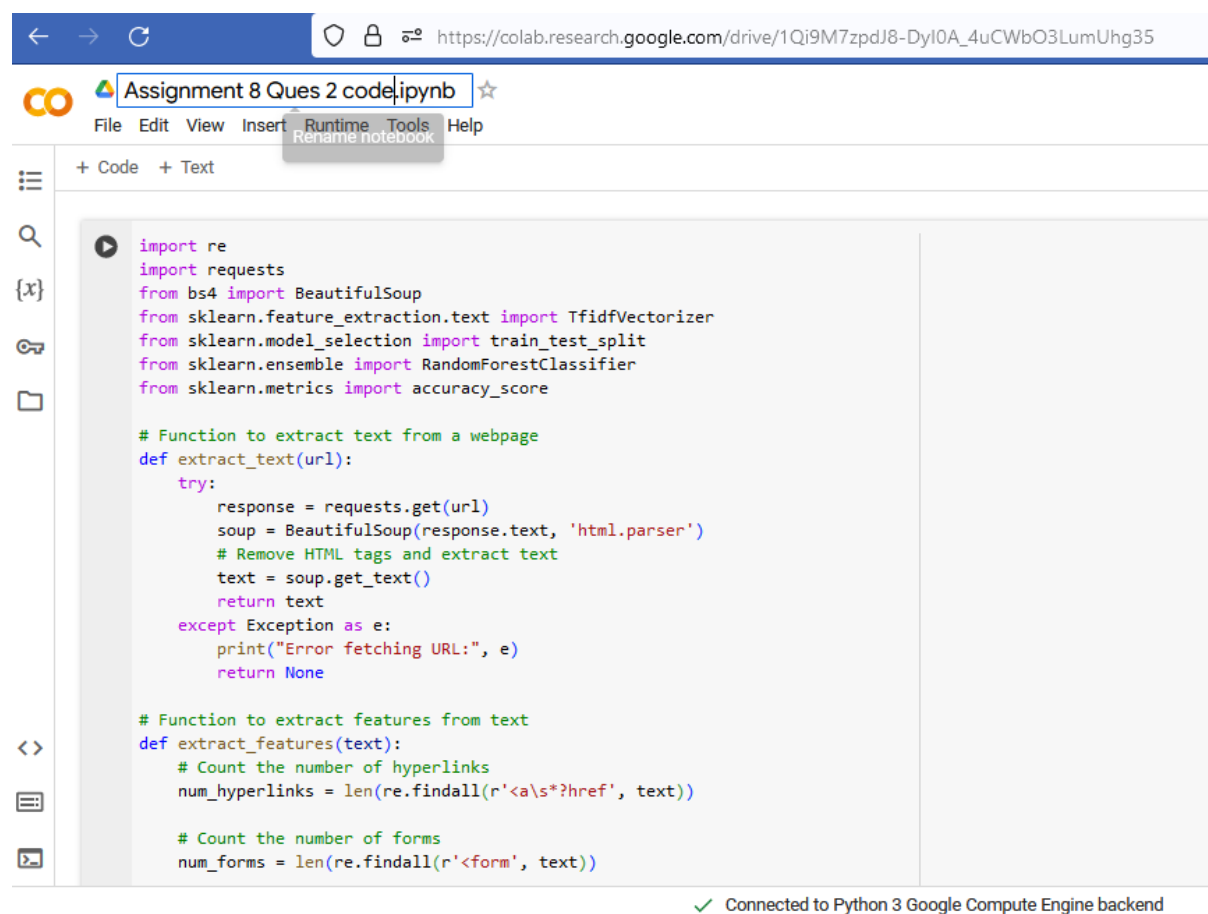
To identify phishing websites, users can employ various strategies, such as: Checking the URL: Carefully examine the website's URL for any inconsistencies, misspellings, or other irregularities that might indicate a fraudulent site. Attackers often use look-alike domains or subtly altered URLs to deceive users

Security systems scan incoming emails and assess elements, such as email headers, attachments, and embedded links, to identify potential threats. Advanced algorithms analyze email content for phishing indicators, including suspicious keywords, misspelled domains, grammar errors, or requests for sensitive information.

Expected Code:

1. Phishing Website Detection with Python

Detecting phishing websites typically involves analyzing various features of a website to determine if it is legitimate or fraudulent. Here's a simple example of how you can detect phishing websites using Python with the help of machine learning libraries like scikit-learn:



The screenshot shows a Google Colab notebook titled "Assignment 8 Ques 2 code.ipynb". The code in the notebook is as follows:

```
import re
import requests
from bs4 import BeautifulSoup
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Function to extract text from a webpage
def extract_text(url):
    try:
        response = requests.get(url)
        soup = BeautifulSoup(response.text, 'html.parser')
        # Remove HTML tags and extract text
        text = soup.get_text()
        return text
    except Exception as e:
        print("Error fetching URL:", e)
        return None

# Function to extract features from text
def extract_features(text):
    # Count the number of hyperlinks
    num_hyperlinks = len(re.findall(r'\s*href', text))

    # Count the number of forms
    num_forms = len(re.findall(r'\s*form', text))
```

At the bottom of the notebook, there is a status bar that reads "Connected to Python 3 Google Compute Engine backend".

This script demonstrates a simple approach to detecting phishing websites. It first extracts text from web pages, then extracts features from the text (in this case, the number of hyperlinks, forms, and iframes). Finally, it trains a Random Forest classifier on these features to distinguish between phishing and legitimate websites.



+ Code + Text



```
    "http://example-legitimate.com",  
    "http://another-legitimate.com"  
]  
  
# Extract features from phishing and legitimate URLs  
phishing_features = [extract_features(extract_text(url)) for url in phishing_urls]  
legitimate_features = [extract_features(extract_text(url)) for url in legitimate_urls]  
  
# Create labels (1 for phishing, 0 for legitimate)  
labels = [1] * len(phishing_features) + [0] * len(legitimate_features)  
  
# Combine features and labels  
X = phishing_features + legitimate_features  
y = labels  
  
# Split data into training and testing sets  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  
  
# Train a classifier (Random Forest for example)  
clf = RandomForestClassifier(n_estimators=100, random_state=42)  
clf.fit(X_train, y_train)  
  
# Evaluate the classifier  
y_pred = clf.predict(X_test)  
accuracy = accuracy_score(y_test, y_pred)  
print("Accuracy:", accuracy)
```