OpenBSD - pf+rdomains create splendid multi-tenancy firewalls

Philipp Bühler
sysfive.com GmbH

Abstract

This paper presents a working OpenBSD environment establishing a multi-tenant firewall with `pf(4)`, `rdomain/pair(4)` and `relayd(8)` as work horses only.

The environment shows how to provision, operate, isolate and manage all the components needed - and what isn't. It'll reveal how even complex setups can be developed, tested and provisioned in a straightforward way.

Besides detailing on the OpenBSD bolts, there will be quick walkthrough how to create testing setups easily using Vagrant in preparation for live usage.

For easy re-enacting all configuration of OpenBSD and Vagrant being used will be available online.

1 Introduction

Modern networks can grow already in small scale to quite complex setups, putting security considerations some sophisticated segregation is a must. While this might be manageable for a single tenant (customer/project/..), it can overwhelm easily if adding multilple tenants. The components seggregated in such a 'simple' network are typically:

- `Management` - IPMI, KVM, PXE, monitoring, backup, ..
- `Services` - Proxies, email, ntp, DNS, ..
- `Application` - devel, test/stage, main/live, DR
- `Data` - RDBMS, NoSQL, LDAP, redis, ..
- `Others` - payment services, weather widget, ..

These adds up in realiter environments to numbers like 40 interfaces, 250-300 pf rules and 30 relayd rules.

Traditionally the approach is to either use massivly detailled firewall configuration to seperate the tenants from each other - or even use multiple physical firewall servers. In situations where multiple tenants using overlapping TCP/IP addressing schemes, this adds another problem layer to deal with.

Routing domains (`rdomains`) can help substantially to keep tenants isolated from each other. It also helps to circumevent routing problems in case of overlapping IP addresses.

If such setups are combined with a streamlined testing and provisioning the installation, rollout and maintenance of multi-tenant firewalls can be way more feasible.

2 Traditional Approach

The common approach to address such configurations have the following 'patterns' - each posing it's own set of problems adding to it.

2.1 handcruft

A very carefully handcrafted pf.conf is possible, but not feasible in long term. Esp. when a shared team should operate it. There will be isolated understanding/knowhow of certain "quirks". It's also difficult to replicate into a testing environment to evaluate bigger changes before they happen. With skipped testing, it's common that changes result in failures and can trigger panic situations - e.g. when one tenant can suddenly "see" another.

2.2 templates

The use of configuration blocks or similar templating approach might help in speed of deployment, but can create just multiple the havoc from above.

2.3 multiple servers

Using multiple servers ensures seggregation more easily but get's to new problems. Which tenant(s) are on

which firewall; are there enough IPv4 addresses left to install yet another new server. Would you have enough rackspace to host the next one and how long does it take to get the machine physically into the rack and get it operable.

3 Routing Domains

The use of rdomains can addresses several problems from above. By using this software each tenant will be able to use its own network/routing configuration that is logically fully seperated from other tenants without additional firewall configuration. Using additional configuration would be needed to allow certain cross communication - e.g. to a centralized backup "tenant".

3.1 Setup

OpenBSD is using the rdomains stack by default. If no further setup happens, everything is `rdomain 0`. This fact is a bit hidden from the user by not showing default use in e.g. `ifconfig(8)`.

To create additional rdomains one has to put interfaces into one by flagging the interface with `ifconfig(8)`. Daemons can then be set to be started within such an rdomain, pf.conf can make use this by seggregation configurations (include/anchors) into rdomains. If need be, the special interface `pair(4)` can be used to route between rdomains directly.

3.2 Tools

Several tools for configuration and/or debugging aswell as daemons in the base system are aware of rdomains in OpenBSD. Following a brief list on how to invoke these for a given rdomain. Furthermore some more detail on how to configure the system and daemons to be properly aware of rdomains.

- `netstat -T <tableid>` - show L3 network information
- `route -T <tableid>` - show/modify routing
- `route -T <tableid> exec somedaemon` - start `somedaemon` in this rdomain
- `arp/ndp -V <tableid>` - show L2 network information
- `ping -V <tableid>` - emit ping packets from this rdomain
- `traceroute -V <tableid>` - trace routes from this rdomain
- `nc -V <tableid>` - bind socket in this rdomain
- `ps -o rtable` - adds ID of rdomain the process runs within
- `pkill /pgrep -T <tableid>` - limit results to this rdomain
- `tcpbench -V <tableid>` - run benchmark in this rdomain
- `telnet -V <tableid>` -
- `ftp-proxy` - via `pf(4)` tagging
- `bgpd/ospfd/ripd/eigrpd/ldpd` - via config options
- `authpf` - via multiple tt pf(4) anchors
- `relayd, rc.d, rcctl, ntpd, ifconfig, hostname.if` - see extra subsections

3.2.1 route(8)

Daemons or other tools that are not directly aware of rdomins can be started via route(8) with the `exec <daemon>` option. Examples:

```
route -T 23 exec iked -ddvvf \
/etc/iked.conf.23
route -T 42 exec iked -ddvvf \
/etc/iked.conf.42
```

Be aware that daemons sharing information in the kernel as `ntpd(8)` can create havoc if invoked multiple times.

3.2.2 pf.conf(5)

The pf(4) configuration supports rdomain on three different keywords: 'on rdomain, 'rtable' and 'anchor'. E.g.

```
pass in on rdomain 21 from $tenant-app \
to $tenant-email
#
pass in from $backup to <tenant1> rtable 21
#
anchor "tenant1.21" on rdomain 21 {
    block
    pass out proto tcp from any to any \
      port { 80 443 }
}

anchor "tenant2.41" on rdomain 41 {
    block
    match out to any nat-to $ext-41-ip \
      rtable 0 tag TENANT_41
    pass out tagged TENANT_41
}
```

3.2.3 hostname.if(5)

Creating persistent rdomains is done by assigning `rdomain N` to an interface config file. Since any `rdomain(4)` configuration removes `inet / inet6` settings it's important to add this after any of those. Examples for physical, vlan(4) and carp(4) should look like this:

```
/etc/hostname.em0:
rdomain 0
inet 10.40.40.254/24

/etc/hostname.vlan41:
description "gw-vlan-41"
vlan 41 vlandev em2
rdomain 41
inet 10.40.41.1/24

/etc/hostname.carp1
description "gw-carp-1"
rdomain 0
vhid 1
pass onetwomany
carpdev em0
inet 10.60.5.1/24
```

To include 'patch' and routing information for the specialized `pair(4)` interfaces the following ordering should be amended: rdomain, inet, patch, route, e.g.:

```
/etc/hostname.pair0:
description "gw-pair-0"
rdomain 0
inet 10.200.21.1/30

/etc/hostname.pair21:
description "gw-pair-21"
rdomain 21
inet 10.200.21.2/30
patch pair0
!/sbin/route -T 21 -qn add default 10.200.21.1
```

3.2.4 rc.d(8)

For automated startup, `rc.d(8)` has 'daemon-name_rtable=N' support, which defaults to 0. Consequently this can be configured using `rcctl(8)` yada.

```
$ doas rcctl set httpd status on
$ doas rcctl set httpd rtable 21

$ doas rcctl get httpd
httpd_class=daemon
httpd_flags=
httpd_rtable=21
httpd_timeout=30
httpd_user=root

$ doas rcctl start httpd
httpd(ok)

$ ps auxo rtable | grep http # last column
www 46042  0.0  0.7  744  1740 ??  Sp \
  4:43PM  0:00.00 httpd: server (h   21
```

From above example the httpd(8) was started like a manually invoked `route -T 21 exec httpd`.

3.2.5 ntpd(8)

This is a 'famous' example for daemons that shall not be started multiple times to cover several routing domains! Each invoked daemon will try to overwrite the kernel's clock resulting in skews that can cover 'months' within some wallclock seconds. To overcome this, ntpd has been taught to operate in multiple rdomains from one invocation. The `listen` socket goes to the rdomain it was invoked in (typically '0') and the `server` can be bound to other domains, e.g. for `/etc/ntpd.conf`:

```
server jp.pool.ntp.org
listen 127.0.0.1
listen 127.0.0.1 rtable 69
listen 10.20.41.1 rtable 41
listen 10.20.21.1 rtable 21
```

3.2.6 pair(4)

With pair(4) and route(8) one can interconnect rdomains. Being virtualized ethernet it needs two endpoints that are then patched to each other:

```
$ doas ifconfig pair0 rdomain 0 10.200.21.1/30 up
$ doas ifconfig pair21 rdomain 21 10.200.21.2/30 up
$ doas ifconfig pair0 patch pair21
```

The above ad-hoc setup can be persisted by using hostname.if as below:

```
/etc/hostname.pair0:
description "gw-pair-0"
rdomain 0
inet 10.200.21.1/30
/etc/hostname.pair21:
description "gw-pair-21"
rdomain 21
inet 10.200.21.2/30
patch pair0
!/sbin/route -T 21 -qn add default 10.200.21.1
```

The pair(4) devices can be added to a bridge(4), too, but STP configuration must be added to avoid looping packets.

3.2.7 pf.conf(5)

The use of rdomains simplifies the burden to be careful along means of first/last match wins or using 'quick'. By using 'anchor X on rdomain N' the rules needed for a tenant are completly isolated and the different match rules wont influence each other. In the following example the first three 'match' lines will influence each other and the result will not be the desired one. In this case the first 'match' will change the 'to' address and thus the second 'match' will not be in effect'.

Using anchors on rdomains, this wont happen. Packets on rdomain-41 will never be processed by the rules of anchor on rdomain-21 and vice versa.

```
match out on $ext inet proto { udp tcp } \
 from <net_tenant1> to !<rfc1918> \
  nat-to $nat_tenant1
match out on $ext inet proto tcp from any \
 to !<rfc1918> port 25 \
  nat-to $nat_tenant1_mail
match out on $ext inet proto { udp tcp } \
 from <net_tenant2> to !<rfc1918> \
  nat-to $nat_tenant2
#rdomains:
anchor "tenant1" on rdomain 21 {
 match out on $ext inet proto { udp tcp } \
  from <net_tenant1> to !<rfc1918> \
   nat-to $nat_tenant1
 match out on $ext inet proto tcp from any \
  to !<rfc1918> port 25 nat-to \
   $nat_tenant1_mail
}
anchor "tenant2" on rdomain 41 {
 match out on $ext inet proto { udp tcp } \
  from <net_tenant2> to !<rfc1918> \
   nat-to $nat_tenant2
}
```

Using includes the manageability of pf.conf can improve dramatically in terms of oversight and delegation.

```
#all-in-one /etc/pf.conf:
 set skip on lo0 enc0 enc1
 set optimization aggressive
 # important below!
 block in from $tenant1 to $tenant2
 pass from $tenant1 to any \
    nat-to $tenant1_public
 match out from $tenant2 to any nat-to \
   $tenant2_public #on request call 3am
 match out from any to any nat-to (egress)
###
#rdomains with includes /etc/pf.conf:
 include "/etc/pf/globals.conf"
 include "/etc/pf/management.conf"
 anchor "tenant1" on rdomain 21 {
   include "/etc/pf/tenant1.conf"
 }
 anchor "tenant1" on rdomain 41 {
 include "/etc/pf/tenant2.conf"
 }
# EOF
```

3.2.8 relayd(8)

As of now `relayd(8)` isnt aware of routing domains, but by forementioned 'route exec' some achievements can be made. Besides a full blown support, a little patch for relayd is available[1] which enables it to use multiple 'anchors' in pf.

3.3 other quirks

For certain cases, the defaults or limitations can be overcome by compiling or creative network interface stacking

3.3.1 limit in number

A maximum of 256 routing domains are possible with a GENERIC kernel as in the default install. If need be, this can be changed in `sys/socket.h` for the define of `RT_TABLEID_MAX`. This will need full 'release' build – or you know what you do.

3.3.2 carp(4)

Interfaces of this driver must be in the same rdomain as its `carpdev`. Besides using `vlan(4)` as the underlying device, it's also possible to use `vether(4)` to "break" the linkage between carp and it's physical counterpart. To put use on the later, the physical interface and two vether interfaces are joined in the same `bridge(4)` and the `carp(4)` devices are stacked on top of it: https://gist.github.com/doublep/d3a20fded7e8ced30735705e1dfea5c4

4 External Tooling

Testing is essential and is of more fun, when one can do it "whereever" - like on a trip on the laptop. Using a stack of 'packer', 'Vagrant' and ansible it's feasible to run extensive testing. Be it to get more familiar with the concept or trying to debug a live problem – on your way home.

4.1 Testbed Layout

The following configuration result in a virtualized networking environment that consists of two tenant ”clients” connected to one firewall, this one to a second and with the second firewall is a mock internet ”server”.

4.2 Provisioning

The following setup uses `packer`, `Vagrant` with `Virtualbox` and `ansible` to create and run the testbed.

4.2.1 packer

This tool takes a stock OpenBSD install image and converts it into a `vbox` VM-image, which is the base for all five VMs the testbed needs. It leverages the `autoinstall(8)` feature of OpenBSD to achieve this. Furthermore it will install `sudo(8)` and `python(1)` to enable Vagrant and ansible.

4.2.2 Vagrant

Based on above vbox ’base box’ the referenced configuration sets the parameters needed to launch the testbed. Most notable are the network settings. The duo of Vagrant and Virtualbox will put the VMs with adjacent (same subnet) networks on the same virtualized cable. Given the corresponding routing, the VMs networks wont see each other unless explicitly told so by routing, rdomains and pf.

4.3 Automation

To lessen the burden of configuring the networking `ansible` can be used to automatically generate `hostname.if(5)` files and call `netstart(8)` with those. On top of the packer/vagrant based VMs, a reproducable testbed networking is ensured and available within minutes.

4.3.1 ansible

The referenced `ansible` code will read global configuration data from `group_vars/testbed` and host/VM specific data from `host_vars/hostname`. It takes this data to fill in jinja2-templates in `roles/firewall/templates/` to generate and upload the needed `hostname.if(5)` files on the testbed firewall/rdomain host.

5 Acknowledgments

Following people have helped me directly or indirectly by writing the used software, documentation, other talks etc.

- ”Peter Hessler” - for the talks, experiences and help in rdomains
- ”Ingo Schwarze” - for helping out with roff/gpresent to create the presentation slides
- ”OpenBSD developers” - for adding pf, rdomains and OpenBSD itself
- ”sysfive.com GmbH” - for giving enough working hours to get this done

6 Availability

This paper, presentation slides, Vagrant and packer templates and also ansible code can be found on github:

`https://github.com/double-p/smtf/`