OpenVPN (client) in Linux containers

This article describes how to setup a **Linux Container** to run OpenVPN in client mode with a "kill switch" for secure/private internet use. Doing so offers a distinct advantage over using full-blown virtualization like **VirtualBox** or **QEMU** in that the resource overhead is minimal by comparison and able to run on low powered devices.

Contents

Related articles

AirVPN

Docker

Linux Containers

OpenVPN

OpenVPN (server) in Linux containers

PeerGuardian_Linux

Systemd-nspawn

ufw

Container setup

Basic setup and understanding of **Linux Containers** is required. This article assumes that readers have a base LXC setup operational. Newcomers to these are directed to the aforementioned article.

LXC config

The container's config should be modified to include several key lines in order to both run OpenVPN and have internet programs (browsers, email clients, torrent clients, etc.) interact with the host system from within the LXC and from behind the VPN.

For the example, the lxc is named "playtime" and a full config is shown:

```
/var/lib/lxc/playtime/config
```

```
...

## for openvpn
lxc.mount.entry = /dev/net dev/net none bind,create=dir
lxc.cgroup.devices.allow = c 10:200 rwm
```

Needed packages within the container

In addition to the base system, **openvpn (https://www.archlinux.org/packages/?name=openvpn)** is required and available from the **official repositories**. A properly configured **firewall** to run within the container is *highly* recommended. The role of the firewall within the container is two fold:

1. Provide a functional "kill switch" to maintain privacy should the connection to the VPN fail.
2. Keep nasty stuff out.

This guide uses **ufw (https://www.archlinux.org/packages/?name=ufw)** which is very easy to configure, but other examples can certainly be used.

Package setup

OpenVPN

Configuration of OpenVPN is beyond the scope of this article. Readers are encouraged to read the **OpenVPN** article to properly setup the software for a given VPN provider. Note that many private VPN providers include links to directly download a properly configured openvpn.opvn profile unique to their particular service. For the purposes of this guide, `/etc/openvpn/client/myprofile.conf` will refer to that config.

Verify openvpn functionality within the container; **start** openvpn via `openvpn-client@myprofile.service` and once satisfied **enable** it to run at boot.

Note: Users running openvpn within an *unprivileged* container will need to create a custom systemd unit to start it within the container. Simply copy the package-provided `/usr/lib/systemd/system/openvpn-client@.service` to `/etc/systemd/system/openvpn-client@.service` and modify the new file commenting out the the line beginning with: `LimitNPROC...`

Avoiding DNS leaks

Users are highly encouraged to setup openvpn with to manage `/etc/resolv.conf` via the instructions in **OpenVPN#DNS**. Failure to do so may lead to DNS leaks if the host's DNS server is specified in said file.

ufw

Note: The following applies to inside the container.

Configuration of **ufw** is described in **OpenVPN#Firewall configuration**. Once modified as described, additional setup is simply to define the protocols to run (ssh, torrent, etc.) and also define the IP address or address range of the VPN provider. Again, these are available from the private providers.

Note: The following needs to be executed as the root user; the "#" sign prefixing these commands per standard wiki notation has been omitted to allow for a clean copy/paste into a

terminal.

First setup the default deny policy and then allow whatever services are to pass though:

```
ufw default deny outgoing
ufw default deny incoming
ufw allow ssh
ufw allow 1194
ufw allow out 1194
ufw allow out on tun0 from any to any
ufw allow in on tun0 from any to any
```

Now add the VPN server IP addresses or ranges. Note two entries for each IP address is required. In the example below, only a single fictitious IP address is shown for illustrative purposes.

```
ufw allow in from 50.120.10.200 to any
ufw allow out from 50.121.10.200 to any
```

Finally, add the internal LAN IP range to allow access regardless of VPN connectivity:

```
ufw allow from 192.168.0.0/24
```

Start ufw and **enable** `ufw.service` to start at boot.

```
# ufw enable
```

pgl

Additional protection can be had via using **pgl** within the container. See: **PeerGuardian Linux#Running pgl from within a container**.

Test the service

From within the running container, (connected via ssh or via `lxc-attach -n playtime`) test the setup by exporting a browser to the host's machine X server:

```
$ DISPLAY=:0 firefox
```

The result should be a firefox window in the host's X server with the title, "Mozilla Firefox (playtime)." A number of websites can be used to verify IP address and status of DNS entries. Once such site is **ipleak dot net (http://ipleak.net)**.

At this point, the IP and DNS entries corresponding to `/etc/openvpn/client/myprofile.conf` should be displayed.