Kevin Yeun and Samuel Xu
CS194-16 Data Science
University of California, Berkeley
kevinyeun@berkeley.edu
skx@berkeley.edu

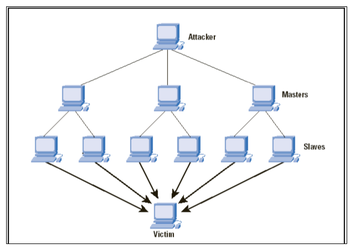
DDoS Classification

Banana is a machine learning project for classifying known Distributed Denial of Service (DDoS) attacks. Unlike other previous classification methods, Banana does not require direct access to packet traces and instead utilizes easy-to-obtain network statistics.

- Banana currently supports classifying some of the most commonly used DDoS attacks like UDP Flooding and TCP SYN Flooding.
- Banana can be easily modified to train for other attacks that are not yet included.
- Banana provides data visualization through the powerful d3.js library.
- Banana code can be run in any environment supporting python.

An introduction to DDoS

A distributed denial of service (DDoS) attack is an attempt to make an online service unavailable by overwhelming it with traffic from multiple sources. A common subtype of DDoS is a flood attack, where a distributed network of attackers will flood the victim server with packets, thus completely consuming its resources. When these attacks first emerged, they were considered minor annoyances. However, they have since grown immensely in size, complexity, maliciousness, and availability.



DDoS diagram by Cisco

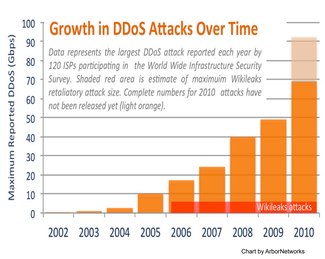


Chart by ArborNetworks

Well-known DDoS Attacks

Banana currently classifies some of the most well-known DDcoS attacks. Here is an explanation of what the attacks are:

SYN Flood – this attack takes advantage of the 3-way handshake that initiates a TCP connection: to establish a connection, the client sends a TCP SYN request, which is followed by the server responding with a SYN ACK acknowledgement, and finally followed by the client acknowledging the SYN ACK. The attacker will spam these handshakes without completing the 3rd step, leaving the connection open & consuming server resources, so it will be unable to accept new connections.

TCP PUSH ACK – this attack floods the victim with TCP packets with the PSH and ACK flags set. These packets instruct the victim to immediately unload the buffer (even if it isn't full) and send an acknowledgement upon completion. Using a distributed network, an attacker can easily deplete the victim server's resources.

Smurf Attack – this is an amplification attack that floods the victim with Internet Control Message Protocol (ICMP) "echo-reply" packets. The attacker forges "echo-request" packets that have the victim as the source IP and sends these to networks where every machine will reply to the victim, thus overwhelming it.

UDP Flood – in this attack, the attacker sends a large number of UDP packets with random ports to the victim. For each packet, the victim checks for an application listening on the port, which there probably won't be, and sends an ICMP destination unreachable packet. This attack relies on the combined attacker network having a greater capacity than the victim.

Problem Statement & Use Case

DDoS is a real threat to network security, and all web applications should be prepared in defending against such attacks. Given the state of a server, there has been much research on determining whether or not it is under attack. However, a more practical piece of information is to find out what the attack consists of, if there is an attack. The first step in defending against a DDoS attack involves identifying the attackers and the vectors they are attacking. By using Banana, we can classify the network flow into a major attack, allowing us to pinpoint where or how to mitigate the attack.

Example Use Case:

- Your web application is not available to customers. Under closer inspection you see an abnormally large amount of traffic, but don't know how to mitigate it. You feed network statistics to Banana, and find that the attack is a UDP Flood. You can mitigate the attack by limiting the ICMP responses your server sends out.

Performance Metrics & Evaluation

To evaluate how well our classifier is performing, we use:

- **Cross Validation** – we split the labeled data into two sets: 80% training and 20% validation. Because of the time-based nature of network flow, we use both random splitting as well as time-based splitting.
- **Precision & Recall** – these are two metrics derived from cross validation. Recall gives us a measure of quantity; it is the percent of the true positives that our classifier managed to identify. Precision gives us a measure of quality; it shows us the percent of our marked positives that are true positives.
- **Receiving operating characteristic (ROC) curve** – this is a graph that plots true positive rate against false positive rate as the discrimination threshold is varied. The area under the ROC curve represents the probability that a randomly chosen true positive will be ranked higher than a randomly chosen true negative.

Feature Extraction & Data Preparation

For our project, we started with anonymized packet traces provided by UCLA & CAIDA. Because packet traces contain sensitive information, they often are not a practical choice for training or test datasets. We provide an alternative by extracting completely anonymous statistical features from the network flow and use those as features for classification. Instead of using sensitive packets as data points, we simply aggregate statistics over a 10 second time slice as a data point. Several of the features we have include:

- Shannon Entropy of packet features (IP, port, etc)
- Averages (packet size, bytes per unique IP, etc)
- Total counts (packets, unique IP, port, etc)

$$H = -\sum p(x) \log p(x)$$

Equation for shannon entropy

After extracting the features, we prepare them for the classifier by scaling them through SK-learn's preprocessing functions.

Machine Learning Algorithms

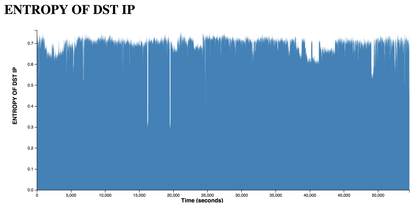
We tried a variety of machine learning algorithms for the classifier of Banana:

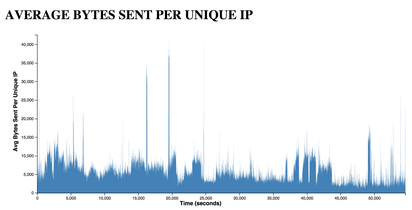
Support Vector Machines (SVMs) – the currently used algorithm for supervised learning and classification of the data points. The advantages of SVMs is that they are effective in high dimensional spaces and has a regularization parameter which can be tuned to avoid overfitting.
K-nearest neighbors (KNN) – while KNN is extremely simple to implement & often provides great results for basic classification problems, it is not robust to noisy data, which is the case for network flow. KNN does not "learn" from training data but rather just uses the training data for classification
Decision Trees – decision trees have the advantage of not requiring much data preparation, so it was the first implementation we tried. However, our classifier was clearly overfitting as it was creating complex trees that did not generalize the data well.

Tools Used

SK Learn – simple & effective machine learning kit that has performance metrics & many models
D3.js – used for data visualization & exploration
Scapy – parsing & exploring small datasets
Wireshark – parsing very large packet captures
Spark – fast data exploration due to keeping objects in memory

Data Exploration & Visualizations

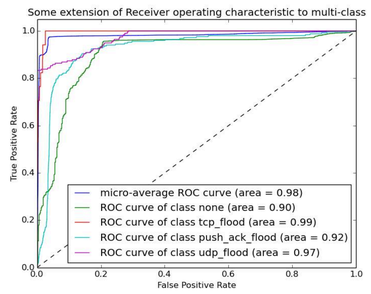




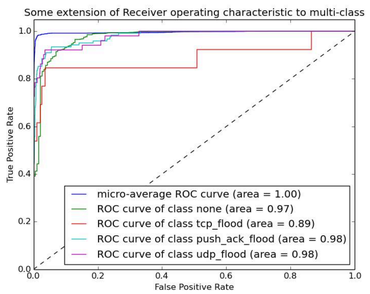
See more at http://code.io/banana

Cross Validation Results (SVMs)

1. ROC Curves



Validation set split by time



Validation set split randomly

2. Precision & Recall (Validation set split by time)

Type	Precision	Recall
PUSH ACK Attack	59.33%	84.52%
TCP Flood	53.33%	47.06%
UDP Flood	98.98%	98.48%

3. Precision & Recall (Validation set split randomly)

Type	Precision	Recall
PUSH ACK Attack	95.69%	86.72%
TCP Flood	77.78%	70.40%
UDP Flood	97.10%	93.47%

Lessons Learned & Challenges

- Lack of data due to confidentiality/sensitivity of data.
- Lack of specific TCP flood attack data.
- Data cleaning, extraction, and parsing takes up a significant amount of time.
- Data is extremely large (days of network flow).
- Don't use too many features that correlate or overfitting will occur
- Different hosts have different magnitudes of bandwidth, so scaling is extremely useful.