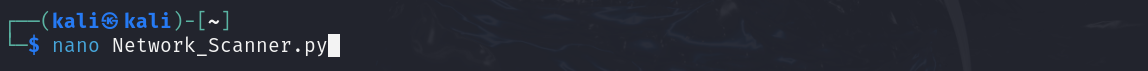
**Application of Python Scripts in Cybersecurity and Using it to Understand Machine Learning**

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

**1) Python Script for network port scanning**

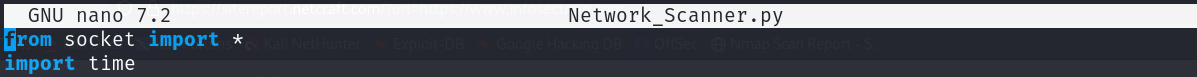
1. Open nano editor using nano command followed by file name and .py file extension.

**nano Network\_Scanner.py**

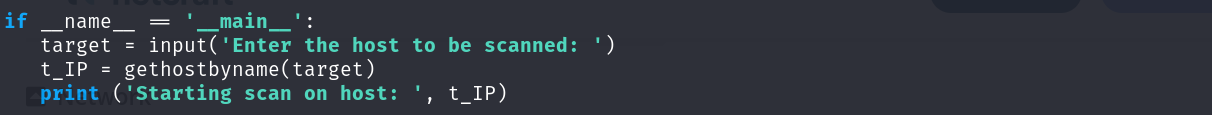


You can begin writing your code here. The entire code is available here: <https://github.com/avnishnaithani/pythonforcybersecurity/blob/main/Network_Scanner.py>

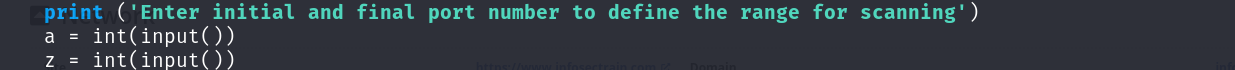
2. Import the required libraries



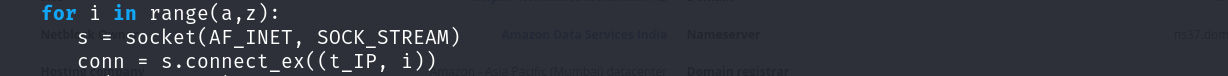
3. Next, we will write down the script for taking in user input for hostname/IP address to be scanned.



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.



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



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



7. To display the time taken to conduct network scan



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**

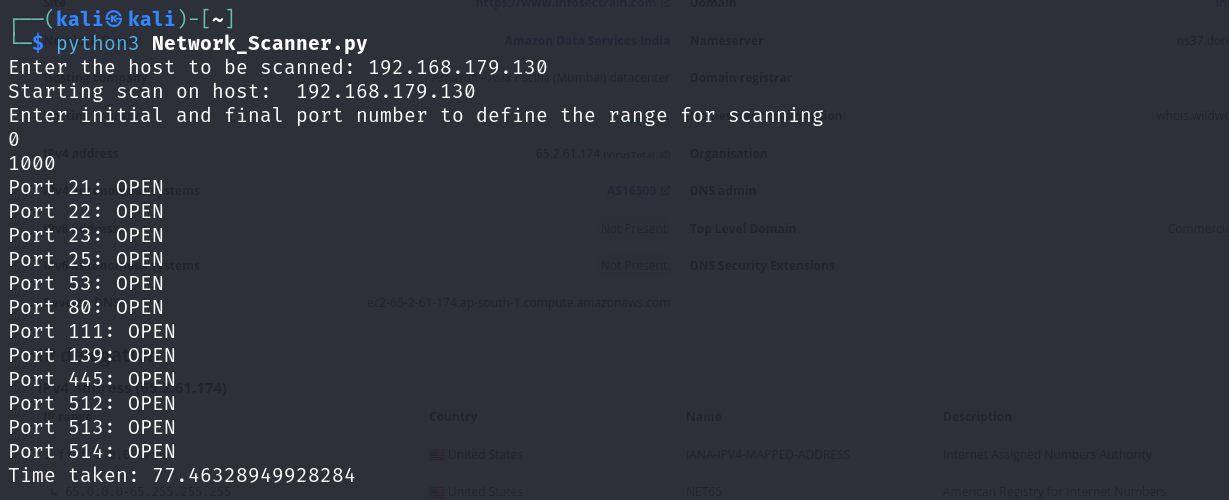


10. Run the Python program on network port scanning using the command: -

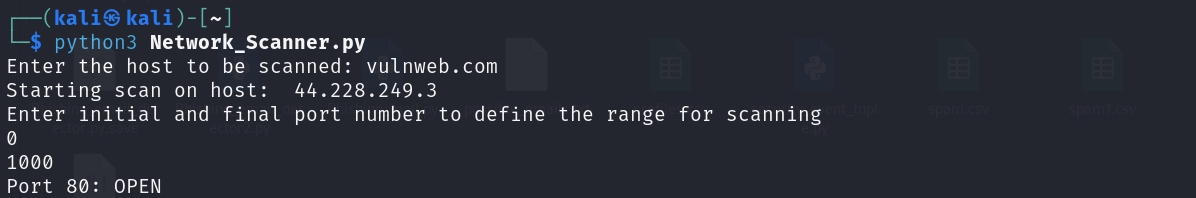
**python3 Network\_Scanner.py**



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).



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.



**2) Developing a Python Script for Malware Detection**

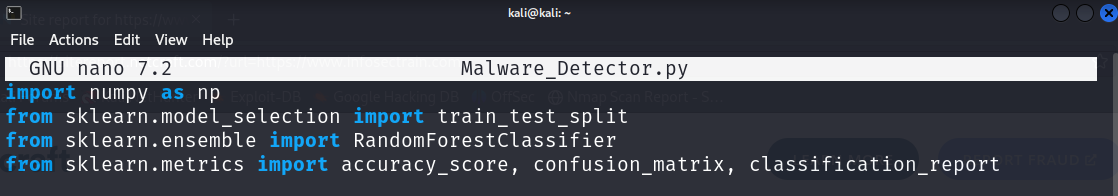
**1. Create a file** and open it in the Nano editor using the command: -

$ **nano Malware\_Detector.py**

****

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.



**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: -

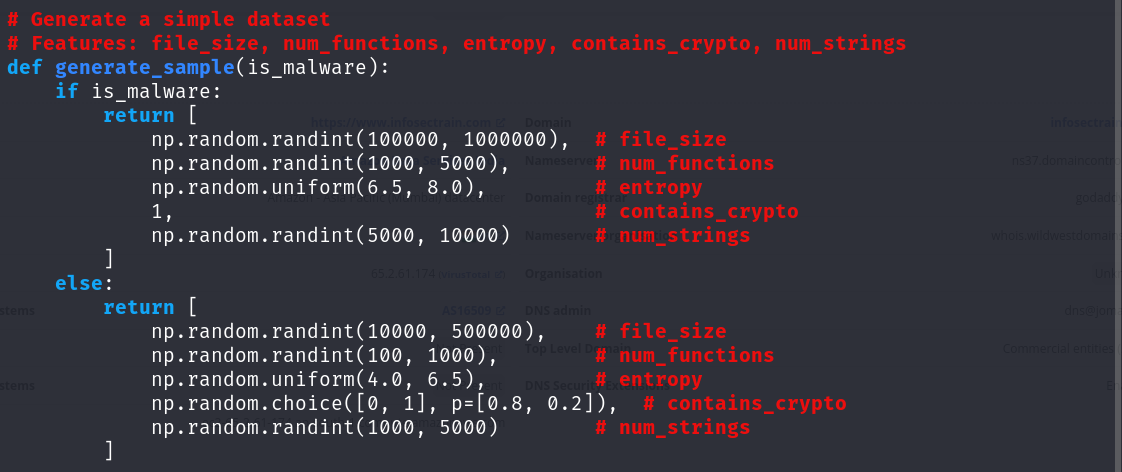
file\_size: Malware often has different size characteristics compared to benign software.

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

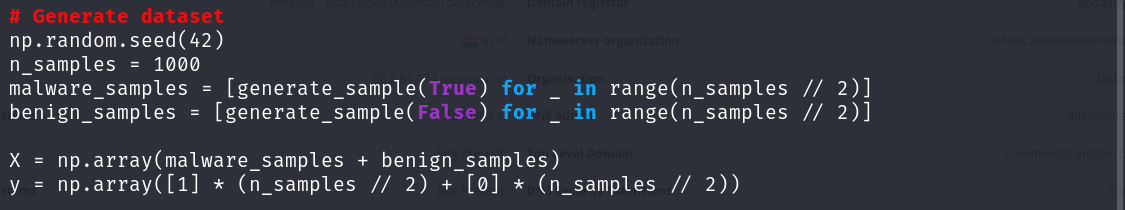
entropy: A measure of randomness in the file, often higher in encrypted or packed malware.

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.



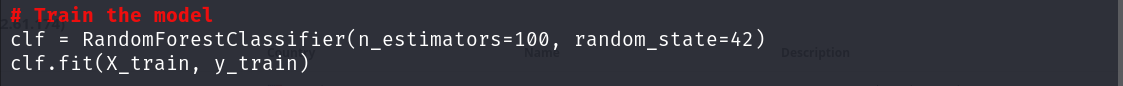
We're creating an equal number of malware and benign samples for a balanced dataset.



**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.



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

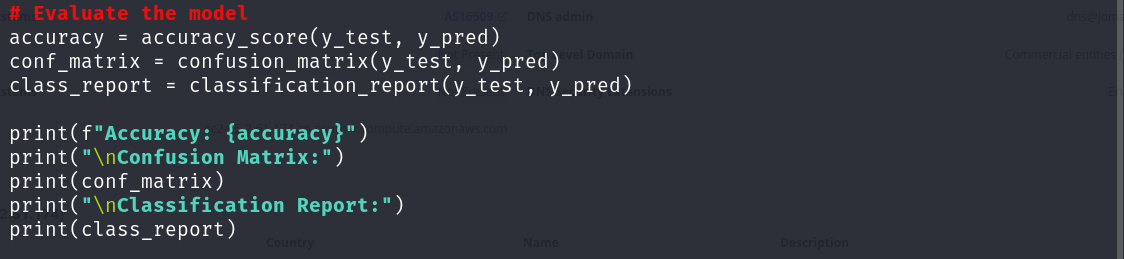


**6. Make predictions:** Here, we use our trained model to make predictions on the test set.



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

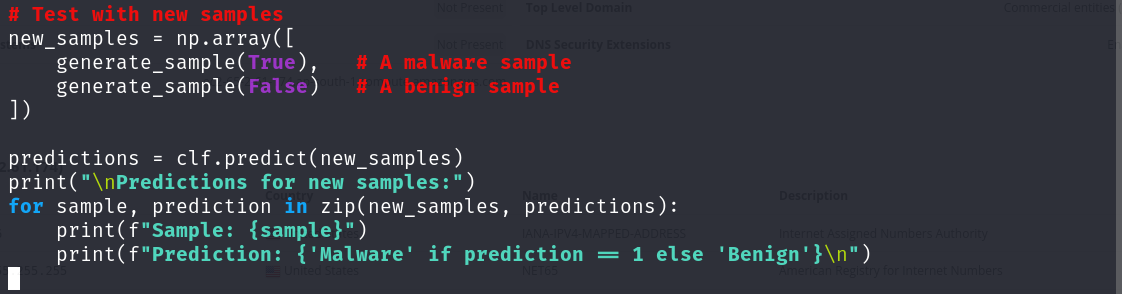
* Accuracy: The proportion of correct predictions (both true positives and true negatives) among the total number of cases examined.
* Confusion Matrix: A table showing the number of correct and incorrect predictions broken down by each class (malware and benign).
* Classification Report: Provides precision,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



**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.



**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.

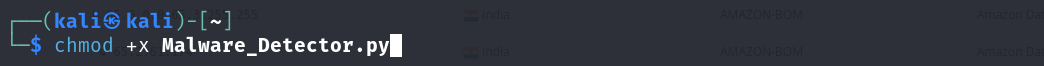


The code provides a working example, but in a real-world scenario, you would need much more complex and diverse features, a larger and real dataset, and you might need to try various models and tuning techniques to achieve better performance. Additionally, you'd need to regularly update your model as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

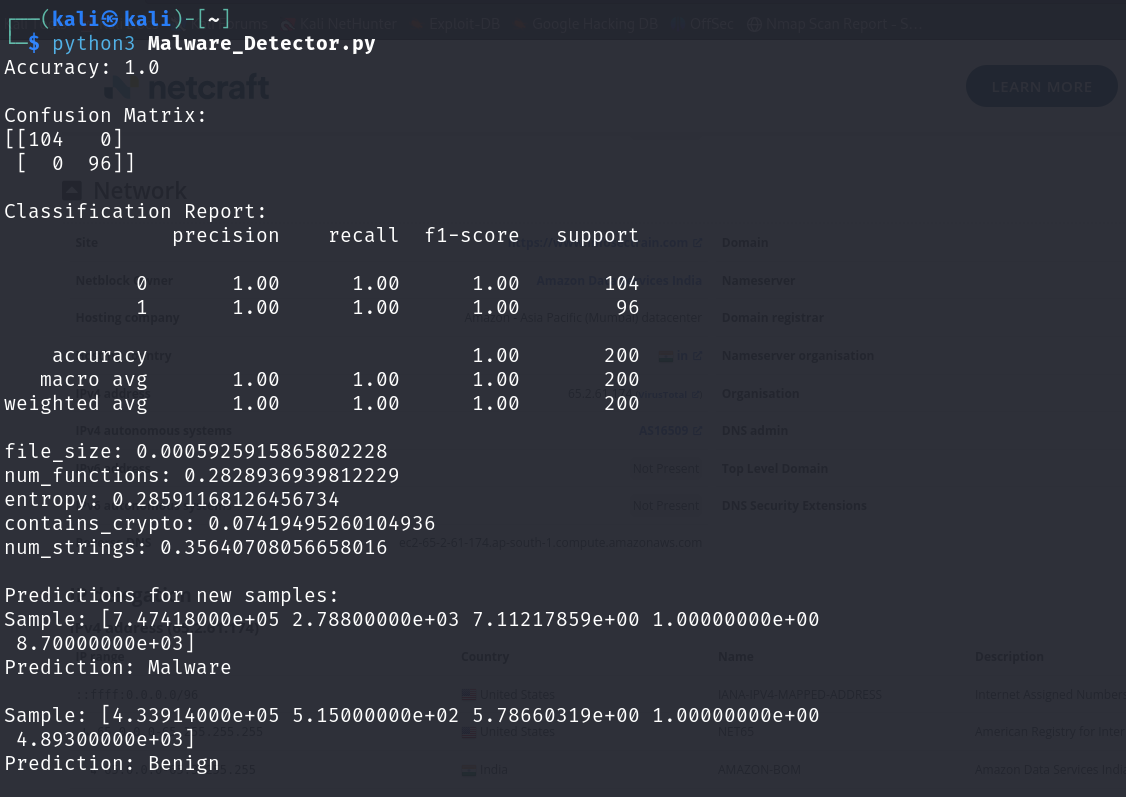
$ **chmod +x Malware\_Detector.py**

****

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

$ **python3 Malware\_Detector.py**

The output obtained is shown below: -

****

**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.

****