# IPv4

- Firewall Setup
- DHCP and DNS Cache

# Firewall Setup

## Install Shorewall

To manage `nftables/iptables` I decided to go with Shorewall since it is easy to configure and very mature. At some point I may look into switching to FireHol since it looks even simpler to configure but I wanted something I knew I'd be able to make do everything I needed.

I started by installing *shorewall* as my firewall, *shorewall-doc* which includes examples, and *shorewall-init* which can lockdown the system at boot before *Shorewall* has had a chance to configure the firewall.

```
# apt install shorewall shorewall-doc shorewall-init
```

Then I update the *shorewall* configuration to reflect that I'm using *ulogd2* for logging and that I want IPv4 forwarding enabled when *shorewall* starts.

```
# /etc/shorewall/shorewall.conf
- LOG_LEVEL="info"
+ LOG_LEVEL="NFLOG(1,0,1)"
...
- LOGFILE=/var/log/messages
+ LOGFILE=/var/log/firewall.log
...
- IP_FORWARDING=Keep
+ IP_FORWARDING=Yes
```

All my configuration files are adapted from the examples that *shorewall-doc* makes available under `/usr/share/doc/shorewall/examples`.

Setting up the zones is pretty self-explanitory. The only addition I made is I have a `warp` zone which I will use later when I am setting up my VPN.

```
# /etc/shorewall/zones
+ #------------------------------------------------------------------------------
+ # For information about entries in this file, type "man shorewall-zones"
+ #
```

```
+ # See http://shorewall.net/manpages/shorewall-zones.html for more information
+ ###############################################################################
+ #ZONE   TYPE    OPTIONS                 IN                      OUT
+ #                                       OPTIONS                 OPTIONS
+ fw      firewall
+ wan     ipv4
+ lan     ipv4
+ dmz     ipv4
+ warp    ipv4
```

Setting up the interfaces and assiging them zones is also pretty self-explanatory.

```
# /etc/shorewall/interfaces
+ #------------------------------------------------------------------------------
+ # For information about entries in this file, type "man shorewall-interfaces"
+ #
+ # See http://shorewall.net/manpages/shorewall-interfaces.html for more information
+ ###############################################################################
+ ?FORMAT 2
+ ###############################################################################
+ #ZONE□□INTERFACE       OPTIONS
+ wan□□WAN_IF□□□tcpflags,dhcp,nosmurfs,routefilter,logmartians,sourceroute=0,physical=eth0
+ lan□□LAN_IF□□□tcpflags,dhcp,nosmurfs,routefilter,logmartians,physical=eth1
+ dmz□□DMZ_IF□□□tcpflags,dhcp,nosmurfs,routefilter,logmartians,physical=eth1.8
+ warp□□WARP_IF□□□tcpflags,dhcp,nosmurfs,routefilter,logmartians,physical=eth1.9
```

My real `/etc/shorewall/policy` file is less liberal than what is shown below (`lan` being allowed to access whatever it wants) but I wanted to show a reasonably secure policy that allowed me to have a very simple `/etc/shorewall/rules` config below.

```
# /etc/shorewall/policy
+ #------------------------------------------------------------------------------
+ # For information about entries in this file, type "man shorewall-policy"
+ #
+ # See http://shorewall.net/manpages/shorewall-policy.html for more information
+ ###############################################################################
+ #SOURCE□DEST□□POLICY□□LOGLEVEL□RATE    CONNLIMIT
+
+ $FW□□all□□□ACCEPT
+ lan□□all□□□ACCEPT
+ dmz□□$FW,wan□□ACCEPT
```

```
+ warp□□$FW□□□ACCEPT
+
+ wan□□all□□□DROP□□$LOG_LEVEL
+ # THE FOLLOWING POLICY MUST BE LAST
+ all□□all□□□REJECT□□$LOG_LEVEL
```

Because my example policy is pretty open, my rules in this example are pretty sparse.

```
# /etc/shorewall/rules
+
#------------------------------------------------------------------------------------------
---------------
+ # For information about entries in this file, type "man shorewall-rules"
+ #
+ # See http://shorewall.net/manpages/shorewall-rules.html for more information
+
##############################################################################################
##############################################################################################
##########
+ #ACTION         SOURCE          DEST            PROTO   DEST    SOURCE
ORIGINAL        RATE          USER/    MARK    CONNLIMIT       TIME
HEADERS         SWITCH          HELPER
+ #                                                       PORT    PORT(S)
DEST            LIMIT         GROUP
+ ?SECTION ALL
+ ?SECTION ESTABLISHED
+ ?SECTION RELATED
+ ?SECTION INVALID
+ ?SECTION UNTRACKED
+ ?SECTION NEW
+
+ #       Don't allow connection pickup from the net
+ Invalid(DROP)   wan             all             tcp
+
+ DNS(ACCEPT)     all!wan,warp    $FW
+ DNS(ACCEPT)     $FW,dmz         lan:10.0.1.2
+
+ Web(ACCEPT)     dmz             $FW
+ Web(DNAT)       wan             dmz:10.0.8.2
```

Lastly is the magic that allows private addresses to access the Internet by masquerading them all as my one public IPv4 address I am assigned. The following just says all traffic heading out of `WAN_IF` (`eth0`) coming from a private IP range should be masqueraded.

```
# /etc/shorewall/snat
+ #------------------------------------------------------------------------------
+ # For information about entries in this file, type "man shorewall-snat"
+ #
+ # See http://shorewall.net/manpages/shorewall-snat.html for more information
+
###############################################################################################################
#############################################
+ #ACTION                 SOURCE                  DEST            PROTO   PORT    IPSEC
MARK    USER    SWITCHORIGDEST PROBABILITY
+ MASQUERADE              10.0.0.0/8,\
+                         169.254.0.0/16,\
+                         172.16.0.0/12,\
+                         192.168.0.0/16          WAN_IF
```

Now that I have everything configured it might be wise to run `shorewall check` just to make sure I didn't have any typos.

I hooked *shorewall* into the boot process to make sure the system is secure during boot by enabling *shorewall-init.service* and *shorewall.service*. First I told *shorewall-init* that it needs to account for *shorewall* when it runs.

```
# /etc/default/shorewall-init
- PRODUCTS=""
+ PRODUCTS="shorewall"
```

Then I simply told those services to start at boot.

```
# systemctl enable shorewall
# systemctl enable shorewall-init
```

# Modify Interfaces

Now that *Shorewall* will secure everything at bootup it is safe to update `/etc/networking/interfaces` and add their IPv4 addresses.

```
# /etc/networking/interfaces
 auto eth1
- iface eth1 inet manual
+ iface eth1 inet static
+         address 10.0.1.1/21

  auto eth1.8
- iface eth1.8 inet manual
+ iface eth1.8 inet static
          vlan-raw-device eth1
+         address 10.0.8.1/24

  auto eth1.9
- iface eth1.9 inet manual
+ iface eth1.9 inet static
          vlan-raw-device eth1
+         address 10.0.9.1/24
```

Now if I reboot the system all my interfaces will come up configured and the system will be protected by *nftables/iptables* configured by *Shorewall*.

> Be sure to sanity check the configuration so Shorewall doesn't block SSH access if that is needed.

```
# reboot
```

# DHCP and DNS Cache

# Install dnsmasq

I decided to use dnsmasq since it can fulfull multiple roles as both a DHCP and DNS cache. I'll first configure it for IPv4 and then later add in the few extra IPv6 lines needed.

# Setup DHCP

The following can look complicated but that is just becuase there are a ton of MAC Addresses and IP Addresses mixed throughout. If you look closely you can see that there are only four types of lines.

1. `no-dhcp-interface=eth0,lo` prevents DHCP binding on our loopback address and `eth0` which is the interface facing the Internet.
2. `dhcp-range=` declares a start and stop address and lease lifetime for each subnet. I am also setting an optional tag for each so I can target them later if I want.
3. `dhcp-option=` allows me to set specific DHCP options. The `tag:` allows me to target addresses matching a specific tag. I am overriding the default DNS servers because I want `lan` and `dmz` to use my *Pi-hole* server and `warp` should use a public DNS server since any device on that subnet is routed through a VPN tunnel so it doesn't have local network access.
4. `dhcp-host=` defines what IP addresses and hostnames get assigned to which network device with a specific MAC address

```
# /etc/dnsmasq.d/dhcp.conf
+ no-dhcp-interface=eth0,lo
+
+ dhcp-range=set:lan,10.0.5.1,10.0.7.254,12h
+ dhcp-range=set:dmz,10.0.8.1,10.0.8.254,12h
+ dhcp-range=set:warp,10.0.9.1,10.0.9.254,5m
+
+ dhcp-option=tag:lan,option:dns-server,10.0.1.2
+ dhcp-option=tag:lan,option:dns-server,10.0.1.2
+ dhcp-option=tag:warp,option:dns-server,1.1.1.1,1.0.0.1
+
```

```
+ # LAN - network infrastructure
+ dhcp-host=aa:af:57:f3:4e:90,10.0.1.2,pihole□□□# pihole
+ dhcp-host=b4:fb:e4:8f:f9:74,10.0.1.3,unifi-switch-8□# unifi-switch-8
+
+ # LAN  - proxmox
+ dhcp-host=e0:d5:5e:63:fe:30,10.0.3.2,blackbox□□□# blackbox
+ dhcp-host=70:85:c2:fe:4c:b7,10.0.3.3,mini□□□□# mini
+ dhcp-host=6e:91:84:4a:74:f1,10.0.3.4,backup□□□# backup
+
+ # LAN - assigned devices
+ dhcp-host=d0:a6:37:ed:8c:7f,10.0.4.4,silverbook□□# silverbook
+ dhcp-host=82:13:00:9c:c7:00,10.0.4.5,thunderbolt□□# thunderbolt
+ dhcp-host=34:36:3b:7f:18:1e,10.0.4.8,jess□□□□# jess
+ dhcp-host=96:64:5F:1C:A6:2C,10.0.5.6,refuge□□□# refuge
+ dhcp-host=7A:BC:46:D1:A3:1B,10.0.5.9,unifi□□□# unifi
+
+ # DMZ - assigned devices
+ dhcp-host=62:59:92:A7:1D:F1,10.0.8.5,bitcoin□□□# bitcoin
+ dhcp-host=32:cc:fb:a3:1a:57,10.0.8.2,contained□□# contained
```

# Setup DNS Caching

Everything here is commented with an explanation of what it does. The only thing slightly interesting is I have two `server=` parameters pointing to the IPv4 loopback addresses which is where *Unbound* is listening. If *Unbound* wasn't being used I'd either remove `no-resolv` and use the system namesevers or change the `server=` parameters to point to a public recursive name sever.

```
# /etc/dnsmasq.d/dns.conf
+ # Add the domain to simple names (without a period) in /etc/hosts in the same way as for
DHCP-derived names.
+ expand-hosts
+
+ # Log the results of DNS queries handled by dnsmasq.
+ log-queries
+
+ # Do not listen on the specified interface.
+ except-interface=eth0,lo
+
+ # Accept DNS queries only from hosts whose address is on a local subnet, ie a subnet for
```

```
which an interface exists on the server.
+ local-service
+
+ # Dnsmasq binds the address of individual interfaces, allowing multiple dnsmasq instances, but if new interfaces or addresses appear, it automatically listens on those
+ bind-dynamic
+
+ # Return answers to DNS queries from /etc/hosts and --interface-name which depend on the interface over which the query was received.
+ localise-queries
+
+ # All reverse lookups for private IP ranges (ie 192.168.x.x, etc) which are not found in /etc/hosts or the DHCP leases file are answered with "no such domain"
+ bogus-priv
+
+ # Later versions of windows make periodic DNS requests which don't get sensible answers from the public DNS and can cause problems by triggering dial-on-demand links.
+ filterwin2k
+
+ # Enable code to detect DNS forwarding loops
+ dns-loop-detect
+
+ # Reject (and log) addresses from upstream nameservers which are in the private ranges.
+ stop-dns-rebind
+
+ # Exempt 127.0.0.0/8 and ::1 from rebinding checks.
+ rebind-localhost-ok
+
+ # Tells dnsmasq to never forward A or AAAA queries for plain names, without dots or domain parts, to upstream nameservers.
+ domain-needed
+
+ # Specifies DNS domains for the DHCP server.
+ domain=hermz.io
+
+ # Don't read /etc/resolv.conf. Get upstream servers only from the command line or the dnsmasq configuration file.
+ no-resolv
+
+ server=127.0.0.1
```

```
+ server=::1
```

# Resolve Static Clients

One problem I ran into was that static clients never use DHCP so the DHCP server doesn't register their hostname with their intended IP address. To work around this limitation I just added those entries to the `/etc/hosts` file since by default *dnsmasq* will resolve using those entries too.

```
# /etc/hosts
+ 10.0.1.1        ember
+ 10.0.1.2        pihol
+ 10.0.1.3        unifi-switch-8
+ 10.0.3.2        blackbox
+ 10.0.3.3        mini
+ 10.0.3.4        backup
+ 10.0.3.5        edge


  # --- BEGIN PVE ---
```

# Reboot

Now that *dnsmasq* is fully configured I just restart it using *systemctl*

```
# systemctl restart dnsmasq.service
```