DDoS Protection With IPtables: The Ultimate Guide

There are different ways of building your own anti-DDoS rules for iptables. We will be discussing the most effective iptables DDoS protection methods in this comprehensive tutorial.

This guide will teach you how to:

1. Select the best iptables table and chain to stop DDoS attacks
2. Tweak your kernel settings to mitigate the effects of DDoS attacks
3. Use iptables to block most TCP-based DDoS attacks
4. Use iptables SYNPROXY to block SYN floods

Please note that this article is written for professionals who deal with Linux servers on a daily basis.

If you just want to protect your online application from DDoS attacks, you can use our remote protection, a VPS with DDoS protection or a DDoS protected bare metal server.

While one can do a lot with iptables to block DDoS attacks, there isn't a way around actual hardware firewalls (we recently reviewed RioRey DDoS mitigation hardware) to detect and stop large DDoS floods.

However, **it isn't impossible to filter most bad traffic at line rate using iptables!**

We'll only cover protection from TCP-based attacks. Most UDP-based attacks are amplified reflection attacks that will exhaust the network interface card of any common server.

The only mitigation approach that makes sense against these types of attacks is to block them at the edge or core network or even at the carrier already.

> *Did you know we now offer unmetered bandwidth VPS plans with DDoS protection in Chicago, Illinois and Bucharest, Romania?*

If they are able to reach your server, there isn't much you can do against those multi-Gbit/s attacks except to move to a DDoS protected network.

What Is IPtables?

netfilter iptables (soon to be replaced by nftables) is a user-space command line utility to configure kernel packet filtering rules developed by netfilter.

It’s the default firewall management utility on Linux systems – everyone working with Linux systems should be familiar with it or have at least heard of it.

iptables can be used to filter certain packets, block source or destination ports and IP addresses, forward packets via NAT and a lot of other things.

Most commonly it’s used to block destination ports and source IP addresses.

Why Your IPtables Anti-DDoS Rules Suck

To understand why your current iptables rules to prevent DDoS attacks suck, we first have to dig into how iptables works.

iptables is a command line tool used to set up and control the *tables* of IP packet filter rules. There are different tables for different purposes.

IPtables Tables

Filter: The `filter` table is the default and most commonly used table that rules go to if you don't use the `-t` (`–table`) option.

NAT: This table is used for Network Address Translation (NAT). If a packet creates a new connection, the `nat` table gets checked for rules.

Mangle: The `mangle` table is used to modify or mark packets and their header information.

Raw: This table's purpose is mainly to exclude certain packets from connection tracking using the NOTRACK target.

As you can see there are four different tables on an average Linux system that doesn't have non-standard kernel modules loaded. Each of these tables supports a different set of iptables *chains*.

IPtables Chains

PREROUTING: raw, nat, mangle

- Applies to packets that enter the network interface card (NIC)

INPUT: filter, mangle

- Applies to packets destined to a local socket

FORWARD: filter, mangle

- Applies to packets that are being routed through the server

OUTPUT: raw, filter, nat, mangle

- Applies to packets that the server sends (locally generated)

POSTROUTING: nat, mangle

- Applies to packets that leave the server

Depending on what kind of packets you want to block or modify, you select a certain iptables table and a chain that the selected table supports.

Of course, we’re still missing an explanation of iptables targets (ACCEPT, DROP, REJECT, etc.), but we’re assuming that if you’re reading this article, you already know how to deal with iptables.

We’re going to explain why your iptables rules suck to stop DDoS and not teach you how to use iptables. Let’s get back to that.

If you want to block a DDoS attack with iptables, performance of the iptables rules is extremely important. Most TCP-based DDoS attack types use a high packet rate, meaning the sheer number of packets per second is what causes the server to go down.

That’s why you want to make sure that you can process and block as many packets per second as possible.

You’ll find that most if not all guides on how to block DDoS attacks using iptables use the `filter` table and the INPUT chain for anti-DDoS rules.

The issue with this approach is that the INPUT chain is only processed after the PREROUTING and FORWARD chains and therefore only applies if the packet *doesn’t* match any of these two chains.

This causes a delay in the filtering of the packet which consumes resources. In conclusion, to make our rules as effective as possible, **we need to move our anti-DDoS rules as far up the chains as possible.**

The first chain that can apply to a packet is the PREROUTING chain, so ideally we'll want to filter the bad packets in this chain already.

However, the `filter` table doesn't support the PREROUTING chain. To get around this problem, we can simply use the `mangle` table instead of the `filter` table for our anti-DDoS iptables rules.

It supports most if not all rules that the `filter` table supports while also supporting **all** iptables chains.

So you want to know why your iptables DDoS protection rules suck? It's because you use the `filter` table and the INPUT chain to block the bad packets!

The best solution to dramatically increase the performance of your iptables rules and therefore the amount of (TCP) DDoS attack traffic they can filter is to **use the `mangle` table and the PREROUTING chain!**

The Best Linux Kernel Settings to Mitigate DDoS

Another common mistake is that **people don't use optimized kernel settings** to better mitigate the effects of DDoS attacks.

Note that this guide focuses on CentOS 7 as the operating system of choice. CentOS 7 includes a recent version of iptables and support of the new SYNPROXY target.

We won't cover every single kernel setting that you need to adjust in order to better mitigate DDoS with iptables.

Instead, we provide a set of CentOS 7 kernel settings that we would use. Just put the below in your `/etc/sysctl.conf` file and apply the settings with `sysctl -p`.

Anti-DDoS Kernel Settings (sysctl.conf)

```
kernel.printk = 4 4 1 7
kernel.panic = 10
kernel.sysrq = 0
kernel.shmmax = 4294967296
kernel.shmall = 4194304
kernel.core_uses_pid = 1
kernel.msgmnb = 65536
kernel.msgmax = 65536
vm.swappiness = 20
vm.dirty_ratio = 80
vm.dirty_background_ratio = 5
fs.file-max = 2097152
net.core.netdev_max_backlog = 262144
net.core.rmem_default = 31457280
net.core.rmem_max = 67108864
net.core.wmem_default = 31457280
net.core.wmem_max = 67108864
net.core.somaxconn = 65535
net.core.optmem_max = 25165824
net.ipv4.neigh.default.gc_thresh1 = 4096
net.ipv4.neigh.default.gc_thresh2 = 8192
net.ipv4.neigh.default.gc_thresh3 = 16384
net.ipv4.neigh.default.gc_interval = 5
net.ipv4.neigh.default.gc_stale_time = 120
net.netfilter.nf_conntrack_max = 10000000
net.netfilter.nf_conntrack_tcp_loose = 0
net.netfilter.nf_conntrack_tcp_timeout_established = 1800
net.netfilter.nf_conntrack_tcp_timeout_close = 10
net.netfilter.nf_conntrack_tcp_timeout_close_wait = 10
net.netfilter.nf_conntrack_tcp_timeout_fin_wait = 20
net.netfilter.nf_conntrack_tcp_timeout_last_ack = 20
net.netfilter.nf_conntrack_tcp_timeout_syn_recv = 20
net.netfilter.nf_conntrack_tcp_timeout_syn_sent = 20
net.netfilter.nf_conntrack_tcp_timeout_time_wait = 10
net.ipv4.tcp_slow_start_after_idle = 0
net.ipv4.ip_local_port_range = 1024 65000
net.ipv4.ip_no_pmtu_disc = 1
net.ipv4.route.flush = 1
net.ipv4.route.max_size = 8048576
net.ipv4.icmp_echo_ignore_broadcasts = 1
net.ipv4.icmp_ignore_bogus_error_responses = 1
net.ipv4.tcp_congestion_control = htcp
net.ipv4.tcp_mem = 65536 131072 262144
net.ipv4.udp_mem = 65536 131072 262144
net.ipv4.tcp_rmem = 4096 87380 33554432
net.ipv4.udp_rmem_min = 16384
net.ipv4.tcp_wmem = 4096 87380 33554432
net.ipv4.udp_wmem_min = 16384
net.ipv4.tcp_max_tw_buckets = 1440000
net.ipv4.tcp_tw_recycle = 0
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_max_orphans = 400000
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_rfc1337 = 1
net.ipv4.tcp_syncookies = 1
net.ipv4.tcp_synack_retries = 1
net.ipv4.tcp_syn_retries = 2
net.ipv4.tcp_max_syn_backlog = 16384
net.ipv4.tcp_timestamps = 1
net.ipv4.tcp_sack = 1
net.ipv4.tcp_fack = 1
net.ipv4.tcp_ecn = 2
net.ipv4.tcp_fin_timeout = 10
net.ipv4.tcp_keepalive_time = 600
net.ipv4.tcp_keepalive_intvl = 60
net.ipv4.tcp_keepalive_probes = 10
net.ipv4.tcp_no_metrics_save = 1
net.ipv4.ip_forward = 0
net.ipv4.conf.all.accept_redirects = 0
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.all.rp_filter = 1
```

These sysctl.conf settings help to maximize the performance of your server under DDoS as well as the effectiveness of the iptables rules that we're going to provide in this guide.

The Actual IPtables Anti-DDoS Rules

Considering you now know that **you need to use the `mangle` table and the PREROUTING chain** as well as optimized kernel settings to mitigate the effects of DDoS attacks, we'll now move on to a couple of example rules to mitigate most TCP DDoS attacks.

DDoS attacks are complex.

There are many different types of DDoS and it's close to impossible to maintain signature-based rules against all of them.

But luckily there is something called connection tracking (nf_conntrack kernel module), which can help us to mitigate almost any TCP-based DDoS attack that doesn't use SYN packets that seem legitimate.

This includes all types of ACK and SYN-ACK DDoS attacks as well as DDoS attacks that use bogus TCP flags.

We'll start with just five simple iptables rules that will already drop many TCP-based DDoS attacks.

Block Invalid Packets

```
iptables -t mangle -A PREROUTING -m conntrack --ctstate INVALID -j DROP
```

This rule blocks all packets that are not a SYN packet and don't belong to an established TCP connection.

Block New Packets That Are Not SYN

```
iptables -t mangle -A PREROUTING -p tcp ! --syn -m conntrack --ctstate NEW -j DROP
```

This blocks all packets that are new (don't belong to an established connection) and don't use the SYN flag. This rule is similar to the "Block Invalid Packets" one, but we found that it catches some packets that the other one doesn't.

Block Uncommon MSS Values

```
iptables -t mangle -A PREROUTING -p tcp -m conntrack --ctstate NEW -m tcpmss ! --m
```

The above iptables rule blocks new packets (only SYN packets can be new packets as per the two previous rules) that use a TCP MSS value that is not common. This helps to block dumb SYN floods.

Block Packets With Bogus TCP Flags

```
iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG NONE -
iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,SYN FIN,SYN -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,RST FIN,RST -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,ACK FIN -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ACK,URG URG -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ACK,FIN FIN -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ACK,PSH PSH -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL ALL -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL NONE -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL FIN,PSH,URG -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j DROP
iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL SYN,RST,ACK,FIN,URG -j DRO
```

The above ruleset blocks packets that use bogus TCP flags, ie. TCP flags that legitimate packets wouldn’t use.

Block Packets From Private Subnets (Spoofing)

```
iptables -t mangle -A PREROUTING -s 224.0.0.0/3 -j DROP
iptables -t mangle -A PREROUTING -s 169.254.0.0/16 -j DROP
iptables -t mangle -A PREROUTING -s 172.16.0.0/12 -j DROP
iptables -t mangle -A PREROUTING -s 192.0.2.0/24 -j DROP
iptables -t mangle -A PREROUTING -s 192.168.0.0/16 -j DROP
iptables -t mangle -A PREROUTING -s 10.0.0.0/8 -j DROP
iptables -t mangle -A PREROUTING -s 0.0.0.0/8 -j DROP
iptables -t mangle -A PREROUTING -s 240.0.0.0/5 -j DROP
iptables -t mangle -A PREROUTING -s 127.0.0.0/8 ! -i lo -j DROP
```

These rules block spoofed packets originating from private (local) subnets. On your public network interface you usually don't want to receive packets from private source IPs.

These rules assume that your loopback interface uses the 127.0.0.0/8 IP space.

These five sets of rules alone already block many TCP-based DDoS attacks at very high packet rates.

With the kernel settings and rules mentioned above, you'll be able to filter ACK and SYN-ACK attacks at line rate.

Additional Rules

```
iptables -t mangle -A PREROUTING -p icmp -j DROP
```

This drops all ICMP packets. ICMP is only used to ping a host to find out if it's still alive. Because it's usually not needed and only represents another vulnerability that attackers can exploit, we block all ICMP packets to mitigate Ping of Death (ping flood), ICMP flood and ICMP fragmentation flood.

```
iptables -A INPUT -p tcp -m connlimit --connlimit-above 80 -j REJECT --reject-with
```

This iptables rule helps against connection attacks. It rejects connections from hosts that have more than 80 established connections. If you face any issues you should raise the limit as this could cause troubles with legitimate clients that establish a large number of TCP connections.

```
iptables -A INPUT -p tcp -m conntrack --ctstate NEW -m limit --limit 60/s --limit-
iptables -A INPUT -p tcp -m conntrack --ctstate NEW -j DROP
```

Limits the new TCP connections that a client can establish per second. This can be useful against connection attacks, but not so much against SYN floods because the usually use an endless amount of different spoofed source IPs.

```
iptables -t mangle -A PREROUTING -f -j DROP
```

This rule blocks fragmented packets. Normally you don't need those and blocking fragments will mitigate UDP fragmentation flood. But most of the time UDP fragmentation floods use a high amount of bandwidth that is likely to exhaust the capacity of your network card, which makes this rule optional and probably not the most useful one.

```
iptables -A INPUT -p tcp --tcp-flags RST RST -m limit --limit 2/s --limit-burst 2
iptables -A INPUT -p tcp --tcp-flags RST RST -j DROP
```

This limits incoming TCP RST packets to mitigate TCP RST floods. Effectiveness of this rule is questionable.

Mitigating SYN Floods With SYNPROXY

SYNPROXY is a new target of iptables that has been added in Linux kernel version 3.12 and iptables 1.4.21. CentOS 7 backported the feature and it's available in its 3.10 default kernel.

The purpose of SYNPROXY is to check whether the host that sent the SYN packet actually establishes a full TCP connection or just does nothing after it sent the SYN packet.

If it does nothing, it discards the packet with minimal performance impact.

While the iptables rules that we provided above already block most TCP-based attacks, the attack type that can still slip through them if sophisticated enough is a SYN flood.

It's important to note that the performance of the rules will always be better if we find a certain pattern or signature to block, such as packet length (`-m length`), TOS (`-m tos`), TTL (`-m ttl`) or strings and hex values (`-m string` and `-m u32` for the more advanced users).

But in some rare cases that's not possible or at least not easy to achieve. So, in these cases, you can make use of SYNPROXY.

Here are iptables SYNPROXY rules that help mitigate SYN floods that bypass our other rules:

```
iptables -t raw -A PREROUTING -p tcp -m tcp --syn -j CT --notrack
iptables -A INPUT -p tcp -m tcp -m conntrack --ctstate INVALID,UNTRACKED -j SYNPRO
iptables -A INPUT -m conntrack --ctstate INVALID -j DROP
```

These rules apply to all ports. If you want to use SYNPROXY only on certain TCP ports that are active (recommended – also you should block all TCP ports that are not in use using the mangle table and PREROUTING chain), you can just add `–dport 80` to each of the rules if you want to use SYNPROXY on port 80 only.

To verify that SYNPROXY is working, you can do `watch -n1 cat /proc/net/stat/synproxy`. If the values change when you establish a new TCP connection to the port you use SYNPROXY on, it works.

The Complete IPtables Anti-DDoS Rules

If you don't want to copy & paste each single rule we discussed in this article, you can use the below ruleset for basic DDoS protection of your Linux server.

```
### 1: Drop invalid packets ###
/sbin/iptables -t mangle -A PREROUTING -m conntrack --ctstate INVALID -j DROP
```

```
### 2: Drop TCP packets that are new and are not SYN ###
/sbin/iptables -t mangle -A PREROUTING -p tcp ! --syn -m conntrack --ctstate NEW -

### 3: Drop SYN packets with suspicious MSS value ###
/sbin/iptables -t mangle -A PREROUTING -p tcp -m conntrack --ctstate NEW -m tcpmss

### 4: Block packets with bogus TCP flags ###
/sbin/iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,SYN,RST,PSH,ACK,URG
/sbin/iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,SYN FIN,SYN -j DROP
/sbin/iptables -t mangle -A PREROUTING -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
/sbin/iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,RST FIN,RST -j DROP
/sbin/iptables -t mangle -A PREROUTING -p tcp --tcp-flags FIN,ACK FIN -j DROP
/sbin/iptables -t mangle -A PREROUTING -p tcp --tcp-flags ACK,URG URG -j DROP
/sbin/iptables -t mangle -A PREROUTING -p tcp --tcp-flags ACK,FIN FIN -j DROP
/sbin/iptables -t mangle -A PREROUTING -p tcp --tcp-flags ACK,PSH PSH -j DROP
/sbin/iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL ALL -j DROP
/sbin/iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL NONE -j DROP
/sbin/iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL FIN,PSH,URG -j DROP
/sbin/iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL SYN,FIN,PSH,URG -j D
/sbin/iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL SYN,RST,ACK,FIN,URG
```

```
### 5: Block spoofed packets ###
/sbin/iptables -t mangle -A PREROUTING -s 224.0.0.0/3 -j DROP
/sbin/iptables -t mangle -A PREROUTING -s 169.254.0.0/16 -j DROP
/sbin/iptables -t mangle -A PREROUTING -s 172.16.0.0/12 -j DROP
/sbin/iptables -t mangle -A PREROUTING -s 192.0.2.0/24 -j DROP
/sbin/iptables -t mangle -A PREROUTING -s 192.168.0.0/16 -j DROP
/sbin/iptables -t mangle -A PREROUTING -s 10.0.0.0/8 -j DROP
/sbin/iptables -t mangle -A PREROUTING -s 0.0.0.0/8 -j DROP
/sbin/iptables -t mangle -A PREROUTING -s 240.0.0.0/5 -j DROP
/sbin/iptables -t mangle -A PREROUTING -s 127.0.0.0/8 ! -i lo -j DROP

### 6: Drop ICMP (you usually don't need this protocol) ###
/sbin/iptables -t mangle -A PREROUTING -p icmp -j DROP

### 7: Drop fragments in all chains ###
/sbin/iptables -t mangle -A PREROUTING -f -j DROP

### 8: Limit connections per source IP ###
/sbin/iptables -A INPUT -p tcp -m connlimit --connlimit-above 111 -j REJECT --reje

### 9: Limit RST packets ###
```

```
/sbin/iptables -A INPUT -p tcp --tcp-flags RST RST -m limit --limit 2/s --limit-bu
/sbin/iptables -A INPUT -p tcp --tcp-flags RST RST -j DROP

### 10: Limit new TCP connections per second per source IP ###
/sbin/iptables -A INPUT -p tcp -m conntrack --ctstate NEW -m limit --limit 60/s --
/sbin/iptables -A INPUT -p tcp -m conntrack --ctstate NEW -j DROP

### 11: Use SYNPROXY on all ports (disables connection limiting rule) ###
# Hidden - unlock content above in "Mitigating SYN Floods With SYNPROXY" section
```

Bonus Rules

Here are some more iptables rules that are useful to increase the overall security of a Linux server:

```
### SSH brute-force protection ###
/sbin/iptables -A INPUT -p tcp --dport ssh -m conntrack --ctstate NEW -m recent --
/sbin/iptables -A INPUT -p tcp --dport ssh -m conntrack --ctstate NEW -m recent --

### Protection against port scanning ###
/sbin/iptables -N port-scanning
```

```
/sbin/iptables -A port-scanning -p tcp --tcp-flags SYN,ACK,FIN,RST RST -m limit --
/sbin/iptables -A port-scanning -j DROP
```

Conclusion

This tutorial demonstrates some of the most powerful and effective methods to stop DDoS attacks using iptables.

We've successfully mitigated DDoS attacks that peaked at **multiple million packets per second** using these iptables rules.

Every single guide on the same topic that we had researched provided inefficient methods to stop DDoS traffic or only a very limited number of iptables rules.

If used correctly, iptables is an extremely powerful tool that's able to block different types of DDoS attacks at line-rate of 1GigE NICs and close to line-rate of 10GigE NICs.

Don't underestimate the power of iptables!

SHARE THIS POST

66 thoughts on “DDoS Protection With IPtables: The Ultimate Guide”

JASON
2016-10-27 AT 16:04

Thank you for the free advice and very thorough discussion. I have one question about the limit rules:
iptables -A INPUT -p tcp -m conntrack –ctstate NEW -m limit –limit 60/s –limit-burst 20 -j ACCEPT
iptables -A INPUT -p tcp -m conntrack –ctstate NEW -j DROP

Will these rules effect a crawler, like GoogleBot, or Bing’s equivalent? I’m concerned about search engine visibility.

Thank you.

Reply

JAVAPIPE
2016-10-27 AT 16:54

Jason,

you're welcome. The rule you mention shouldn't affect any legitimate traffic. To make sure it really doesn't, I recommend that you use Siege https://www.joedog.org/siege-home/ or a similar tool to test if packets get dropped when an IP sends many HTTP requests within a short amount of time.

Reply

JASON

2016-10-27 AT 17:25

Hey, thanks for the link to Siege. That's going to be really helpful in a lot of ways!

Re: GoogleBot – I've re-read the rule and I get it. I was thinking in terms of minutes, not seconds. Thanks again!

Reply

TOM

2016-11-01 AT 22:07

Hey,

thanks for the very interesting and helping writeup. I´ve never configured iptables before and had to spend a lot of time to get at least a bit into it. I see that you really know what you are talking about.

It would be really nice if you could give me a feedback to what I came up with until now:

https://bbs.archlinux.org/viewtopic.php?id=218834

https://github.com/tiiiecherle/linux_scripts/blob/master/network_select_iptables_script_server.sh

Thanks in advance

Reply

JAVAPIPE
2016-11-02 AT 07:20

Hi Tom,

thank you for your feedback. I had a quick look at your script (sorry, but I don't have time to actually check every rule) and it seems to be fine. However, the iptables rules in this article are meant for DDoS mitigation. DDoS attacks are usually larger and you'd need a 10GigE NIC and uplink to mitigate even smaller ones. With the network setup you mention (100Mbps down, 12Mbps up), I believe that the bandwidth will become the bottleneck long before those iptables rules against DDoS could even really kick in. Most servers have a 1Gbps uplink nowadays, so even if only one single server is being used to launch an attack on you with spoofed SYN packets, your uplink will become the bottleneck. Therefore I suggest you try to implement the DDoS protection further upstream, ie. ask your ISP for options they can provide. This sounds like a home/office connection, so they probably won't have any though. Another alternative would be using a GRE or VPN tunnel to access the internet through a DDoS protected network such as JavaPipe's.

Reply

TOM

2016-11-02 AT 10:22

Hey,

thanks for the quick response. Yes, it is "only" a small business server. It would be very nice if you had some time if you could take a deeper look if the security limits and the order of the packets going through iptables are reasonable. The ports opening and closing is working fine. If you find anything else that I can do better I would appreciate a feedback. Otherwise I would leave it like that because I couldn`t find anything else when researching.

Thanks

Reply

JAVAPIPE

2016-11-02 AT 10:37

Hi Tom,

you're welcome to submit a request for a quote for going through and possibly correcting your rules at salesrequest@javapipe.com. But like I said before, they do look fine at first glance.

Reply

KEVIN

2016-11-04 AT 07:46

It's a wonderful sharing!

But I've one question about the iptables SYNPROXY rules, why the parameter is '-D' but not '-A'?
I'm not quite understand here ☹

""""

#/sbin/iptables -t raw -D PREROUTING -p tcp -m tcp –syn -j CT –notrack

#/sbin/iptables -D INPUT -p tcp -m tcp -m conntrack –ctstate INVALID,UNTRACKED -j SYNPROXY –sack-perm –timestamp –wscale 7 –mss 1460
#/sbin/iptables -D INPUT -m conntrack –ctstate INVALID -j DROP
“””

Thanks in advance!

Reply

JAVAPIPE
2016-11-04 AT 08:06

Thank you for pointing this out! Of course it’s supposed to be -A, not -D. We’ll update the article shortly.

Reply

YZORD
2016-11-04 AT 23:23

I have a question. Can i still use your sysctl.conf with an OpenVZ VPS? Because that one uses an older kernel related to the hardware node (openvz kernel). And yours is using one of the newer kernels.

Reply

JAVAPIPE

2016-11-05 AT 07:29

When you run sysctl -p it will show errors if a setting isn't supported by your kernel. So simply try it and adjust the settings that don't work with your kernel accordingly.

Reply

SHAD DESIGNER

2016-12-05 AT 13:14

how can block the attacker's IP automatically ?
is there any direct command over the Iptable could be used directly??

thx

Reply

MALAD

2017-04-13 AT 19:50

for temporary or permanent blocking i assume you would need tools like DenyHosts, Fail2ban or better “CSF”

Reply

VASILIS M

2017-01-09 AT 19:12

this kernel settings working in Debian 7.3.0 ? thanks

Reply

JAVAPIPE

2017-01-09 AT 19:51

It’s has only been tested with CentOS 7. I think it would work with Debian 8, but with Debian 7 you’d need to run kernel > 3.12 and iptables > iptables 1.4.21

Reply

VASILIS M

2017-01-09 AT 20:55

kernel == 3.2.0-4 iptables == 1.4.14

should i try ?

Reply

JAVAPIPE

2017-01-09 AT 21:03

At least iptables SYNPROXY rules won't work with that – sysctl.conf might. You can't break anything with unsupported kernel settings, worst case you just get a message that it's an unknown one, so you could just give it a shot.

Reply

FUGITIVE90

2017-01-19 AT 02:04

Hi! Thanks for great guide. Do you mind explaining why bellow rules aren't placed in mangle Prerouting chain(speaking about performances)?
iptables -A INPUT -p tcp -m tcp -m conntrack –ctstate INVALID,UNTRACKED -j SYNPROXY –sack-perm –timestamp –wscale 7 –mss 1460
iptables -A INPUT -m conntrack –ctstate INVALID -j DROP

Reply

JAVAPIPE

2017-01-19 AT 07:51

If I recall correctly the SYNPROXY target didn't work within the MANGLE table. I could be wrong though, please feel free to try this and share the results.

Reply

NITROUS OXIDE

2017-03-09 AT 08:03

Thanks for the article but I think there's some errors. "Limit new TCP connections" rules have nothing to do with source IP, right? To limit source IP you need something like hashlimit. "Protection against port scanning" ruleset seems to work only if your kernel is ancient. "Block packets with bogus TCP flags" ruleset is full of redundant rules that consumes extra resources.

Reply

JAVAPIPE

2017-05-12 AT 08:25

Thanks for pointing this out. We'll work on double-checking and improving these rules when we have the capacity. These are more of a PoC.

Reply

COLLISION
2019-03-31 AT 01:03

Dear JP,

Could you include an example with hashlimit please?

Best Regards

Collision

Reply

HAJES
2019-01-20 AT 10:09

my first thought… – m limit has no idea about per IP attack and blocks all connections no matter legit or not.

Reply

WEBER K.

2017-03-10 AT 01:40

Hi! FIN,SYN and SYN,FIN are duplicated? Why?

Reply

BOLTRONICS

2018-03-21 AT 03:07

It looks like FIN,SYN,RST,PSH,ACK,URG NONE is a duplicate of ALL NONE as well. No big deal though.

Reply

JOHN

2018-03-22 AT 09:33

Yes, the rules aren’t perfect, but merely an example of how DDoS can be blocked with iptables. Thanks for pointing this out.

Reply

HAROLD JAMES QUILANG

2017-03-29 AT 03:48

what connection limit rule i will disable for #11 synproxy? is it #8 and #10?

Reply

JAVAPIPE

2017-05-12 AT 08:24

It will break #8, #9 and #10. You don’t need to manually disable them.

Reply

SH

2017-04-05 AT 07:13

I’m amazed that in 2016, well now 2017, people are still blocking ICMP and break path MTU discovery…

Reply

MALAD

2017-04-13 AT 19:47

Why? Mind explaining please?

Reply

KRZYSZTOF PAWLUCH

2017-04-20 AT 07:59

MTU Discovery is based on ICMP packets – if you block all ICMP traffic you can create blackholes in your network. If you have don't fragment bit set in your IP header and send too big packets (for example if you have tunnel somewhere) then sender will be not informed about it – packets will be dropped.

For UDP connections all traffic will be send but none of packets will reach destination and for TCP you will have a lot of retransmissions.

One of solutions for TCP is to set MSS size in TCP SYN – it can be done via iptables. MSS size informs other side about MSS. You need to calculate MSS, but it is very simple.

Other solution for TCP is to set MSS via setsockopt (in linux).
For other L4 protocols you will have to fragment manually.

Good option is to set MTU and MSS at the entry point of tunnels or other points where MTU is smaller and enable ICMP type 3 and 4 (Fragmentation needed, which also contains proper MTU)

But blocking all ICMP traffic is considered a bad idea, especially ICMPv6 – this is definitely bad idea.

Reply

JAVAPIPE

2017-05-12 AT 08:22

If it comes down to downtime vs. breaking PMTUD, what would you prefer? These IPtables rules are a last resort when you're under attack or are likely going to be and don't have better means of defense – not a best practice ruleset you should put on every machine.

Reply

BRANDON HEARD

2017-04-14 AT 21:56

Is there something in this that would prevent me from connecting through SSH?

Reply

JAVAPIPE

2017-05-12 AT 08:30

No, as long as you don't change anything. Wrong SYNPROXY rules could mess things up.

Reply

DENNA

2017-04-21 AT 03:10

Everywhere I've read and each time I've asked, I've been told never to put filtering rules in the mangle table. If you do, bad, but never described things will happen. ☹ However, in this article that is the solution. What is the problem with putting filtering rules in the mangle table ?

Question – Can you block Internet hosts from initiating a NEW state connection to / through a router with the rule below without blocking reply connections that were initiated by the router (OUTPUT chain) or internal client (FORWARD chain) ?

iptables -t mangle -I PREROUTING -i wan_face -m state –state NEW -j DROP

Reply

JAVAPIPE

2017-05-12 AT 08:30

Putting them in the mangle table is indeed “wrong” and might lead to unexpected behaviour with other rules. Like the rule you mentioned is going to block packets in FORWARD as well, when the same rule in filter table and INPUT wouldn’t.

Reply

DENNA

2017-05-13 AT 21:38

1) Is placing your anti-DDoS rules in the mangle table / PREROUTING chain more efficient due to the assumption that non-anti-DDoS rules will be unnecessarily processed in the tables and chains prior to the filter table causing unnecessary CPU and memory utilization ?

2) I want to block Internet initiated traffic on the INPUT and FORWARD chains. The rule above would be more efficient than creating two separate rules for the INPUT and FORWARD chains, correct ?

3) When a packet traverses two or more tables, does that packet use up more memory in the conntrack table than if it was blocked in the first table ?

Reply

JAVAPIPE

2017-05-14 AT 06:58

The answer to all of your questions is likely “yes”. You’ll have to test this yourself.

Reply

EMANUELE APOLLONIO

2017-05-21 AT 03:44

After this i can’t usa SSH and my Sinusbot web panel is not work, but the ts3 server is on… any solution?

Reply

JAVAPIPE

2017-09-23 AT 05:16

Did you use synproxy on all ports? Try using it only on your TS3 port if you run it on TCP.

Reply

ALI SHIRVANI

2017-07-02 AT 07:22

why this line:
/sbin/iptables -A INPUT -p tcp -m tcp -m conntrack –ctstate INVALID,UNTRACKED -j SYNPROXY –sack-perm –timestamp –wscale 7 –mss 1460
blocks ssh connection?

Reply

JAVAPIPE

2017-09-23 AT 05:17

Pretty sure it didn’t when we tested this. Instead of using synproxy on all ports I suggest using it only on ports that run the applications that are being attacked.

Reply

AMMARZ

2017-09-22 AT 19:52

I am using CentOs 7, VPS is getting constantly attacked, IP doesnt work, server dies, website dies. Some kind of IP stresser is causing this problem. I am sick of this. Will it prevent such attacks?

Reply

JAVAPIPE

2017-09-23 AT 05:15

You can definitely try, but usually your hosting provider will become an issue if they don't offer DDoS protection. Also these rules only help against some types of attacks and by far not all. I suggest you check out our unmetered VPS plans at https://javapipe.com/unmetered-vps/ which all include DDoS protection.

Reply

JUSTAS JOCAS

2017-10-08 AT 13:30

Port scanning doenst work, i can still easily get all ports

Reply

MUSTANG

2017-11-04 AT 21:54

Will this help on a machine running OpenVPN server or will it break it?

Reply

MUSTANG

2017-11-05 AT 01:00

Will this help secure a server that has OpenVPN running on it or will this end up breaking OpenVPN

Reply

TITI

2017-11-23 AT 15:42

Hey, According to me the rule :

### 1: Drop invalid packets ###

iptables -t mangle -A PREROUTING -m conntrack –ctstate INVALID -j DROP

Is not compatible with your SYNPROXY configuration.

iptables -t raw -A PREROUTING -p tcp -m tcp –syn -j CT –notrack

iptables -A INPUT -p tcp -m tcp -m conntrack –ctstate INVALID,UNTRACKED -j SYNPROXY –sack-perm –timestamp –wscale 7 –mss 1460

iptables -A INPUT -m conntrack –ctstate INVALID -j DROP

Wich make sens because PREROUTING mangle will occurs between PREROUTING-raw and INPUT-filter. So the prerouting rules will DROP the packet before it reach this test and a chance to be ACCEPTED?
iptables -A INPUT -p tcp -m tcp -m conntrack –ctstate INVALID,UNTRACKED -j SYNPROXY –sack-perm –timestamp –wscale 7 –mss 1460

Or am i wrong ? Can’t use both on my server or every packets gets DROP.

Reply

LINUS REIBER

2018-04-07 AT 07:42

My Sinusbot webinterface is not working how can i do synproxy only on this ports?

Reply

GIOVANNI

2018-04-19 AT 07:24

i know i might be a bit late to this but what is the -j CT used for cause it seems to be the only instance to be in this article and before anyone starts asking yes i am new to this

Reply

WAKWEK

2018-04-19 AT 22:20

can i use this in ubuntu?

Reply

PAULO

2018-04-24 AT 14:59

Very interesting article! do you already had time to test this setting on distros based on Debian? Thank you!

Reply

AMM

2018-06-09 AT 03:20

When you do:

iptables -t mangle -A PREROUTING -m conntrack –ctstate INVALID -j DROP

It automatically blocks all invalid / bogus tcp-flags. So all those tests of tcp-flags is not required.

Also if you still insist on keeping them then this rule is not required.

```
/sbin/iptables -t mangle -A PREROUTING -p tcp –tcp-flags ALL ALL -j DROP
```

Because preceding rule

```
/sbin/iptables -t mangle -A PREROUTING -p tcp –tcp-flags FIN,SYN FIN,SYN -j DROP
```

Takes care of it.

Reply

JOHN

2018-06-14 AT 10:13

Thanks!

Reply

TAISN

2018-08-18 AT 10:56

Good article. Can you please mention which tool should be used to simulate DoS/ DDoS attack?

Reply

JOHN

2018-08-23 AT 08:46

You could use hping for that.

Reply

MATTHIAS

2018-09-30 AT 07:35

PREROUTING of raw table is the first chain checked and also capable of SYNPROXY target:

https://serverfault.com/questions/933200/iptables-questions-regarding-the-raw-table

Reply

GONDIM

2018-11-01 AT 18:39

Hi,

Very good this article!

Could you update this article with rules for IPv4 (iptables) and IPv6 (ip6tables)?

Reply

MERT ATAKAN CENIKUT

2018-11-15 AT 02:51

Hi,

I agree with Gondim!

Could you update this article with rules for IPv4 (iptables) and IPv6 (ip6tables)?

Reply

NOMADEWOLF

2018-12-31 AT 02:05

Can’t access Webmin after aplying these rules… any ideas?

Reply

CHARLIE G MARCENZI

2019-01-30 AT 18:38

Our system we use Debian as routers connected to ISP with 10 Gbps, and using QUAGGA for BGP routing externally and OSFP internely and iptables.
In you example you only reefer to INPUT but for our routers we need to block dos/ddos attacks on the FORWARD chain.
Is there any difference to use same config for forward chain…

Reply

İBRAHIM

2019-04-02 AT 05:08

I want to mention that, if you located a firewall in front of several servers and put syn proxy on forward chain, I suggest you set “net.ipv4.tcp_window_scaling=0” on sysctl because window scale may differ on behind servers.

Reply

USER-DEB

2019-05-17 AT 00:13

I tested these script at Centos 7 and not work , blocking my connection ssh by putty . NOT WORK

Reply

DALTON

2019-06-01 AT 04:51

Using the “The Complete IPtables Anti-DDoS Rules” list:

Rules 2 and 3 both individually stop SSH and HTTP connections from being established.

SYNPROXY rules stop SSH and HTTP connections from being established.

I came here mostly for the SYN protection, but the SYN rules provided here stop all types of connections from even being made.

Reply

H-MMER

2019-06-05 AT 13:27

Hi,

I’ve got the same problem as the person above, once i applied all these rules im unable to SSH to the server or even use HTTP/HTTPS connections to the server and dont seem to understand why it blocks them.

Reply

HARALD REINDL

2019-06-27 AT 23:40

-m limit is pure nonsense – it’s a simple self DOS because it has no concept of source ip’s

xt_recent is your friend

and Xmas packets should be dropped in – t raw PREROUTING given that mangle is stateful

for real protect against dos you want xt_recent in mangle, escalate over-limit clients to a ipset with automatic timeout and drop everything listet in the ipset as first inbound rule in your stateless raw table

some ideas here are nice but very sloppy and most are not effective

Reply

Leave a Comment

Your email address will not be published. Required fields are marked *

Type here..

Name*

Email*

Post Comment »

JavaPipe LLC

12180 South 300 East #502

Draper, Utah 84020

USA

+1-800-918-1890

salesrequest@javapipe.com