**Check if your server is under DDOS attack**

Login to your server as root and fire the following command, using which you can check if your server is under DDOS attack or not.

**#netstat -anp |grep 'tcp\|udp' | awk '{print $5}' | cut -d: -f1 | sort | uniq -c | sort -n**

This command will show you the list of IP's which have logged in is maximum number of connections to your server.

ddos becomes more complex as attackers use fewer connections with more number of attacking IP's. In such cases, you should get less number of connections even when your server is under ddos. One important thing that you should check is the number of active connections that your server currently has. For that execute the following command.

**#****netstat -n | grep :80 |wc -l**

The above command will show the active connections that are open to your server.

You can also fire the following command .

**#netstat -n | grep :80 | grep SYN |wc -l**

Result of active connections from the first command will vary but if it shows connections more than 500, then you will be definitely having problems. If the result after you fire second command is 100 or above then you are having problems with sync attack.

Once you get an idea of the ip attacking your server, you can easily block it.

Fire the following command to block that ip or any other specific ip:

**#route add ipaddress reject**

Once you block a paricular IP on the server, you can even crosscheck if the IP is blocked or not

by using the following command:

**#route -n |grep IPaddress**

You can also block a IP with iptables on the server by using the following command.

**#iptables -A INPUT 1 -s IPADRESS -j DROP/REJECT**

**#service iptables restart**

**#service iptables save**

After firing the above command, KILL all httpd connection and than restart httpd service by

using following command:

**#killall -KILL httpd**

**#service httpd startssl**

**What is Sync Flood Attack? Detection and prevention in Linux**

**What is Sync Flood Attack?**

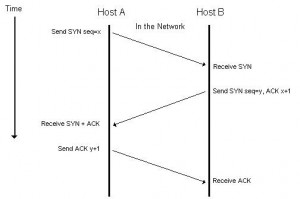
A SYN flood attack is a form of denial-of-service attack in which an attacker sends a large number of SYN requests to a target system’s services that uses TCP protocol. This will consume the server resources to make the system unresponsive to legitimate traffic. This attack can occur on any services that use TCP protocol and mainly on web service. In this article, we will go through the basics of SYN flood attacks and the mitigation steps in detail.

The SYN Flood attack exploits an implementation characteristic of the Transmission Control Protocol (TCP), which is called 3-way handshake. Following are the steps that happen in a normal 3-way handshake.

1. The client requests a connection by sending a SYN (synchronize) message to the server.

2. The server acknowledges this request by sending SYN-ACK back to the client.

3. The client responds with an ACK, and the connection is established



A SYN flood attack works by not responding to the server with the expected ACK code. By these half-open connections, the target machines TCP backlog will get filled up and hence all new connections may get ignored. This will cause the legitimate users will also get ignored.

This attack can take place in two ways

**1. Direct Attack**

In this kind of attack, attackers rapidly send SYN segments without spoofing their IP source address. When detected, this type of attack is very easy to defend, because we can add a simple firewall rule to block packets with the attacker's source IP address which will shutdown the attack.

**2. User IP Address Spoofing**

This is a more complex form of attack than the direct attack. In this method, the malicious machine will send SYN request floods to the target machine from spoofed IP addresses, causing the server to send the SYN-ACK to a falsified IP address - which will not send an ACK because it "knows" that it never sent a SYN.

**Detecting Sync Flood Attack**

The generic symptom of SYN Flood attack to a web site visitor, is that a site takes a long time to load, or loads some elements of a page but not others. If you suspect a SYN Flood attack on a web server, you can use netstat command to check the web server connection requests that are in “SYN\_RECEIVED” state.

**#netstat -tuna | grep :80 | grep SYN\_RECV**

If this shows numerous connections with this state, the server could be under SYN Flood attack. If the attack is direct with large number of SYN\_RECV packets from a single IP address, you can stop this attack by adding that IP in firewall. If you have APF or CSF firewall installed in your server, you can accomplish this by executing the following command.

**#apf –d IPADDRESS**

**#csf –d IPADDRESS**

**Defending Sync Flood Attack**

**Using Sync Cookies**

This is the most effective defending method for SYN Flood attack. The use of SYN cookies allows a server to avoid dropping connections when the SYN queue fills up. Instead, the server behaves as if the SYN queue had been enlarged. The server sends back the appropriate SYN+ACK response to the client but discards the SYN queue entry. If the server then receives a subsequent ACK response from the client, the server is able to reconstruct the SYN queue entry using information encoded in the TCP sequence number.

SYN cookies can be enabled by adding the following to **/etc/sysctl.conf**

**net.ipv4.tcp\_syncookies = 1**

After modifying the sysctl configuration file, you need to execute the following command to load sysctl settings from the file /et**c/sysctl.conf** file.

**#sysctl –p**

**Increasing the Sync backlog queue**

Optional defending technique is to increasing the SYS backlog queue size. The default size is 1024. This can be done by adding the following to **/etc/sysctl.conf**

**net.ipv4.tcp\_max\_syn\_backlog = 2048**

**Reducing SYN\_ACK retries**

Tweaking the kernel parameter tcp\_synack\_retries causes the kernel to close the SYN\_RECV state connections earlier. Default value is 5.

**net.ipv4.tcp\_synack\_retries = 3**

**Setting SYN\_RECV timeout**

Lowering the timeout value for SYN\_RECV will help to reduce the SYN flood attack. The default value is 60 and we can reduce it to 40 or 45. This can be done by adding the following line to sysctl.conf.

**net.ipv4.netfilter.ip\_conntrack\_tcp\_timeout\_syn\_recv=45**

**Preventing IP Spoofing**

The following sysctl parameter will help to protect against IP spoofing which is used for SYN flood attacks.

**net.ipv4.conf.all.rp\_filter = 1**

Many hosting companies provides protection against SYN attack by deploying firewalls that employs SYN flood defense such as Netscreen or Appsafe.

**Monitor TCP Sync Flooding Attacks**

**#netstat -n | grep SYN\_RECV**

You can also see SYN flood traffic under ss, although by default ss hides this traffic category. To see if type ss -a state SYN-RECV.

**#ss -a state SYN-RECV**

For our check we'll try to avoid both netstat and ss because they are a bit to resource intensive. We want something we can easily monitor and run frequently, and although ss is still better than netstat, its still not perfect. Instead we'll go straight to '/proc/net/tcp'. The following script though will parse the file, count all the connections in SYN\_RECV and create a hash of counters for each ip address with open connections. If the warning or critical thresholds are reached the script will exit with the correct status code and return an output with who the top offenders are (although the source ip is probably spoofed anyways).

The output like this.

**#ruby /usr/lib64/nagios/plugins/check\_syn\_flood.rb -w 500 -c 1000**

SYN Count: 239

**#ruby /usr/lib64/nagios/plugins/check\_syn\_flood.rb -w 500 -c 100**

SYN FLOOD CRITICAL SYN Count: 182 | DST: 192.168.1.1: 37 SRC: 1.2.3.4: 6

And here's the full script:

**#!/usr/bin/ruby**

**# Nagios check for TCP SYN Flooding Attack**

**## check\_syn\_flood.rb -w WarningLevel -c CriticalLevel**

**#**

**# Written by Robert Birnie**

**# Source: http://www.uberobert.com/nagios-check-for-tcp-syn-flooding-attacks/**

**#**

**# /proc/net/tcp format:**

**# sl local\_address rem\_address st tx\_queue rx\_queue tr tm->when retrnsmt uid timeout inode**

**# 0: 0100007F:46E0 00000000:0000 0A 00000000:00000000 00:00000000 00000000 0 0 36206 1 ffff810224e52140 3000 0 0 2 -1**

**#**

**# %nethash = (**

**# '01', => TCP\_ESTABLISHED,**

**# '02', => TCP\_SYN\_SENT,**

**# '03', => TCP\_SYN\_RECV,**

**# '04', => TCP\_FIN\_WAIT1,**

**# '05', => TCP\_FIN\_WAIT2,**

**# '06', => TCP\_TIME\_WAIT,**

**# '07', => TCP\_CLOSE,**

**# '08', => TCP\_CLOSE\_WAIT,**

**# '09', => TCP\_LAST\_ACK,**

**# '0A', => TCP\_LISTEN,**

**# '0B', => TCP\_CLOSING,**

**# );**

**require 'optparse'**

**require 'scanf'**

**options = {}**

**optparse = OptionParser.new do |opts|**

**opts.on('-w', '--warn warning') do |f|**

**options[:warn] = f**

**end**

**opts.on('-c', '--critical critical') do |f|**

**options[:crit] = f**

**end**

**end**

**optparse.parse!**

**raise OptionParser::MissingArgument if options[:warn].nil?**

**raise OptionParser::MissingArgument if options[:crit].nil?**

**@src\_ips = Hash.new(0)**

**@dst\_ips = Hash.new(0)**

**@count = 0**

**exit\_code = 0**

**File.readlines('/proc/net/tcp').each do |line|**

**i = line.split(' ')**

**if i[3] == '03'**

**@count += 1**

**@dst\_ips[i[1].split(':')[0].scanf('%2x'\*4)\*"."] += 1**

**@src\_ips[i[2].split(':')[0].scanf('%2x'\*4)\*"."] += 1**

**end**

**end**

**msg = "SYN Count: #{@count}"**

**if @count > options[:crit].to\_i or @count > options[:warn].to\_i**

**top\_dst\_ip = @dst\_ips.max\_by{|k,v| v}**

**top\_src\_ip = @src\_ips.max\_by{|k,v| v}**

**crit = "| DST: #{top\_dst\_ip[0].split('.').reverse.join('.')}: #{top\_dst\_ip[1]} SRC: #{top\_src\_ip[0].split('.').reverse.join('.')}: #{top\_src\_ip[1]}"**

**if @count.to\_i > options[:crit].to\_i**

**exit\_code = 2**

**msg = "SYN FLOOD CRITICAL #{msg} #{crit}"**

**elsif @count.to\_i > options[:warn].to\_i**

**exit\_code = 1**

**msg = "SYN FLOOD WARN #{msg} #{crit}"**

**end**

**end**

**puts msg**

**exit exit\_code**

Now the script is completed.