Creating Proxied Subnets with LXC

tropf

ABSTRACT

short howto on creating a subnet for lxc containers which is proxied through a VPN

1. Concept

1.1. Goals

All communication of a set of containers is routed and sent through a VPN. In the case of an outage of that VPN, the containers in the subnet will not be able to communicate to the outside world.

1.2. Architecture

The "network" is a bridge on the host system. All isolated containers will have a connection to only that bridge, while the proxy container will be attached to the normal lxc bridge (the uplink) and the isolated bridge. All traffic on the isolated bridge will be NATed and tunneled through a VPN by the proxy container.

While this setup is fairly simple, it places some restrictions:

- no DHCP: to prevent accidentally configuring a gateway, all isolated containers will have static IP addresses
- iptables galore: to prevent accidental communication to the outside, heavily restrictive iptables rules are employed

1.3. Example Values

The following values are used throughout the document.

Description	**Value**
default LXC bridge name	`lxcbr0`
LXC bridge subnet	`10.0.3.0/24`
isolated bridge name	`brvpn`
isolated bridge network	`10.0.5.0/24`
proxy container name	`proxycontainer`
proxy container address	`10.0.5.2`
isolated container name	`isolatedcontainer`
isolated container address	`10.0.5.101`
API to check if VPN is connected	`https://api.vpn.example/connectcheck`
group used to launch openvpn with	`vpngrp`

2. Setting up the Host System

2.1. Configuring the Bridge

Add a new bridge with no attached devices, and give it a separate subnet. There are plenty of ways to configure a bridge; I the following lines to `/etc/network/interfaces`.

```
auto brvpn
iface brvpn inet static
        bridge_ports none
        address 10.0.5.1
        netmask 255.255.255.0
        bridge_fd 5
        bridge_stp no
```

› Make sure you get this right, or your network might refuse to load up at all.

Afterwards bring up the bridge with `sudo ifup brvpn` and check its existance with `ip addr show dev brvpn`.

3. Setting up the Proxy Container

3.1. Network Configuration

Edit the container configuration in `/var/lib/lxc/proxycontainer/config`, copy the net section a second time and adjust it to connect to the bridge:

```
lxc.net.0.type = veth
lxc.net.0.link = lxcbr0
lxc.net.0.flags = up
lxc.net.0.hwaddr = 00:16:3e:a0:97:e8

lxc.net.1.type = veth
lxc.net.1.link = brvpn
lxc.net.1.flags = up
lxc.net.1.hwaddr = 02:16:3e:a0:97:e8
lxc.net.1.ipv4.address = 10.0.5.2/24
```

› Both the MAC and IP address of the second interface are changed.

Bring up the container and check the internet connection. Check `/etc/network/interfaces` and disable DHCP on the interface to `brvpn` if required.

3.2. VPN Connection

This example describes how to setup openvpn. Other VPNs might work very differently.

3.2.1. Configuration

Openvpn in containers doesn't work out of the box, as a tun device has to be created. This is an adaptation from here ⟨`https://web.archive.org/web/20190428024612/heider.io/blog/2013/10/26/openvpn-in-a-lxc-container/`⟩.

On the host system add the file `/var/lib/lxc/proxycontainer/autodev` with the following content:

```
#!/bin/bash

cd ${LXC_ROOTFS_MOUNT}/dev
mkdir net
mknod net/tun c 10 200
chmod 0666 net/tun
```

Make the file executable: `chmod +x /var/lib/lxc/proxycontainer/autodev`

Add this to the container config `/var/lib/lxc/proxycontainer/config`:

```
lxc.hook.autodev=/var/lib/lxc/proxycontainer/autodev
```

Inside the container install the openvpn client and place the configuration files somewhere, for example `/etc/openvpn/client/`.

› Maybe adjust the access rights.

3.2.2. Testing

Try to open a vpn connection with`openvpn --config /etc/openvpn/client/vpn.conf` (or whereever you placed your config).

Use an API provided by your VPN provider with curl to check if the VPN is online. Build a pipeline like `curl https://api.vpn.example/connectcheck | grep 'You are connected.'` that will succeed if you are connected and fail if not. **We will need that pipeline in a second.**

3.2.3. Adding a Custom Service

Add a group that executes the vpn to later flag the traffic easily with iptables:

```
groupadd -r vpngrp
```

Add two scripts for when the VPN goes up/down and make them executable:

```
touch /etc/openvpn/up /etc/openvpn/down
chmod +x /etc/openvpn/up /etc/openvpn/down
```

Then create a systemd service unit at `/etc/systemd/system/vpn.service` to connect to the vpn and call these scripts.

```
[Unit]
Description=Proxy all traffic via vpn
After=network.target

[Service]
Type=simple
ExecStart=/usr/sbin/openvpn --config /etc/openvpn/client/vpn.conf
ExecStartPost=/etc/openvpn/up
ExecStopPost=/etc/openvpn/down
WorkingDirectory=/etc/openvpn/client
Group=vpngrp
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Enable the service

```
systemctl daemon-reload
systemctl enable vpn.service
```

3.2.4. Enable and Restrict Traffic Proxying

Start by disabling all traffic forwarding by default. Forwarding will only be enabled by our custom scripts after the firewall has been adjusted. Add to `/ect/sysctl.conf`:

```
net.ipv4.ip_forward=0
net.ipv6.conf.all.forwarding=0
```

Add the content of `/etc/openvpn/up`. Keep in mind that that file is executed immediately after openvpn is launched, at which point it is not yet connected. So our script will first wait for openvpn get up. Afterwards firewall rules are added:

- Traffic from internal networks will be NATed. (Openvpn will have routes set up that will catch that traffic.)
- Traffic to internal networks is allowd.
- Traffic to the tun device is allowed.
- Traffic by openvpn is allowed.
- All other traffic is forbidden.

Only after all iptables rules are in place will forwarding be enabled:

```
#!/bin/bash

VPN_MARK=13
FORWARD_MARK=6

tries=0
try_result=1

while [ $try_result -ne 0 ]
do
    # no max number of tries, as openvpn will retry forever

    if [ $tries -ge 1 ]
    then
        sleep 5
    fi

    echo "checking connection..."
    curl https://api.vpn.example/connectcheck | grep 'You are connected.' > /dev/nu
    try_result=$?
    tries=$(( $tries + 1))
done

echo "connected."

iptables -t filter -A FORWARD -s 10.0.5.0/24 -j MARK --set-mark $FORWARD_MARK
iptables -t filter -A FORWARD -m mark --mark $FORWARD_MARK -j ACCEPT
iptables -t filter -A FORWARD -d 10.0.5.0/24 -j ACCEPT
iptables -t filter -A FORWARD -j DROP

iptables -t filter -A OUTPUT -m owner --gid-owner vpngrp -j MARK --set-mark $VPN_MA

iptables -t mangle -A POSTROUTING -m mark --mark $VPN_MARK -j ACCEPT
iptables -t mangle -A POSTROUTING -o tun0 -j ACCEPT
iptables -t mangle -A POSTROUTING -d 10.0.5.0/24 -j ACCEPT
iptables -t mangle -A POSTROUTING -j DROP

iptables -t nat -A POSTROUTING -m mark --mark $FORWARD_MARK -j MASQUERADE

sysctl net.ipv4.ip_forward=1
```

› Don’t forget to insert your connection-test-pipeline from above.

The teardown script in `/etc/openvpn/down` is much simpler, it just disables forwarding and clears up the created rules.

```
#!/bin/bash

sysctl net.ipv4.ip_forward=0

iptables -t filter -F
iptables -t mangle -F
iptables -t nat -F
```

Restart the container and check the VPN connection to finish.

4. Setting up the isolated container(s)

Create a container and give it the isolated bridge as only interface. Change network config in `/var/lib/lxc/isolatedcontainer/config` to:

```
lxc.net.0.type = veth
lxc.net.0.link = brvpn
lxc.net.0.flags = up
lxc.net.0.hwaddr = 02:16:3e:61:83:d4
lxc.net.0.ipv4.address = 10.0.5.101/24
lxc.net.0.ipv4.gateway = 10.0.5.2
```

Inside of the container disable DHCP by editing the `/etc/network/interfaces`:

```
auto eth0
iface eth0 inet static
```

Restart the isolated container and check the vpn connection with curl. For testing stop the proxy container and try to connect somewhere again, it should not work.

5. See also

- running openvpn inside lxc containers ⟨`https://web.archive.org/web/20190428024612/heider.io/blog/2013/10/26/openvpn-in-a-lxc-container/`⟩
- **lxc.container.conf**(5)
- **iptables-extensions**(8)
- **tcpdump**(8)
- **systemd.service**(5)