English • Deutsch • Español • Français • Italiano • ??? • Polski • Português • ??????? • Svenska • ???(????)? • ???(??)? •

Contents

Preventing Brute Force Attacks

Feel free to apply either the basic SSH protection or skip to the advanced section for a more elegant and flexible way of implementing this kind of protection.

Introduction

Thanks to the suggestion by JoFa, and the legwork, testing and code from jbarbieri, you can now rest easy in having SSH, as well as other service ports open on the WAN portion of your router for ease of use, while severely hampering the attempts of a brute force hack attempt. Remember, always use unique and long passwords!

Prerequisites

- **Version**: theoretically any DD-WRT; tested on v24sp1+ [1]
- **Builds**: VPN, VoIP, Standard, Mega, Mini. Basically anything except Micro and Micro+.

The First Method: Basic Protection

Implementing Basic Protection for SSH (outdated)

This has been integrated into all builds (exept micro) >01.2011. See firewall page.

UPDATE As of K26 builds with dates above December 2010, a better solution has been implemented using the 'recent' module. It can be deployed in two different ways, either via rc_firewall scripting, or through

Frater's Optware, the Right Way using his ingenious 'fixtables' script. If running OTRW, there is no need to update, as DD-WRT's netfilter firewall has had this option the entire time, just not compiled as a module.

If not running the "fixtables" script, the proper procedure is as follows as placed in rc_firewall:

```
iptables -N bruteprotect
iptables -A bruteprotect -m recent --set --name BRUTEFORCE --rsource
iptables -A bruteprotect -m recent ! --update --seconds 60 --hitcount 4 --name BRUTEFORCE --rsour
iptables -A bruteprotect -j LOG --log-prefix "[DROP BRUTEFORCE] : " --log-tcp-options --log-ip-op
iptables -A bruteprotect -j DROP

iptables -I FORWARD 4 -i vlan2 -p tcp -m tcp --dport 22 -j bruteprotect
iptables -I INPUT 3 -i vlan2 -p tcp -m tcp --dport 21:23 -j bruteprotect
```

NOTE For protecting multiple open ports, it is necessary to use the - m multiport matcher (if it exists in the kernel, which as of 01/17/2011 it does).

Example for protecting port 3389:

```
iptables -I INPUT 2 -i vlan2 -p tcp -m multiport --dports 21,22,23,3389 -j bruteprotect
```

Here's what it looks like:

INPUT chain

```
 Code:
-A INPUT -i vlan2 -p tcp -m tcp --dport 22 -j bruteprotect
-A INPUT -d 192.168.10.1 -p tcp -m tcp --dport 22 -j logaccept
```

```
FORWARD chain
Code:
-A FORWARD -i vlan2 -p tcp -m tcp --dport 5900 -j bruteprotect
-A FORWARD -i vlan2 -p tcp -m tcp --dport 3389 -j bruteprotect
-A FORWARD -i vlan2 -p tcp -m tcp --dport 22 -j bruteprotect
-A FORWARD -i vlan2 -p tcp -m tcp --dport 21 -j bruteprotect

bruteprotect chain
Code:
-A bruteprotect -m recent --set --name BRUTEFORCE --rsource
-A bruteprotect -m recent ! --update --seconds 60 --hitcount 4 --name BRUTEFORCE --rsource -j RET
-A bruteprotect -j LOG --log-prefix "[DROP BRUTEFORCE] : " --log-tcp-options --log-ip-options
-A bruteprotect -j DROP
```

More information can be found here:

http://www.debian-administration.org/articles/187

Warning! *This is a bit outdated. It will work, but try the solution in the final section - that's the more elegant way to do it.*

Note that this protection *example* is designed to prevent connections on port 22 (SSH). It will not cover telnet (it's better to disable that anyway), nor VPN (PPTP/OpenVPN). It will not work on Micro(+) builds (as of SVN 10431).

Go to the Administration -> Commands page of your router's GUI. Paste the following code in the script box the click Save Firewall:

```
iptables -I INPUT 2 -p tcp --dport 22 -m state --state NEW -m limit --limit 3/min --limit-burst 3
iptables -I INPUT 3 -p tcp --dport 22 -j logreject
```

This code shown here was made for limiting connection attempts to port 22 (SSH) to only 3 per minute. You can adapt this code to implement other ports, change the amount of attempts before timeout, change the length of the timeout, etc.

Protecting Telnet in Addition to SSH

The above code can be extended to protect against multiple connections on various other ports. For instance, assuming you want to protect your SSH and telnet enabled router, you could use the following code:

```
iptables -I INPUT 2 -p tcp --dport 22 -m state --state NEW -m limit --limit 3/min --limit-burst 3
iptables -I INPUT 3 -p tcp --dport 23 -m state --state NEW -m limit --limit 3/min --limit-burst 3
iptables -I INPUT 4 -p tcp --dport 22 -j logreject
iptables -I INPUT 5 -p tcp --dport 23 -j logreject
```

As you can see, extending the script is not difficult; all that you must do is make sure the lines containing RELATED are first, then the ones containing NEW then the logreject lines.

Applying the concept to forwarded ports

Say you have an FTP server running on a LAN machine. You can forward port 21 to the internal server, protecting it using the above concept.

```
iptables -t nat -I PREROUTING -p tcp -d $(nvram get wan_ipaddr) --dport 21 -j DNAT --to 192.168.1
iptables -I FORWARD -p tcp -d 192.168.1.101 --dport 21 -m state --state RELATED,ESTABLISHED -j AC
iptables -I FORWARD 2 -p tcp -d 192.168.1.101 --dport 21 -m state --state NEW -m limit --limit 3/
iptables -I FORWARD 3 -p tcp -d 192.168.1.101 --dport 21 -j logreject
```

Note: In this example, 192.168.1.101 is the IP address of the LAN machine that is running the FTP server.

The Advanced Method: Protection for Any Open Port

The Elegant Way of Implementing Protection

First, the **rate_limit** chain has to be created, and the rules entered into it:

```
iptables -N rate_limit
iptables -F rate_limit
iptables -A rate_limit -p tcp --dport 22 -m limit --limit 3/min --limit-burst 3 -j ACCEPT
```

```
iptables -A rate_limit -p udp --dport 1194 -m limit --limit 3/min --limit-burst 3 -j ACCEPT
iptables -A rate_limit -p ICMP --icmp-type echo-request -m limit --limit 3/sec -j ACCEPT
iptables -A rate_limit -p <protocol> --dport <port> -m limit --limit <x/sec/min/hr> --limit-burst
iptables -A rate_limit -p ! ICMP -j LOG --log-prefix " Connection dropped!! "
iptables -A rate_limit -p tcp -j REJECT --reject-with tcp-reset
iptables -A rate_limit -p udp -j REJECT --reject-with icmp-port-unreachable
iptables -A rate_limit -j DROP
```

To actually make use of this chain, you just need to add the rules in the **INPUT** chain, which makes any new connection go to the **rate_limit** chain:

```
iptables -I INPUT -p ICMP --icmp-type echo-request -j rate_limit
iptables -I INPUT -p tcp --dport 22 -m state --state NEW -j rate_limit
iptables -I INPUT -p udp --dport 1194 -m state --state NEW -j rate_limit
iptables -I INPUT -p <protocol> --dport <port> -m state --state NEW -j rate_limit
```

This sends any new connection for SSH or OpenVPN to the rate_limit chain. You can replace the <...> parameters with the actual protocol, port, and accepted rate and to have it go through the rate_limit as well. This script also limits ICMP pings, to 3 per second... any faster and it blocks them. Since ping floods are common, the LOG statement was changed so that if the protocol is ICMP, it will NOT log, which won't cause your router to bog down.

Last Resort

If the above has not the desired effect you can block IP addresses permanently with the following script which scans /var/log/messages for failed login attempts. It then takes the last entry and blocks the corresponding IP address. Just put it in crontab and have it run every few minutes.

```
#!/bin/sh

############################################################
#
# check_brute_force
# Checks for failed logins and blocks IP addresses
#
############################################################

IP=`awk -F'[ :]' '/login attempt/ {print $(NF-1)}' /var/log/messages | tail -1`
rc=0

# Do nothing if there is an existing rule for this IP address
if `iptables -L -n | grep $IP > /dev/null 2>&1`; then
    exit 0
fi

case $IP in
    "") # Do nothing with empty IP
    ;;
    192.168*) # Exclude local LAN
    ;;
    *) # Add rule against intruding IP
    iptables -I INPUT -s $IP -j DROP
    RC=$?
    ;;
esac
```

```
exit $RC

# EOF
```

You can check for permanently banned IP addresses with the following script:

```
#!/bin/sh

############################################################
#
# show_blocked_ip
#
# Shows explicitly blocked IP addresses
#
############################################################

IP=`iptables -L -n | awk '$4~/[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}/ && $4!~/0\.0\.0\.0/

if [ "$IP" == "" ]; then
    echo "No blocked IP addresses found."
else
    echo "Blocked IP addresses:"
    for n in $IP; do
        echo $n
    done
fi

exit 0

# EOF
```