Deep Learning Analysis of System Call Timing for Rootkit Detection

By

Vibhuti Aggarwal ( 18BCE0757 )

Sarthak Pranesh ( 18BCE0814 )

Abstract →

In today’s era of viruses, malware and cyber attacks, rootkits pose the most dangerous threat to personal computers as well as computers working as main frame in big companies like Google.

A rootkit is a collection of programs or program that allows a threat actor, remote access to another computer of other systems. Most rootkits open a backdoor on victim systems to introduce malicious software, such as viruses, ransomware, keylogger programs or other types of malware.

They also try to prevent any sort of detection of malicious software software and also manipulate system software to stay undetected. Therefore the purpose of this project is to find out techniques and ways to detect these rootkits using different techniques of machine learning and deep learning.

Different types of rootkits will be detected through the means of a trained model which will ultimately classify a process/program to be a rootkit or not

Key Words →

* Rootkits
* Cyber Attacks ( Backdoor )
* Reptile ( open source rootkit )
* Machine learning
* Decision Tree Algorithm
* Random Forest
* Support Vector Algorithm ( Gaussian Kernel )
* Deep learning

Introduction →

In this project the team focuses mainly on the analysis of using different techniques in Artificial Intelligence to classify a computer as infected or not infected by a rootkit. In today's era the threat posed by rootkits in enormous hence successful detection of these rootkits is a necessity for the security of many big companies and even for individuals. A rootkit in general opens a backdoor on the victim’s computer with root privileges without getting detected and hence allows a remote hacker to execute root level commands on the victim's machine without their knowledge.

The project holds a general overview of what rootkits are and various methods used in designing rootkits. The project also talks about an open source rootkit that can be used to infect a system ( only for demonstration and data extraction ). A brief survey about the risks posed by the rootkits and the current methods used in detection and defence against such rootkits are also discussed in the project.

Further the project discusses different techniques in AI like Decision Tree, Random Forest and different Deep learning algorithms to detect these rootkits in a computer and mark them as infected or not infected. The dataset collected for training these models where collected from a Virtual Machine Simulation of Elementary OS, once while the machine was not infected and once when it was infected ( further details on this also discussed ).

AI is used because of the numerous advantages that it provides in this topic, which are recognising data collected ( system call data of infected and not infected machines ) which has patterns, the algorithm can automatically adapt in future to recognise new

rootkits previously unknown to the algorithm, the efficiency provided in detecting rootkits can be scaled up to 100%, etc.

The only disadvantage seen for this approach in the modern era is the collection of data

From different systems. As no defined dataset for the purpose exists, for research in this field a whole new dataset needs to be complied with proper collection of each value from the system or the models will train poorly.

Finally a conclusion is derived out of all the techniques is used and future research on the topic is stated.

Existing Techniques and Analysis →

In the case study of our base paper the author collected system calls and their corresponding execution time on an operating system. Then they infected the system with a rootkit, and collected the system call information of the infected operating system.

For their analysis they used the Kbeast rootkit. They ran a total of six collections out which four recorded uninfected system calls and the remaining two recorded the infected ones. They assigned a value of zero for the infected and one for the uninfected system calls.The data set was shuffled and then it was divided into three sub datasets for their analysis.

The software they used for their analysis was MATLAB and the MATLAB Neural Network Toolbox. Their experiment involved two network architectures. The first was a static feed-forward architecture and the second was a recurrent nonlinear auto-regressive architecture. Both their architectures used the Levenberg Marquardt algorithm for training their dataset. The error function used was Mean Square Error function.

The first feed forward network consisted of two hidden layers containing five and three neurons respectively, was able to provide the accuracy upto 82.8% when 70% of the dataset was used to train the model .

The second network tested was the recurrent network. It again had two layers with five neurons each. The tested accuracy for the recurrent network was 95.9%.

The author mentions the inefficiency of the dataset and suggests the use of parallel computers to accomodate large datasets. He further suggests testing of other machine learning algorithms such as decision tree method, support vector machine and unsupervised methods such as random forest.

Problem Statement:

Implement the some machine learning algorithms to classify systems calls as infected and uninfected and thereby detecting the presence of rootkit in the operating system.

Objectives:

→ Analyse whether the operating system is infected with a rootkit or not.

→ Determine the most suitable machine learning algorithm.

→ Analyse the accuracy of each algorithm and its efficiency in this domain.

Techniques and Algorithms:

1. Dataset Collection

Our data set included the name of the system call and its execution time . To collect the data we created a virtual machine and first collected the execution time for 200 system calls . These system calls were marked zero. Then we infected the operating system with a custom made rootkit named Reptile. Again around 200 system calls were recorded and were marked as one.

1. Training the dataset

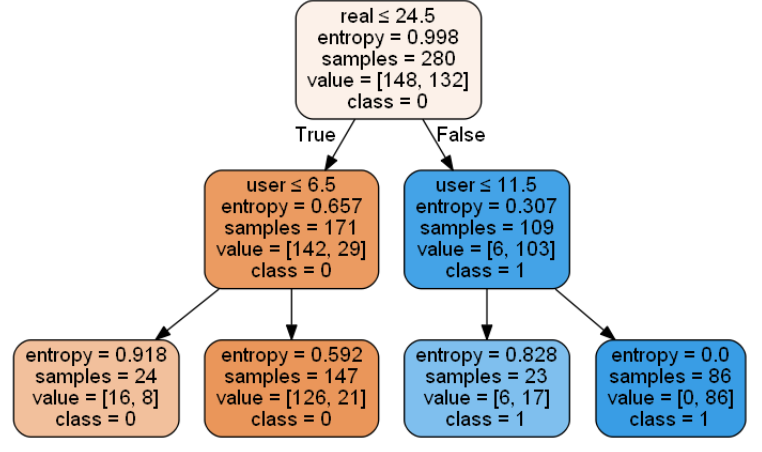
The language we used for our analysis is Python . Our experiment involved the following algorithms

→ Decision tree algorithm

Decision tree algorithm falls under the category of supervised learning. Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree. We can represent any boolean function on discrete attributes using the decision tree.

Below are some assumptions that we made while using decision tree:

* At the beginning, we consider the whole training set as the root.
* Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
* On the basis of attribute values records are distributed recursively.

In this model we use the sklearn library to split the data into training and testing data,to calculate the accuracy of the model and to get the confusion metrics. We use the DecisionTreeClassifier to build our model. We used 70% of our dataset to train the model and the rest 30% was used for testing. This model classified the dataset with an accuracy of 87-90%. The decision tree is as follows:

→ Random Forest Algorithm:-

The random forest is a model made up of many decision trees. It is also a supervised learning model. Rather than just simply averaging the prediction of trees (which we could call a “forest”), this model uses two key concepts that gives it the name *random*:

1. Random sampling of training data points when building trees
2. Random subsets of features considered when splitting node.

The random forest combines hundreds or thousands of decision trees, trains each one on a slightly different set of observations, splitting nodes in each tree considering a limited number of features. The final predictions of the random forest are made by averaging the predictions of each individual tree.

In this model we use the sklearn library to split the data into training and testing data,to calculate the accuracy of the model and to get the confusion metrics. We use the RandomForestClassifier to build our model. We used 70% of our dataset to train the model and the rest 30% was used for testing. This model classified the dataset with an accuracy of 85-88%.

→ Simple Vector Machine Algorithm

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.

SVM has a technique called the kernal trick. These are functions which takes low dimensional input space and transform it to a higher dimensional space i.e. it converts not separable problem to separable problem, these functions are called kernels. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then find out the process to separate the data based on the labels or outputs you’ve defined.

In this model we use the sklearn library to split the data into training and testing data,to calculate the accuracy of the model and to get the confusion metrics. We used the Gaussian kernel in this model which provided the best results. We used 70% of our dataset to train the model and the rest 30% was used for testing. This model classified the dataset with an accuracy of 93-96%.

→ Deep learning Model

It is a subset of machine learning that incorporates learning through non - linear way and uses the hierarchical structure to do so. The model consists of layers of neural network formed of neurons. The layer is made a Dense layer which simply means that the layer consists of classic neuron layer which is fully connected to the layer before it and the layer head it. The use of the Dropout layer is also analysed, which is basically a layer which sets some node or neurons to have zero as activation function.

A multi-layer model was formed and trained on the training set. The initial model made had following structure :-

Dense input layer - number of neurons = 3 - input dim = 3

Dense hidden layer - number of neurons = 3

Dense hidden layer - number of neurons = 3

Dense hidden layer - number of neurons = 5

Dense output layer - number of neurons = 1

The model was compiled with - loss = mean square error, optimizer = adam, metrics = accuracy

The model was trained on 150 epochs on a batch size of 10. With this configuration the model gave 88.4% accuracy.

Accuracy can be further enhanced by trial and error method and by modifying the model structure and layers.

Comparison

As the author of the base paper suggested we tested the different algorithms and got the following outputs :-

|  |  |
| --- | --- |
| Algorithm | Accuracy |
| 1. Decision Tree | 87 - 90 % |
| 1. Random forest | 85 - 88 % |
| 1. SVM | 93 - 96 % |

Comparison of deep learning model :- The existing techniques which had two hidden layers containing five and three neurons respectively, was able to predict and gave 82.8% at 70% of dataset was used to train the dataset. Where as our existing model gave a better accuracy of about 88.4% of accuracy.The future work stated by the base paper was analysed and it was found that deep learning algorithm was the most extensible among all of them as it can be mended and optimised to learn the dataset better and the initial model itself gave decent results in comparison with other models.

The second network tested by the base paper was the RNN which provided them 95% accuracy, hence the next model going to be tested is the LSTM which is a more advanced version of RNN.

Comparative Analysis Of Existing Solutions :-

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name of the Paper | Year of publishing | Problem addressed | Proposed technique | Future work |
| Detecting kernel level rootkits using binary analysis | 2004 | Determining if the system is infected or not in kernel level | Binary Analysis | Using System call for detection of kernel level rootkits |
| Countering persistent kernel rootkits through systematic hook discovery | 2008 | Detecting hooks in the system and protection against such hooks | Detection using system administrator and anti-malware software | Identify and defend hooks on kernel level |
| Guest transparent prevention of kernel rootkits | 2008 | Prevent unauthorised code execution for guest operating system | Using a virtual machine monitor to establish a kernel module that executes as an extension of linux operating system to detect rootkits | to expand the coverage of protected data static analysis of the kernel source code analysis to capture dynamic runtime memory man-agement information |
| Rootkit detection on virtual machines through deep information extraction at hypervisor level | 2013 | Detecting Virtual machine based rootkits | Cross verification of components in a virtual machine | introduce intelligence into the construction of and anomaly detection |

Conclusion :-

This project report shows how various supervised machine learning algorithms can be used in detection of the rootkits at kernel level. The techniques used here where all included in the supervised machine learning concepts only. Future work related to the project includes the unsupervised and reinforced learning where the decision of classifying a PC is infected or not can be left upon the algorithm itself. Hence algorithms like kNN and other clustering algorithms.

Future work may also include more parameters then the current work to provide more entities on the bases of which the algorithm can classify the pc as infected or not infected more accurately.

These parameters may include the change in CPU power consumption and other parameters. More complex Deep learning networks can be used for the purpose of classification too.

References

1. Neural Network Analysis of System Call Timing for rootkit detection

* Patrick Luckett, J. Todd McDonald, Joel Dawason -- Cybersecurity Symposium 2016

1. Detecting kernel level rootkits using binary analysis

* Kruegel, Christopher, William Robertson, Giovanni Vigna -- Computer Security Application Conference 2004, 20th Annual IEEE

1. Guest-transparent prevention of kernel rootkits with vmm-based memory shadowing

* Riley, Ryan, Xuxian Jiang, and Dongyan Xu -- RecentAdvances in Intrusion Detection. Springer Berlin Heidelberg, 2008.

1. Countering persistent kernel rootkits through system-atic hook discovery

* Wang, Zhi, et al -- Recent Advances in Intrusion Detection. SpringerBerlin Heidelberg, 2008.

1. Rootkit detection on virtual machines through deep information extraction at hypervisor-level

* Xie, Xiongwei, and Weichao Wang -- Com-munications and Network Security (CNS), 2013 IEEE Conference on.IEEE, 2013.