NAME

ufw−framework − using the ufw framework

DESCRIPTION

ufw provides both a command line interface and a framework for managing a netfilter firewall. While the **ufw** command provides an easy to use interface for managing a firewall, the **ufw** framework provides the administrator methods to customize default behavior and add rules not supported by the command line tool. In this way, **ufw** can take full advantage of Linux netfilter's power and flexibility.

OVERVIEW

The framework provides boot time initialization, rules files for adding custom rules, a method for loading netfilter modules, configuration of kernel parameters and configuration of IPv6. The framework consists of the following files:

/lib/ufw/ufw−init
: initialization script

/etc/ufw/before.init
: initialization customization script run before ufw is initialized

/etc/ufw/after.init
: initialization customization script run after ufw is initialized

/etc/ufw/before[6].rules
: rules file containing rules evaluated before UI added rules

/etc/ufw/user[6].rules
: rules file containing UI added rules (managed with the **ufw** command)

/etc/ufw/after[6].rules
: rules file containing rules evaluated after UI added rules

/etc/default/ufw
: high level configuration

/etc/ufw/sysctl.conf
: kernel network tunables

/etc/ufw/ufw.conf
: additional high level configuration

BOOT INITIALIZATION

ufw is started on boot with /lib/ufw/ufw−init. This script is a standard SysV style initscript used by the **ufw** command and should not be modified. The /etc/before.init and /etc/after.init scripts may be used to perform any additional firewall configuration that is not yet supported in ufw itself and if they exist and are executable, ufw−init will execute these scripts. ufw−init will exit with error if either of these scripts exit with error. ufw−init supports the following arguments:

start: loads the firewall

stop: unloads the firewall

restart: reloads the firewall

force−reload:
: same as restart

status: basic status of the firewall

force−stop:
: same as stop, except does not check if the firewall is already loaded

flush−all:
flushes the built−in chains, deletes all non−built−in chains and resets the policy to ACCEPT

ufw−init will call before.init and after.init with start, stop, status and flush−all, but typically, if used, these scripts need only implement start and stop.

ufw uses many user−defined chains in addition to the built−in iptables chains. If MANAGE_BUILTINS in /etc/default/ufw is set to 'yes', on stop and reload the built−in chains are flushed. If it is set to 'no', on stop and reload the **ufw** secondary chains are removed and the **ufw** primary chains are flushed. In addition to flushing the **ufw** specific chains, it keeps the primary chains in the same order with respect to any other user−defined chains that may have been added. This allows for **ufw** to interoperate with other software that may manage their own firewall rules.

To ensure your firewall is loading on boot, you must integrate this script into the boot process. Consult your distribution's documentation for the proper way to modify your boot process if **ufw** is not already integrated.

RULES FILES

ufw is in part a front−end for **iptables−restore**, with its rules saved in /etc/ufw/before.rules, /etc/ufw/after.rules and /etc/ufw/user.rules. Administrators can customize **before.rules** and **after.rules** as desired using the standard **iptables−restore** syntax. Rules are evaluated as follows: **before.rules** first, **user.rules** next, and **after.rules** last. IPv6 rules are evaluated in the same way, with the rules files named **before6.rules**, **user6.rules** and **after6.rules**. Please note that **ufw status** only shows rules added with **ufw** and not the rules found in the /etc/ufw rules files.

Important: **ufw** only uses the *filter table by default. You may add any other tables such as *nat, *raw and *mangle as desired. For each table a corresponding COMMIT statement is required.

After modifying any of these files, you must reload **ufw** for the rules to take effect. See the EXAMPLES section for common uses of these rules files.

MODULES

Netfilter has many different connection tracking modules. These modules are aware of the underlying protocol and allow the administrator to simplify his or her rule sets. You can adjust which netfilter modules to load by adjusting IPT_MODULES in /etc/default/ufw. Some popular modules to load are:

nf_conntrack_ftp
nf_nat_ftp
nf_conntrack_irc
nf_nat_irc
nf_conntrack_netbios_ns
nf_conntrack_pptp
nf_conntrack_tftp
nf_nat_tftp
nf_conntrack_sane

KERNEL PARAMETERS

ufw will read in /etc/ufw/sysctl.conf on boot when enabled. Please note that /etc/ufw/sysctl.conf overrides values in the system systcl.conf (usually /etc/sysctl.conf). Administrators can change the file used by modifying /etc/default/ufw.

IPV6

IPv6 is enabled by default. When disabled, all incoming, outgoing and forwarded packets are dropped, with the exception of traffic on the loopback interface. To adjust this behavior, set IPV6 to 'yes' in /etc/default/ufw. See the **ufw** manual page for details.

EXAMPLES

As mentioned, **ufw** loads its rules files into the kernel by using the **iptables−restore** and **ip6tables−restore** commands. Users wanting to add rules to the **ufw** rules files manually must be familiar with these as well as the **iptables** and **ip6tables** commands. Below are some common examples of using the **ufw** rules files. All examples assume IPv4 only and that DEFAULT_FORWARD_POLICY in /etc/default/ufw is set to DROP.

IP Masquerading

To allow IP masquerading for computers from the 10.0.0.0/8 network on eth1 to share the single IP address on eth0:

Edit /etc/ufw/sysctl.conf to have:

```
net.ipv4.ip_forward=1
```

Add to the end of /etc/ufw/before.rules, after the *filter section:

```
*nat
:POSTROUTING ACCEPT [0:0]
−A POSTROUTING −s 10.0.0.0/8 −o eth0 −j MASQUERADE
COMMIT
```

If your firewall is using IPv6 tunnels or 6to4 and is also doing NAT, then you should not usually masquerade protocol '41' (ipv6) packets. For example, instead of the above, /etc/ufw/before.rules can be adjusted to have:

```
*nat
:POSTROUTING ACCEPT [0:0]
−A POSTROUTING −s 10.0.0.0/8 ! −−protocol 41 −o eth0 −j MASQUERADE
COMMIT
```

Add the **ufw route** to allow the traffic:

```
ufw route allow in on eth1 out on eth0 from 10.0.0.0/8
```

Port Redirections

To forward tcp port 80 on eth0 to go to the webserver at 10.0.0.2:

Edit /etc/ufw/sysctl.conf to have:

```
net.ipv4.ip_forward=1
```

Add to the end of /etc/ufw/before.rules, after the *filter section:

```
*nat
:PREROUTING ACCEPT [0:0]
−A PREROUTING −p tcp −i eth0 −−dport 80 −j DNAT \
  −−to−destination 10.0.0.2:80
COMMIT
```

Add the **ufw route** rule to allow the traffic:

```
ufw route allow in on eth0 to 10.0.0.2 port 80 proto tcp
```

Egress filtering

To block RFC1918 addresses going out of eth0:

Add the **ufw route** rules to reject the traffic:

```
ufw route reject out on eth0 to 10.0.0.0/8
ufw route reject out on eth0 to 172.16.0.0/12
ufw route reject out on eth0 to 192.168.0.0/16
```

Full example

This example combines the other examples and demonstrates a simple routing firewall. **Warning**: this setup is only an example to demonstrate the functionality of the **ufw** framework in a concise and simple manner and should not be used in production without understanding what each part does and does not do. Your firewall will undoubtedly want to be less open.

This router/firewall has two interfaces: eth0 (Internet facing) and eth1 (internal LAN). Internal clients have addresses on the 10.0.0.0/8 network and should be able to connect to anywhere on the Internet. Connections to port 80 from the Internet should be forwarded to 10.0.0.2. Access to ssh port 22 from the administrative workstation (10.0.0.100) to this machine should be allowed. Also make sure no internal traffic goes to the Internet.

Edit /etc/ufw/sysctl.conf to have:

```
net.ipv4.ip_forward=1
```

Add to the end of /etc/ufw/before.rules, after the *filter section:

```
*nat
:PREROUTING ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
-A PREROUTING -p tcp -i eth0 --dport 80 -j DNAT \
  --to-destination 10.0.0.2:80
-A POSTROUTING -s 10.0.0.0/8 -o eth0 -j MASQUERADE
COMMIT
```

Add the necessary **ufw** rules:

```
ufw route reject out on eth0 to 10.0.0.0/8
ufw route reject out on eth0 to 172.16.0.0/12
ufw route reject out on eth0 to 192.168.0.0/16
ufw route allow in on eth1 out on eth0 from 10.0.0.0/8
ufw route allow in on eth0 to 10.0.0.2 port 80 proto tcp
ufw allow in on eth1 from 10.0.0.100 to any port 22 proto tcp
```

NOTES

When using ufw with libvirt and bridging, packets may be blocked. The libvirt team recommends that the following sysctl's be set to disable netfilter on the bridge:

```
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
```

Note that the bridge module must be loaded in to the kernel before these values are set. One way to ensure this works properly with ufw is to add 'bridge' to IPT_MODULES in /etc/default/ufw, and then add the above rules to /etc/ufw/sysctl.conf.

Alternatively to disabling netfilter on the bridge, you can configure iptables to allow all traffic to be forwarded across the bridge. Eg, add to /etc/ufw/before.rules within the *filter section:

```
-I FORWARD -m physdev --physdev-is-bridged -j ACCEPT
```

SEE ALSO

ufw(8), **iptables**(8), **ip6tables**(8), **iptables-restore**(8), **ip6tables-restore**(8), **sysctl**(8), **sysctl.conf**(5)

AUTHOR

ufw and this manual page was originally written by Jamie Strandboge <jamie@canonical.com>