

Application of Python Scripts in Cybersecurity and Using it to Understand Machine Learning

All the codes used here can be downloaded from the following GitHub repository: -

1) Python Script for network port scanning

1. Open nano editor using nano command followed by file name and .py file extension.

`nano Network_Scanner.py`

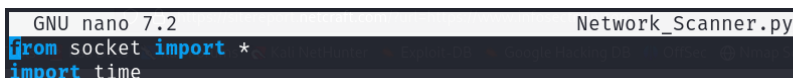


```
(kali㉿kali)-[~]  
$ nano Network_Scanner.py
```

You can begin writing your code here. The entire code is available here:

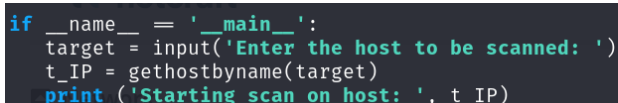
https://github.com/avnishnaithani/pythonforsecurity/blob/main/Network_Scanner.py

2. Import the required libraries



```
GNU nano 7.2 Network_Scanner.py  
from socket import *  
import time
```

3. Next, we will write down the script for taking in user input for hostname/IP address to be scanned.



```
if __name__ == '__main__':  
    target = input('Enter the host to be scanned: ')  
    t_IP = gethostbyname(target)  
    print('Starting scan on host: ', t_IP)
```

4. After this we need to prompt the user to enter a port range. The network will be scanned for open ports between the range provided.

```
print('Enter initial and final port number to define the range for scanning')
a = int(input())
z = int(input())
```

5. The script shown in the screenshot below will perform the network port scanning for the range provided above.

```
for i in range(a,z):
    s = socket(AF_INET, SOCK_STREAM)
    conn = s.connect_ex((t_IP, i))
```

6. We then need to display the open ports. This can be done by entering the script shown in the screenshot below.

```
if(conn == 0) :
    print('Port %d: OPEN' % (i,))
```

7. To display the time taken to conduct network scan

```
print('Time taken:', time.time() - startTime)
```

8. After you have completed entering the entire code, save the file using “CTRL+x” followed by pressing “y” and “Enter” keys.

9. Convert this file to an executable file using the command: -

chmod +x Network_Scanner.py

```
(kali@kali)-[~]
$ chmod +x Network_Scanner.py
```

10. Run the Python program on network port scanning using the command: -

python3 Network_Scanner.py

```
(kali@kali)-[~]
$ python3 Network_Scanner.py
```

11. We can see the output displayed below. We used the IP address of a vulnerable system present in our own network. (Metasploitable2 Virtual Machine downloaded from vulnhub.com).

```
(kali@kali)-[~]
$ python3 Network_Scanner.py
Enter the host to be scanned: 192.168.179.130
Starting scan on host: 192.168.179.130
Enter initial and final port number to define the range for scanning
0
1000
Port 21: OPEN
Port 22: OPEN
Port 23: OPEN
Port 25: OPEN
Port 53: OPEN
Port 80: OPEN
Port 111: OPEN
Port 139: OPEN
Port 445: OPEN
Port 512: OPEN
Port 513: OPEN
Port 514: OPEN
Time taken: 77.46328949928284
```

We can also enter any host name you want to check. We check open port details for Vulnweb.com which is a vulnerable website with open ports.

```
(kali@kali)-[~]
$ python3 Network_Scanner.py
Enter the host to be scanned: vulnweb.com
Starting scan on host: 44.228.249.3
Enter initial and final port number to define the range for scanning
0
1000
Port 80: OPEN
```

2) Developing a Python Script for Malware Detection

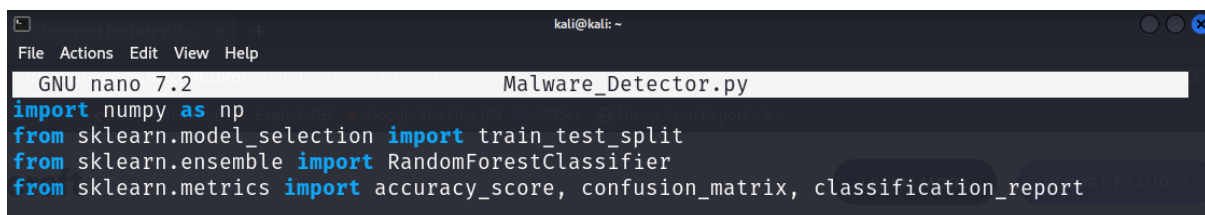
1. Create a file and open it in the Nano editor using the command: -
\$ nano Malware_Detector.py

```
(kali@kali)-[~]
$ nano Malware_Detector.py
```

This will open the nano editor with the mentioned file name and type. This is where you can write down the code. You can obtain this code from

https://github.com/avnishnaithani/pythonforcybersecurity/blob/main/Malware_Detector.py

2.Importing Libraries: Begin by importing the necessary libraries. These contain all the predefined functions which will be required for implementing our model. It also contains the functions which can be used for implementing the machine learning algorithms.



```

GNU nano 7.2 Malware_Detector.py
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
  
```

3. Creating the Dataset: Instead of working on actual malware samples, we will create a dataset to simulate the features of real-world malware issues. These features are: -

file_size: Malware often has different size characteristics compared to benign software.

num_functions: The number of functions in the code, which might differ between malware and legitimate software.

entropy: A measure of randomness in the file, often higher in encrypted or packed malware.

contains_crypto: A binary feature indicating the presence of cryptographic functions, common in some types of malware.

num_strings: The number of string literals in the code, which might have different patterns in malware vs. benign software.

```
# Generate a simple dataset
# Features: file_size, num_functions, entropy, contains_crypto, num_strings
def generate_sample(is_malware):
    if is_malware:
        return [
            np.random.randint(100000, 1000000), # file_size
            np.random.randint(1000, 5000), # num_functions
            np.random.uniform(6.5, 8.0), # entropy
            1, # contains_crypto
            np.random.randint(5000, 10000) # num_strings
        ]
    else:
        return [
            np.random.randint(10000, 500000), # file_size
            np.random.randint(100, 1000), # num_functions
            np.random.uniform(4.0, 6.5), # entropy
            np.random.choice([0, 1], p=[0.8, 0.2]), # contains_crypto
            np.random.randint(1000, 5000) # num_strings
        ]
```

We're creating an equal number of malware and benign samples for a balanced dataset.

```
# Generate dataset
np.random.seed(42)
n_samples = 1000
malware_samples = [generate_sample(True) for _ in range(n_samples // 2)]
benign_samples = [generate_sample(False) for _ in range(n_samples // 2)]

X = np.array(malware_samples + benign_samples)
y = np.array([1] * (n_samples // 2) + [0] * (n_samples // 2))
```

4. Split the dataset: The process of dividing our dataset into training and testing sets is a crucial step in machine learning to assess how well our model generalizes to unseen data.

```
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

5. Train the model: The script below trains our Random Forest classifier (Supervised ML algorithm) on the training data.

```
# Train the model
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
```

6. Make predictions: Here, we use our trained model to make predictions on the test set.

```
# Make predictions
y_pred = clf.predict(X_test)
```

7. Evaluate the model: This section introduces the code that assesses how well our model performed. We use several metrics:

- **Accuracy:** The proportion of correct predictions (both true positives and true negatives) among the total number of cases examined.
- **Confusion Matrix:** A table showing the number of correct and incorrect predictions broken down by each class (malware and benign).
- **Classification Report:** Provides precision, recall, F1-score, and support for each class.

```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)
```

8. Feature importance: This part of the code examines which features the model found most useful in making its predictions. This can provide insights into what characteristics are most indicative of malware in our model.

```
# Feature importance
feature_importance = clf.feature_importances_
features = ['file_size', 'num_functions', 'entropy', 'contains_crypto', 'num_strings']
for feature, importance in zip(features, feature_importance):
    print(f"{feature}: {importance}")
```

9. Test with new samples: Finally, this section demonstrates how to use the trained model on new, unseen data. We create two new

samples (one malware and one benign) and show how the model classifies them.

```
# Test with new samples
new_samples = np.array([
    generate_sample(True),    # A malware sample
    generate_sample(False)   # A benign sample
])

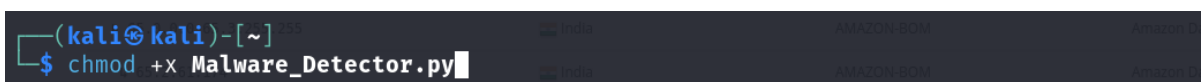
predictions = clf.predict(new_samples)
print("\nPredictions for new samples:")
for sample, prediction in zip(new_samples, predictions):
    print(f"Sample: {sample}")
    print(f"Prediction: {'Malware' if prediction == 1 else 'Benign'}\n")
```

The code provides a working example, but in a real-world scenario, you would need much more complex and diverse features, a larger and real dataset, and you might need to try various models and tuning techniques to achieve better performance. Additionally, you'd need to regularly update your model as new types of malwares emerge.

10. Save this file using "CTRL+x" and then press "y" for yes followed by "Enter".

11. Convert the file to an executable one using the command: -

\$ chmod +x Malware_Detector.py



12. Finally execute the program. Use the command mentioned below for this: -

\$ python3 Malware_Detector.py

The output obtained is shown below: -

```
(kali@kali)-[~]
$ python3 Malware_Detector.py
Accuracy: 1.0

Confusion Matrix:
[[104  0]
 [ 0 96]]

Classification Report:

```

	precision	recall	f1-score	support
accuracy	1.00	1.00	1.00	200
macro avg	1.00	1.00	1.00	200
weighted avg	1.00	1.00	1.00	200

```

file_size: 0.0005925915865802228
num_functions: 0.2828936939812229
entropy: 0.28591168126456734
contains_crypto: 0.07419495260104936
num_strings: 0.35640708056658016

Predictions for new samples:
Sample: [7.47418000e+05 2.78800000e+03 7.11217859e+00 1.00000000e+00
 8.70000000e+03]
Prediction: Malware

Sample: [4.33914000e+05 5.15000000e+02 5.78660319e+00 1.00000000e+00
 4.89300000e+03]
Prediction: Benign

```

Confusion Matrix

- 104 samples were correctly classified as benign (True Negatives)
- 0 benign samples were incorrectly classified as malware (False Positives)
- 0 malware samples were incorrectly classified as benign (False Negatives)
- 96 samples were correctly classified as malware (True Positives)

This matrix shows perfect classification, as there are no misclassifications.

Classification Report

Precision: The ratio of correctly predicted positive samples to the total predicted positive samples.

- For both classes (0 and 1), precision is 1.00, meaning 100% of predictions for each class were correct.

Recall: The ratio of correctly predicted positive samples to all samples in the actual class.

- For both classes, recall is 1.00, meaning 100% of actual samples in each class were correctly identified.

F1-score: The harmonic mean of precision and recall, providing a single score that balances both metrics.

- F1-score is 1.00 for both classes, indicating perfect balance between precision and recall.

Support: The number of samples for each class in the test set.

- 104 samples for class 0 (benign)
- 96 samples for class 1 (malware)

Accuracy: The ratio of correctly predicted samples to the total samples.

- Overall accuracy is 1.00, or 100%, meaning all samples were correctly classified.

Macro avg: The unweighted mean of the metrics for each class.

Weighted avg: The weighted mean of the metrics for each class, considering the number of samples in each class.

```

kali@kali: ~
File Actions Edit View Help
(kali@kali)~$ python NLP_Security_Logs.py
[nltk_data] Downloading package punkt to /home/kali/nltk_data ...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /home/kali/nltk_data ...
[nltk_data] Package stopwords is already up-to-date!
Extracting IP addresses:
['192.168.1.100', '203.0.113.42', '192.168.1.100', '198.51.100.77', '203.0.113.42']

Extracting usernames:
['admin', 'john', 'account']

Most common events:
ip: 5
login: 4
failed: 3
user: 3
attempt: 2

Clustering logs:
Log 1 - Cluster 2: 2023-10-01 08:30:15 Failed login attempt from IP 192.168.1.100
Log 2 - Cluster 1: 2023-10-01 09:15:22 Successful login by user admin
Log 3 - Cluster 2: 2023-10-01 10:45:30 Firewall blocked connection from IP 203.0.113.42
Log 4 - Cluster 1: 2023-10-01 11:20:18 File deletion in /sensitive/data by user john
Log 5 - Cluster 2: 2023-10-01 12:05:45 Multiple failed login attempts from IP 192.168.1.100
Log 6 - Cluster 0: 2023-10-01 13:10:33 Unusual outbound traffic detected to IP 198.51.100.7
7
Log 7 - Cluster 0: 2023-10-01 14:30:27 System update installed successfully
Log 8 - Cluster 1: 2023-10-01 15:45:12 New user account created: alice
Log 9 - Cluster 1: 2023-10-01 16:20:55 Suspicious file quarantined: malware.exe
Log 10 - Cluster 2: 2023-10-01 17:05:40 Failed login attempt from IP 203.0.113.42

Potential security insights:
Multiple events from IP 192.168.1.100
Multiple events from IP 203.0.113.42
Failed login attempts detected
Suspicious activity detected

(kali@kali)~$

```