CyberSec Simulator

Sarthak Joshi
170020040
Indian Institute of Technology
Dharwad, India
170020040@iitdh.ac.in

Sonu Sourav
170020021
Indian Institute of Technology
Dharwad, India
170020021@iitdh.ac.in

I. Introduction

The objective of this project is to simulate and detect cyber attacks on a node in a network. We shall simulate a UDP flood attack conducted using botnets. We shall detect this attack using stream mining algorithms.

II. Cyber Attacks and Botnets

Cyber attacks have endlessly evolved over the decades as the use of computer networks has become the standard all over the world. One of the simplest of these attacks is an UDP flood attack. The default response on receiving an UDP packet on a port in which no application is expecting to receive it is to reply with an ICMP response to notify the sender of the failed transmission. However, it is possible to exploit this procedure by flooding the target address with UDP packets at an extremely high rate. The target is overwhelmed by the sheer amount of UDP packets that need to responded to. This causes the target to either lag heavily or even crash. In the past, this attack was done by one or a small number of devices on a target. Therefore, it was easy to detect the attack simply by checking the rate of packet arrival. However, with the advent of botnets, this task has become much more complicated.

A botnet is a number of Internet-connected devices, each of which is running one or more bots. Botnets can be used to perform Distributed Denial-of-Service (DDoS) attacks, steal data, send spam, and allow the attacker to access the device and its connection. A number of simple devices all over our homes like cameras, printers, etc. already have the functionality of communicating over a network. Many of these are easily compromised as people generally don't change the default passwords on these devices. As a result, modern hackers are easily able to assemble an army of bots over a network that can be used to attack the target. As long as the number of bots is large enough, even a slow packet transmission rate can flood the target and overwhelm it.

III. Simulating the Attack

In our project we assume that the attacker already has the authentication credentials to a number of addresses (addresses of bots).

A. Finding active bots

We first send a simple request to each of the bot addresses and wait for a response. The addresses which reply are the bots which are currently running and can be used for an attack. We place all these addresses in a list.

B. Preparing the bots

Once we have a list of active bots we access each of them remotely and copy our attacking script to their local storage. As we already have the authentication credentials this can be performed quickly.

C. Attacking

After copying the script, we remotely start it up in each of the bots. The attacking script simply floods the target with UDP packets for 60 seconds.

IV. Stream Mining

Due to the nature of botnets, it is very difficult to detect these attacks by through static algorithm. Furthermore, even using machine learning classification algorithms does not yield good results as the nature of the network continuously changing. A node on an address may suddenly change its functionality which requires it to respond to many requests which may now be considered attacks by the classification model as in the past, the node responded to much less requests. Therefore there is a need for a classification model that trains for the recent network changes while determining whether the address is under attack at the same time.

Data Stream Mining (also known as stream learning) is the process of extracting knowledge structures from continuous, rapid data records. A data stream is an ordered sequence of instances that in many applications of data stream mining can be read only once or a small number of times using limited computing and storage capabilities. We can consider the packets we receive as a single data stream and analyze them to classify the state of the node as being under attack or not. This is performed using a small number of recently received packets and the model is trained in real-time.

A. Decision Trees

Decision tree is a type of classification model that divides the input data set into various categories based on its attributes. The classifications divide the input set further in

essentially a tree hierarchy as shown Figure 1.

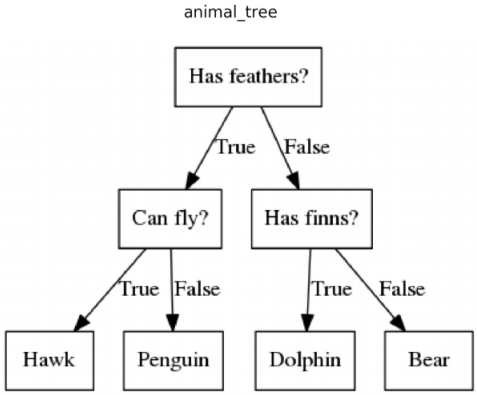

Figure 1: Example of a decision tree

We shall be using decision trees for classification in this project. The attributes we shall be using are as follows:

1) Duration of the flow.
2) Type of Protocol.
3) Connection source IP address and port.
4) Connection destination IP address and port.
5) Direction of the flow
6) Number of packets in the flow.
7) Number of bytes in the flow.
8) Flags

B. Hoeffding Decision Tree

The Hoeffding decision tree is a Very Fast Decision Tree (VFDT) that we shall be utilizing for our project. It is based on the concept of the Hoeffding inequality that states:

$$\begin{aligned} &Probability\ of\ mistake = \\ &Pr(|expected_{value} - true_{value}| \geq \epsilon) \\ &\leq 2^{-2N\epsilon^2} \leq \delta \end{aligned}$$

This is where ϵ is the amount of error that is acceptable after N.
Furthermore:

$$\epsilon = \sqrt{\frac{R^2 \ln \frac{1}{\delta}}{2N}}$$

Where R is the range of the random variable and δ is the desired probability of the estimate not being within ϵ of its expected value.

If the value of the best Gini index out of a group of attributes (at any node in the tree) exceeds the second best by ϵ, we add 2 new nodes in the tree as the children of this node that classify further based on those attributes. Gini index of a random variable C is another nonlinear measure of the dispersion of C, defined as:

$$G(C) = \sum_{c} p_c(1 - p_c) = \sum_{c \neq c'} p_c p_{c'}$$

The Hoeffding inequality helps classify the input set when the classification would cause a much purer set (less variation in output) to form. Furthermore, due to its fast learning capabilities, a high accuracy output can be achieved in a much lower time.

V. Our Contribution

At this stage of the project, our attacking script is working correctly. We have also written code to check whether the bot IPs are running and to transfer the script to their local storage and execute it remotely. However, as we currently do not have access to multiple systems on a network, we are not able to test it.

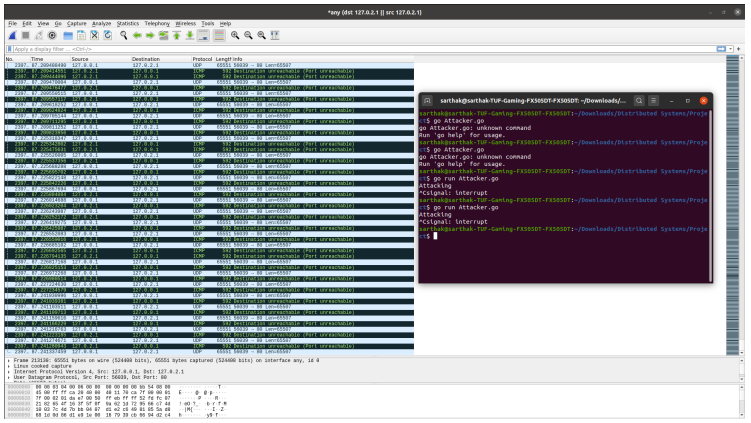

Figure 2: Simulation of a UDP flood attack

A. Hoeffding Tree Classification

We have setup a Hoeffding Tree with the following parameters:

1) Grace Period = 50 = The number of instances a leaf should observe between split attempts
2) Hoeffding Tie Threshold = 0.05 = Threshold below which a split will be forced to break ties
3) Split Confidence = 0.0001 = The allowable error in a split decision

We tested this using the publicly available CTU-13 database of botnet attacks. The dataset divides the trasmissions into labels LEGITIMATE, BACKGROUND and BOTNET. We simulated the dataset as a stream of incoming data that we updated the classification tree with instead of performing the classical train-test split. This would allow us to test the performance of the classification as it would operate in a live environment. We started our making our predictions just after the first 1000 inputs. We observed a consistent 95-98 percent accuracy in the output feed.

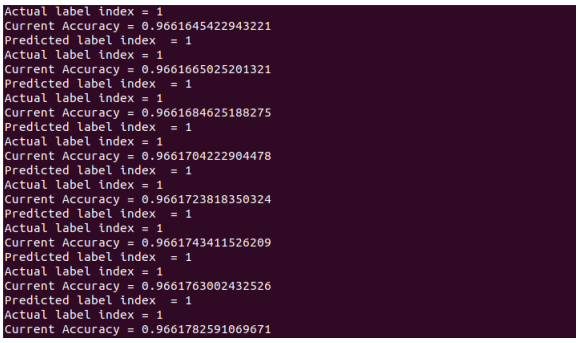

Figure 3: Output feed of data stream fed to Hoeffding tree

VI. Future Work

Some of the future work that can be done in this project would be to test our classification model with the completed attack simulation. Furthermore, some improvements can also be made to the classification tree to overcome some of its shortcomings. The Hoeffding decision tree we have implemented currently lacks the mechanism to recombine its leaves into a higher level node to account for changes in the data stream over time. Therefore, a case of overfitting is likely in an actual deployment. Furthermore, it is also viable to add a functionality to update the Hoeffding tree parameters like the grace period, etc. dynamically depending on the nature of the input stream.

Acknowledgment

We would like to thank Prof. Kedar Vithal for introducing us to this interesting project and providing us with some resources from where we could start.

References

[1] https://www.cms.waikato.ac.nz/ abifet/book/chapter6.html
[2] http://www.otnira.com/2013/03/28/hoeffding-tree-for-streaming-classification/
[3] Guilherme Henrique Ribeiro, Elaine R. de Faria Paiva, Rodrigo Sanches Miani, A Comparison of Stream Mining Algorithms on Botnet Detection