Programmatically Detecting and Mitigating DDoS Attacks

Bryan Andrews, Brandon Bosh, Jacob Gold, Lavaniyaa Seshathiri

Introduction

The Distributed Denial of Service, or DDoS, the attack is a common network-oriented attack method typically used to disrupt online services. DDoS attacks aim to incapacitate a computer network by flooding it with traffic from a large number of hosts simultaneously. There are various ways to perform a DDoS attack, such as SYN flooding. Consequently, there are a number of methods required to combat these attacks.

Given today's heavy reliance on online cloud services, it is critical for businesses to be able to detect, mitigate, and prevent these attacks from affecting the availability of their services to customers. We will write a set of programs to detect and partially mitigate a DDoS attack on a target host.

We will develop three programs that accomplish the following: detect unusually-high network traffic, analyzes traffic for malicious patterns, and automatically block IP addresses determined to be participants of the attack. To achieve this, we will use a pcap file containing a data set representing a single DDoS attack. Although we will not be replicating a DDoS attack, this detection mechanism is designed to be suitable for determining whether real-time traffic is malicious or not.

Background

DDoS attacks come in a variety shapes and sizes and, as such, demand an equal variety of algorithms for detection and mitigation. No two attacks can be addressed in exactly the same way, although they typically fall into one of three categories: volume-based, protocol attacks, and application attacks. Our solution addresses attacks from the first two of these categories.
Volume-based attacks are the most infamous form of DDoS attack, typically slowing or crashing a network by exhausting its bandwidth. These included UDP floods, ICMP floods, and other spoofed packet attacks. Initial detection of a volume-based attack is often simple, requiring only the most basic of network monitoring to determine when bandwidth is becoming saturated, or when there are large spikes in network traffic. Correctly determining the culprits, however, is often much harder or even impossible. Because of this, dealing with DDoS attacks of this variety usually stems from mitigation techniques rather than outright prevention. These techniques include blacklisting/whitelisting, connection tracking, and redirection to high-capacity networks with “scrubbing” filters.

Protocol attacks target specific server resources or communication equipment rather than flooding an entire network. There are many varieties of protocol attacks, with some of the most well-known being SYN floods, fragmented packet attacks, and Ping of Death. Mitigating a protocol attack is highly dependent on the specific attack being used. Because of this, separate techniques must be employed for every type of attack that one wishes to protect against.

Motivation

As previously stated, online services are a common target of Distributed Denial of Service attacks. Service providers often seek to keep these services up and accessible to clients for as long as possible without interruption, maintaining what is known as high availability. While DDoS attacks do not typically bring down the service, they do tend to make them unavailable for an extended period of time. This is especially detrimental to companies whose business model relies almost exclusively on paid, online software-as-a-service solutions, such as Netflix and Spotify. Interruptions to these services can cause companies to lose money and customers.

It is critical for businesses to be able to detect when their infrastructure is being targeted by DDoS attacks so that they can take action to mitigate them. Companies can also take precautionary measures to prevent DDoS attacks from overwhelming their systems. Specifically, companies with cloud-based infrastructure are able to dynamically scale services based on the amount of traffic received due to their architecture being scalable, or alternatively using a service like CloudFlare. However,
services that are hosted on-premises, are not scalable in their design may still be susceptible to these attacks.

Our project aims to solve this issue by detecting unusual traffic patterns on a host, analyzing that traffic to confirm that it is an attack (and not just an unusually-high load), and mitigating it in the form of blocking the attacking IP address. While more robust solutions currently exist, namely those who address this issue through the infrastructure architecture, our solution is applicable to legacy applications and systems which were not designed with DDoS detection and mitigation in mind.

Methodology & Results

Detection Component

The detection component is a bash script which monitors the specified network interface for unusual spikes in traffic, as well as network interface saturation. The script continuously samples received network traffic data from the `rx_bytes` file located in the directory `/sys/class/net/X/statistics`, where `X` is the name of the network interface being monitored. Using this sample data, the script then computes the number of bytes received per second. It then checks to see if the difference in number of bytes received from a time of 0 seconds to a time of 1 second exceeds the maximum allowable deviation (in bytes), which is specified to the script as a parameter. Note that the maximum allowable deviation should be determined by examining the contents of the `rx_bytes` file for a few seconds, and multiplying the approximate difference observed by a factor of at least 2. If the script detects that this maximum allowable deviation is exceeded, it invokes our Python script for further network traffic analysis.

Additionally, the detection component checks the number of bytes received against the available bandwidth of the specified network interface. To achieve this, it retrieves the total available bandwidth from the `speed` file located in `/sys/class/net/X` for the specified network interface, `X`. If the computed bytes received per second exceeds 80% of the interface’s total bandwidth, we consider the interface to be “saturated” at that point in time. If the interface remains “saturated” for longer than 30 seconds, we also invoke our Python script for further analysis.
Analysis Component

For the Python script, we used pyshark (a wrapper for tshark), to capture packets from a packet capture, or pcap, file. Note that the pcap file could be substituted for a live capture of network traffic with minimal modifications. Then, for each packet, we copy its source IP address into a list with the number of times the IP address has been seen as the next element. The loop doing this is meant to break on how many different packets you have seen so that you can inform another program of your results or this could be modified to have a child process modify without exiting the loop. The last loop iterates through and checks to see if too many packets with the same address were sent to the user during the capture period.

We are testing the Python script based on the pcap files downloaded from the GitHub repositories and other online resources like Wireshark. Our initial goal is to detect the occurrence of SYN flood in the network. Thus, the dataset has a large number of SYN packets. We also have a benign capture downloaded from the Stratosphere Lab dataset which will be used to evaluate the accuracy of our logic on detecting the malicious traffic. If time permits, we also would like to simulate a SYN flood attack and test our entire flow.

TCP SYN Flood

To detect a TCP SYN flood the script keeps track of all open connections that are being idle. It detects this by logging any syn packet that is trying to open a connection on the system. This is compared to subsequent packets from the same ip address. If after a predetermined time or a predetermined number of packets has passed, then the script compares if the IP address has sent the final packet in the three way handshake. If it hasn't, it logs the IP address. To compensate for false positives a threshold is set and once the threshold is breached the script exits and alerts that a SYN flood attack was detected.

UDP Flood Detection

As part of the grad student project, the proposal was to work on DDoS detection mechanism based on TCP SYN flood or UDP flood attack with Bro IDS. Due to the lack of proper dataset online, the detection mechanism was performed with Python by analyzing the pcap files with pyshark. In this section, the Bro IDS and the scripting framework will be explained in brief and later the detection mechanism with Python will be described in detail.
Bro Intrusion Detection System

Intrusion Detection Systems, in general, monitor the network traffic continuously and alert the admin/security team in terms of a suspicious event. IDS like snort mostly works based on stored signatures and alerts when a similar event happens. Bro supports for both signature and anomaly-based detection mechanism where there are a constant collection and storage of different log files conn.log, dns.log, etc. Apart from security alerts, it can also help with performance measurements and troubleshooting. The architecture of Bro is as follows:

The raw packets coming in from the network are changed to events by the event engine module like HTTP request are transferred to HTTP events. But the main task of categorizing whether it is malicious or benign packets is done by the Script interpreter engine. For example, bro has built-in scripts that check for ssh brute force attempts, malware files downloaded, etc. When these scripts detect something amiss, then an alert would be triggered according to the configuration made by the system administrator.

Initially, the log files of DDoS botnet families like Avzhan was taken as the dataset but there were not enough traces of UDP/TCP flood in the acquired log files. Thus the detection of UDP flood with Python and pcap files has been performed considering the availability of pcap files.

UDP Flood

A kind of DDoS attack in which the attacker sends a stream of UDP packets continuously from multiple or single hosts to the victim machine. A single host mode is usually in a simple DoS attack. But most of the recent attacks are based on multiple hosts which is typically a bot controlled by the attackers. As the victim will be overwhelmed by the requests it will saturate and start accepting legit traffic thereby creating a denial of service.

In general, whenever a system receives a UDP packet, it looks for the port number in the packet. Then, it checks whether an application is running on a particular port and expecting some packets. If so, the packet will be forwarded to that port for further processing. Else, the system will send a “Destination not reachable” to the machine that sent the request. When an attacker floods the system with thousands of UDP packets, the system will exhaust its resources on the way of responding to these packets.
For the project, we have considered the DDoS UDP flood, that is, UDP flood from random source IPs which will be challenging to detect and block. Due to the time constraint, we have implemented the solution with pcap files rather than live capture but we believe the algorithm remains the same for detection.

Algorithm

After acquiring the pcap files from the sysadmin or any security specialist, read the pcap through pyshark and iterate over the packets one by one.

1. Check the ratio of the number of UDP packets against the total packets. This is the first simple indicator.
2. Iterate through a baseline of 100 UDP packets (avoid iterating a large pcap) and save the IP addresses of the packets to a CSV.
3. Filter the ICMP packets from the pcap and retrieve the destination IPS of packets with “Destination Unreachable” message.
4. Compare the destination IP addresses from ICMP packets with the IPs are saved in the CSV. If there is more than 80% match, then there is a higher chance of UDP flood happening in the system and the code alerts the admin.
5. One more parameter to look for is the ratio of the number of ICMP destination unreachable packets against the total ICMP packets.

The code was tested against one malicious pcap and one benign pcap. The sample screenshots are attached below:

Mitigation Component

To mitigate the effect of an attack we have created a script to block IP’s determined to be suspicious by the Python script. It is a fairly simple (~20 line) bash script which utilizes the iptables program to configure the Linux kernel firewall rules. The kernel firewall contains 5 chains that are used to determine how incoming and outgoing packets should be routed. This firewall cannot be accessed and configured directly, so the kernel supplies iptables as a user-space program to add or drop rules. The script
‘blockIP.sh’ takes in an IP address (as a dotted decimal ipv4 address) and supplies it to a command which adds a rule containing the supplied IP to the INPUT chain of the kernel’s firewall with policy DROP.

Once this rule has been added, the kernel will drop any packets received from the designated IP. To view a listing of all addresses blocked this way, one can run the command ‘sudo iptables -L INPUT -v -n’, which will display a listing like such:

![iptables listing example](image)

User Guide

Installing Dependencies

The detection and mitigation components of our project are comprised of bash scripts and largely use standard commands included in most Linux distributions. We used Ubuntu 16.04 LTS to develop and test our programs, however, any Debian-based Linux distribution should be compatible. The mitigation component uses `iptables`, which may or may not be enabled on some systems, but it will enable it automatically if this is the case.

Use the below command to install tshark, Python 3, pip, and several other dependencies required by the analysis component of our project:

```
sudo apt-get install python3 python3-pip tshark -y &&
pip install pyshark pandas --user
```
When prompted, select Yes to allow tshark to be used by non-root users.

**Obtaining Packet Capture Files**

Given the large size of the packet capture files used to develop and test the analysis component of our project, we are unable to provide those with our source code. Instead, you can download them at the links below:

- **Dataset 1:**
- **Dataset 2:** from Stratosphere project
  [https://drive.google.com/open?id=13RUmuYrCtjXtXD-R3G99CT0qfhfrdqb](https://drive.google.com/open?id=13RUmuYrCtjXtXD-R3G99CT0qfhfrdqb)

**Generating Traffic**

We used the Hping3 tool to generate the UDP flood and TCP SYN flood traffic to test against the code. The tool provides the flexibility of performing attacks from random source IPs by spoofing. An example command is as follows:

```
hping3 --rand-source --flood --udp -p 53 130.127.135.10
```

To obtain the pcap, one machine (attacker) should be running the hping3 command with TCP/UDP option and the victim machine should have the tcpdump running with the option of storing it to a pcap file.

**Establishing a Detection Baseline**

Prior to starting the entry point of our set of scripts, a baseline of deviation in traffic on the interface to be monitored should be established. We recommend briefly performing a task that is network-intensive or represents a typical network pattern for the given host and observing the difference in bytes received for several seconds. For example, if the name of the interface to be monitored is `enp0s3`, issue the following command a few times:

```
cat /sys/class/net/enp0s3/statistics/rx_bytes
```
In our tests, we performed a video streaming task and recorded the following three measurements in bytes: 202707211, 203290976, and 204299820. From this, we see that the number of bytes received is increasing by roughly 1,000,000 bytes per second. We recommend multiplying this amount by a factor of at least 2, making the deviation slightly larger than what is observed to be “typical,” in order to avoid false positives in the detection component. In this case, we choose 4,000,000 bytes as the maximum allowed deviation in traffic.

Starting the Scripts

After establishing a baseline deviation, invoke the detection script with the following command:

```
./monitor_traffic.sh <network interface> <maximum allowed deviation>
```

The network interface parameter is the name of the interface to monitor traffic on, which can be obtained using `ifconfig`, and maximum allowed deviation is the number of bytes calculated from the previous step. The detection component is the entry point to our set of programs, and will continuously run. It will only call the analysis component when the maximum allowed deviation is exceeded or the interface becomes saturated for further analysis of traffic deemed atypical.

Conclusion

Accomplishments

We developed a detection script that successfully detects unusual network traffic patterns potentially caused by a DDoS attack based on deviation in received traffic and network interface saturation. Additionally, we developed two analysis scripts that are able to correctly identify SYN flood attacks and UDP flood attacks from a data set representing traffic to a network interface. Finally, we developed a mitigation script that blocks IP addresses determined to be associated with a DDoS attack.
Further Work

In its current state, our algorithm can only detect a handful of DDoS attack varieties and has only a single mitigation solution. In order to be used as a one-stop DDoS mitigation solution, the algorithm used in our Python file must be expanded to include other varieties of protocol attacks as well as the most widely-used application-level attacks. This is likely beyond the scope of tshark’s monitoring abilities, so a custom solution would have to be implemented for each distinct protocol/application attack. Our mitigation solution is also limited since it does not prevent attacks that target communications equipment in between the attacker and host. In addition, this solution only applies to Linux machines as it relies on both Bash and iptables.
References

7. https://pdfs.semanticscholar.org/4c51/eefe95d9c0a5bda937805dc342d33e9cdd0c.pdf
8. https://www.us-cert.gov/ncas/alerts/TA14-017A