Understanding Firewalld in Multi-Zone Configurations

Nathan R. Vance and William F. Polik, Hope College, Holland, MI 49423

Introduction

Stories of compromised servers and data theft fill today's news. It isn't difficult for someone who has read an informative blog post to access a system via a misconfigured service, take advantage of a recently exposed vulnerability, or gain control using a stolen password. Any of the many internet services found on a typical Linux server could harbor a vulnerability that grants unauthorized access to the system.

Since it's an impossible task to harden a system at the application level against every possible threat, firewalls provide security by limiting access to a system. Firewalls filter incoming packets based on their IP of origin, their destination port, and their protocol. This way, only a few IP/port/protocol combinations interact with the system, and the rest do not.

Linux firewalls are handled by netfilter, which is a kernel level framework. For over a decade, iptables has provided the userland abstraction layer for netfilter. Iptables subjects packets to a gauntlet of rules where, if the IP/port/protocol combination of the rule matches the packet, the rule is applied causing the packet to be accepted, rejected, or dropped.

Firewalld is a newer userland abstraction layer for netfilter. Unfortunately, its power and flexibility are underappreciated due to a lack of documentation describing multi-zoned configurations. This article provides examples to remedy this situation.

Firewalld Design Goals

The designers of firewalld realized that most iptables usage cases involve only a few unique IP sources, for each of which a whitelist of services is allowed and the rest are denied. To take advantage of this pattern, firewalld categorizes incoming traffic into zones defined by the source IP and/or network interface. Each zone has its own configuration to accept or deny packets based on specified criteria.

Another improvement over iptables is a simplified syntax. Firewalld makes it easier to specify services by using the name of the service rather than its port(s) and protocol(s), for example, samba rather than UDP ports 137 and 138 and TCP ports 139 and 445. It further simplifies syntax by removing the dependence on the order of statements as was the case for iptables.

Finally, firewalld enables the interactive modification of netfilter, allowing a change in the firewall to occur independently of the permanent configuration stored in XML. Thus,

```
# firewall-cmd <some modification>
```

is a temporary modification which will be overwritten by the next reload, while

```
# firewall-cmd --permanent <some modification>
# firewall-cmd --reload
```

is a permanent change that persists across reboots.

Zones

The top layer of organization in firewalld is zones. A packet is part of a zone if it matches that zone's associated network interface or ip/mask source. Several predefined zones are available:

```
# firewall-cmd --get-zones
block dmz drop external home internal public trusted work
```

An active zone is any zone that is configured with an interface and/or a source. To list active zones:

```
# firewall-cmd --get-active-zones
public
  interfaces: eno1 eno2
```

Interfaces are the system's names for hardware and virtual network adapters, as can be seen in the above example. All active interfaces will be assigned to zones, either to the default zone or to a user specified one. However, an interface cannot be assigned to more than one zone.

In its default configuration, firewalld pairs all interfaces with the public zone and doesn't set up sources for any zones. As a result, public is the only active zone.

Sources are incoming IP address ranges, which can also be assigned to zones. A source (or overlapping sources) cannot be assigned to multiple zones. Doing so results in undefined behavior, as it would not be clear which rules should be applied to that source.

Since specifying a source is not required, for every packet there will be a zone with a matching interface but there won't necessarily be a zone with a matching source. This indicates some form of precedence with priority going to the more specific source zones, but more on that later. First, let's inspect how the public zone is configured:

```
# firewall-cmd --zone=public --list-all
public (default, active)
  interfaces: eno1 eno2
  sources:
  services: dhcpv6-client ssh
  ports:
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
# firewall-cmd --permanent --zone=public --get-target
default
```

Going line by line through the output,

- `public (default, active)` indicates that the public zone is the default zone (interfaces default to it when they come up), and it is active because it has at least one interface or source associated with it.
- `interfaces: eno1 eno2` lists the interfaces associated with the zone.
- `sources:` lists the sources for the zone. There aren't any now, but if there were they would be of the form `xxx.xxx.xxx.xxx/xx`.
- `services: dhcpv6-client ssh` lists the services allowed through the firewall. An exhaustive list of firewalld's defined services is available by executing `firewall-cmd --get-services`.
- `ports:` lists port destinations allowed through the firewall. This is useful if you need to allow a service that isn't defined in firewalld.
- `masquerade: no` indicates that ip masquerading is disabled for this zone. If enabled, this would allow ip forwarding, with your computer acting as a router.
- `forward-ports:` lists ports that are forwarded.

- `icmp-blocks:` a blacklist of blocked icmp traffic.
- `rich rules:` advanced configurations, processed first in a zone.
- `default` is the target of the zone, which determines the action taken on a packet that matches the zone yet isn't explicitly handled by one of the above settings.

A Simple Single-Zoned Example

Say you just want to lock down your firewall. Simply remove the services currently allowed by the public zone and reload:

```
# firewall-cmd --permanent --zone=public --remove-
service=dhcpv6-client
# firewall-cmd --permanent --zone=public --remove-service=ssh
# firewall-cmd --reload
```

These commands result in the following firewall:

```
# firewall-cmd --zone=public --list-all
public (default, active)
  interfaces: eno1 eno2
  sources:
  services:
  ports:
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
# firewall-cmd --permanent --zone=public --get-target
default
```

In the spirit of keeping security as tight as possible, if a situation arises where you need to open a temporary hole in your firewall (perhaps for ssh), you can add the service to just the current session (omit --permanent) and instruct firewalld to revert the modification after a specified amount of time:

```
# firewall-cmd --zone=public --add-service=ssh --timeout=5m
```

The timeout option takes time values in seconds (s), minutes (m), or hours (h).

Targets

When a zone processes a packet due to its source or interface, but there is no rule that explicitly handles the packet, the target of the zone determines the behavior:

- `ACCEPT`: Accept the packet.
- `%%REJECT%%`: Reject the packet, returning a reject reply.
- `DROP`: Drop the packet, returning no reply.
- `default`: Don't do anything. The zone washes its hands of the problem, and kicks it "upstairs."

There was a bug present in firewalld 0.3.9 (fixed in 0.3.10) for source zones with targets other than `default` in which the target was applied regardless of allowed services. For example, a source zone with the target `DROP` would drop all packets, even if they were whitelisted. Unfortunately, this version of firewalld was packaged for RHEL7 and its derivatives, causing it to be a fairly common bug. The examples in this article avoid situations that would manifest this behavior.

Precedence

Active zones fulfill two different roles. Zones with associated interface(s) act as interface zones, and zones with associated source(s) act as source zones (a zone could fulfill both roles). Firewalld handles a packet in the following order:

1. The corresponding source zone. Zero or one such zones may exist. If the source zone deals with the packet because the packet satisfies a rich rule, the service is whitelisted, or the target is not default, we end here. Otherwise, we pass the packet on.
2. The corresponding interface zone. Exactly one such zone will always exist. If the interface zone deals with the packet, we end here. Otherwise, we pass the packet on.
3. The firewalld default action. Accept icmp packets and reject everything else.

The take away message is that source zones have precedence over interface zones. Therefore, the general design pattern for multi-zoned firewalld configurations is to create a privileged source zone to allow specific IP's elevated access to system services, and a restrictive interface zone to limit the access of everyone else.

A Simple Multi-Zoned Example

To demonstrate precedence, let's swap ssh for http in the public zone and set up the default internal zone for our favorite IP address, 1.1.1.1. The following commands accomplish this task:

```
# firewall-cmd --permanent --zone=public --remove-service=ssh
# firewall-cmd --permanent --zone=public --add-service=http
# firewall-cmd --permanent --zone=internal --add-source=1.1.1.1
# firewall-cmd --reload
```

which results in the following configuration:

```
# firewall-cmd --zone=public --list-all
public (default, active)
  interfaces: eno1 eno2
  sources:
  services: dhcpv6-client http
  ports:
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
# firewall-cmd --permanent --zone=public --get-target
default
# firewall-cmd --zone=internal --list-all
internal (active)
  interfaces:
  sources: 1.1.1.1
  services: dhcpv6-client mdns samba-client ssh
  ports:
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
# firewall-cmd --permanent --zone=internal --get-target
default
```

With the above configuration, if someone attempts to ssh in from 1.1.1.1 the request would succeed because the source zone (internal) is applied first, and it allows ssh access.

If someone attempts to ssh from somewhere else, say 2.2.2.2, there wouldn't be a source zone because no zones match that source. Therefore, the request would pass directly to the interface zone (public), which does not explicitly handle ssh. Since public's target is `default`, the request passes to the firewalld default action, which is to reject it.

What if 1.1.1.1 attempts http access? The source zone (internal) doesn't allow it, but the target is `default`, so the request passes to the interface zone (public), which grants access.

Now let's suppose someone from 3.3.3.3 is trolling your website. To restrict access for that IP, simply add it to the preconfigured drop zone, aptly named because it drops all connections:

```
     # firewall-cmd --permanent --zone=drop --add-source=3.3.3.3
     # firewall-cmd --reload
```

The next time 3.3.3.3 attempts to access your website, firewalld will send their request first to the source zone (drop). Since the target is `DROP`, the request will be denied and won't make it to the interface zone (public) to be accepted.

A Practical Multi-Zoned Example

Suppose you are setting up a firewall for a server at your organization. You want the entire world to have http and https access, your organization (1.1.0.0/16) and workgroup (1.1.1.0/8) to have ssh access, and your workgroup to have samba access. Using zones in firewalld, we can set up this configuration in an intuitive manner.

Given the naming, it seems logical to commandeer the public zone for our world-wide purposes, and the internal zone for local use. We start by replacing the dhcpv6-client and ssh services in the public zone with http and https:

```
     # firewall-cmd --permanent --zone=public --remove-
     service=dhcpv6-client
     # firewall-cmd --permanent --zone=public --remove-service=ssh
     # firewall-cmd --permanent --zone=public --add-service=http
     # firewall-cmd --permanent --zone=public --add-service=https
```

Then we trim mdns, samba-client, and dhcpv6-client out of the internal zone (leaving only ssh) and add our organization as the source:

```
     # firewall-cmd --permanent --zone=internal --remove-service=mdns
     # firewall-cmd --permanent --zone=internal --remove-
     service=samba-client
     # firewall-cmd --permanent --zone=internal --remove-
     service=dhcpv6-client
     # firewall-cmd --permanent --zone=internal --add-
     source=1.1.0.0/16
```

To accommodate our elevated workgroup samba privileges, we add a rich rule:

```
     # firewall-cmd --permanent --zone=internal --add-rich-rule='rule
     family=ipv4 source address="1.1.1.0/8" service name="samba"
     accept'
```

Finally we reload, pulling our changes into the active session:

```
     # firewall-cmd --reload
```

Only a few more details remain. Attempting to ssh into our server from an IP outside of the internal zone results in a reject message, which is the firewalld default. It is more secure to exhibit the behavior of an inactive IP and instead drop the connection. Change the public zone's target to `DROP` rather than `default` to accomplish this.

```
# firewall-cmd --permanent --zone=public --set-target=DROP
# firewall-cmd --reload
```

But wait, we can no longer ping, even from the internal zone! And icmp (the protocol ping goes over) isn't on the list of services that firewalld can whitelist. That's because icmp is an IP layer 3 protocol and has no concept of a port, unlike services which are tied to ports. Before setting the public zone to `DROP`, pinging could pass through the firewall because both of our `default` targets passed it on to the firewalld default, which allowed it. Now it's dropped.

To restore pinging to the internal network, we use a rich rule:

```
# firewall-cmd --permanent --zone=internal --add-rich-rule='rule
protocol value="icmp" accept'
# firewall-cmd --reload
```

In summary, here's how we've configured our two active zones:

```
# firewall-cmd --zone=public --list-all
public (default, active)
  interfaces: eno1 eno2
  sources:
  services: http https
  ports:
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
# firewall-cmd --permanent --zone=public --get-target
DROP
# firewall-cmd --zone=internal --list-all
internal (active)
  interfaces:
  sources: 1.1.0.0/16
  services: ssh
  ports:
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
        rule family=ipv4 source address="1.1.1.0/8" service
name="samba" accept
        rule protocol value="icmp" accept
# firewall-cmd --permanent --zone=internal --get-target
default
```

This setup demonstrates a three-layer nested firewall. The outermost layer, public, is an interface zone and spans the entire world. The next layer, internal, is a source zone and spans your organization, which is a subset of public. Finally, a rich rule adds the innermost layer spanning your workgroup, which is a subset of internal.

The take away message is that, when a scenario can be broken into nested layers, the broadest layer should use an interface zone, the next layer should use a source zone, and additional layers should use rich rules within the source zone.

Debugging

Firewalld employs intuitive paradigms for designing a firewall, yet gives rise to ambiguity much more easily than its predecessor, iptables. Should unexpected behavior occur, or to better understand how firewalld works, it can be useful to obtain an iptables description of how netfilter has been configured to operate. Output for the previous example follows, with forward, output, and logging lines trimmed for simplicity:

```
# iptables -S
-P INPUT ACCEPT
... (forward and output lines) ...
-N INPUT_ZONES
-N INPUT_ZONES_SOURCE
-N INPUT_direct
-N IN_internal
-N IN_internal_allow
-N IN_internal_deny
-N IN_public
-N IN_public_allow
-N IN_public_deny
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -j INPUT_ZONES_SOURCE
-A INPUT -j INPUT_ZONES
-A INPUT -p icmp -j ACCEPT
-A INPUT -m conntrack --ctstate INVALID -j DROP
-A INPUT -j REJECT --reject-with icmp-host-prohibited
... (forward and output lines) ...
-A INPUT_ZONES -i eno1 -j IN_public
-A INPUT_ZONES -i eno2 -j IN_public
-A INPUT_ZONES -j IN_public
-A INPUT_ZONES_SOURCE -s 1.1.0.0/16 -g IN_internal
-A IN_internal -j IN_internal_deny
-A IN_internal -j IN_internal_allow
-A IN_internal_allow -p tcp -m tcp --dport 22 -m conntrack --
ctstate NEW -j ACCEPT
-A IN_internal_allow -s 1.1.1.0/8 -p udp -m udp --dport 137 -m
conntrack --ctstate NEW -j ACCEPT
-A IN_internal_allow -s 1.1.1.0/8 -p udp -m udp --dport 138 -m
conntrack --ctstate NEW -j ACCEPT
-A IN_internal_allow -s 1.1.1.0/8 -p tcp -m tcp --dport 139 -m
conntrack --ctstate NEW -j ACCEPT
-A IN_internal_allow -s 1.1.1.0/8 -p tcp -m tcp --dport 445 -m
conntrack --ctstate NEW -j ACCEPT
```

```
-A IN_internal_allow -p icmp -m conntrack --ctstate NEW -j
ACCEPT
-A IN_public -j IN_public_deny
-A IN_public -j IN_public_allow
-A IN_public -j DROP
-A IN_public_allow -p tcp -m tcp --dport 80 -m conntrack --
ctstate NEW -j ACCEPT
-A IN_public_allow -p tcp -m tcp --dport 443 -m conntrack --
ctstate NEW -j ACCEPT
```

In the above iptables output, new chains (lines starting with `-N`) are first declared. The rest are rules appended (starting with `-A`) to iptables. Established connections and local traffic are accepted and incoming packets go to the `INPUT_ZONES_SOURCE` chain, at which point IP's are sent to the corresponding zone, if one exists. After that, traffic goes to the `INPUT_ZONES` chain, at which point it is routed to an interface zone. If it isn't handled there, icmp is accepted, invalids are dropped, and everything else is rejected.

Conclusion

Firewalld is an under-documented firewall configuration tool with more potential than many people realize. With its innovative paradigm of zones, firewalld allows the system administrator to break traffic up into categories where each receives a unique treatment, simplifying the configuration process. Because of its intuitive design and syntax, it is practical for both simple single-zoned and complex multi-zoned configurations.

Biographical Sketch

Nathan Vance is a computer science major at Hope College in Holland, MI. He installed Linux Mint 12 as a high school Junior and now prefers Arch Linux. He drives a home-built electric powered '95 Ford Probe with a Raspberry Pi car computer.

William Polik is a computational chemistry professor at Hope College in Holland, MI. He cut his programming teeth with Turbo Pascal 3 in 1986 and joined the Linux revolution with Red Hat 5 in 1997. He founded two web-based software companies, DiscusWare LLC and WebMO LLC.