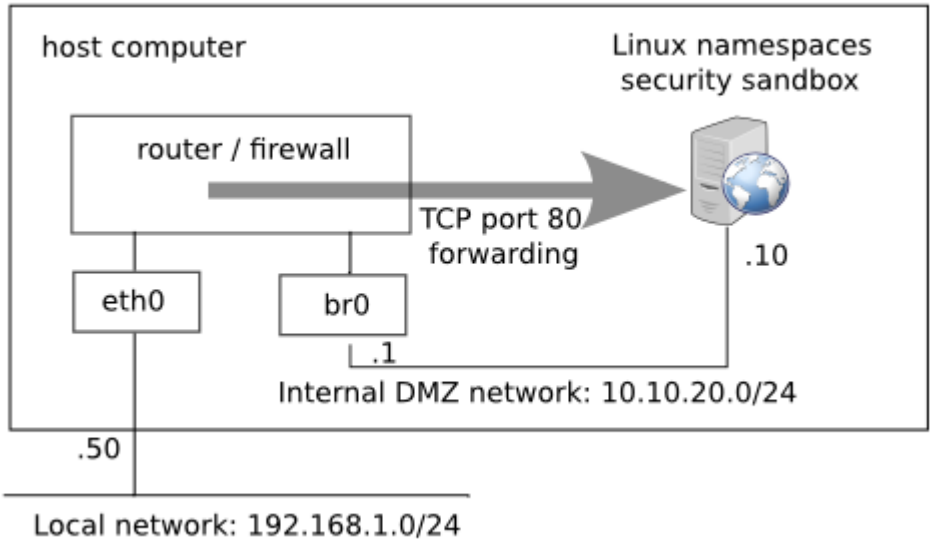
Securing a Web Server Using a Linux Namespaces Sandbox

The goal of this article is to isolate a small public web server on a simulated demilitarized zone (DMZ) network, and to restrict the local network access in case the server is breached. It is an extra security layer added to an existing home server setup.



Internal DMZ network setup

The DMZ consists of an internal network 10.10.20.0/24 connected to br0 bridge device. On this network I place a Linux namespaces security sandbox at 10.10.20.10, running a web server. In case an intruder gets control of the web server, he will be running with low privileges as a generic *www-data* user. The host firewall configuration will not allow him to open connections anywhere outside DMZ network.

To build the sandbox I use Firejail on a Debian 7 computer. For any other Linux distribution the setup steps are similar. All the commands specified below are executed as *root*.

Step 1: Install Firejail

The download page provides source code (*./configure && make && sudo make install*), deb (*dpkg -i firejail.deb*) and rpm (*rpm -i firejail.rpm*) packages. The project page also lists an Arch Linux package. The software has virtually no dependencies and it will work with any 3.x Linux kernel.

Step 2: Install nginx web server

Install nginx or any other web server you are familiar with. Stop the server and make sure it is not started by default at power up:

```
# apt-get install nginx
# /etc/init.d/nginx stop
Stopping nginx: nginx.
# insserv -r nginx
```

On Debian and Ubuntu, nginx serves pages from */usr/share/nginx/www* directory. The logs are stored in */var/log/nginx/*.

Step 3: Configure 10.10.20.0/24 network

Create and configure the internal 10.10.20.0/24 network. The host interface br0 has the address 10.10.20.1:

```
# apt-get install bridge-utils
# brctl addbr br0
# ifconfig br0 10.10.20.1/24
```

You should have in this moment the bridge interface up an running:

```
# ifconfig br0
br0       Link encap:Ethernet  HWaddr e6:55:ca:1c:29:4a
          inet addr:10.10.20.1  Bcast:10.10.20.255  Mask:255.255.255.0
          inet6 addr: fe80::e455:caff:fe1c:294a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

```
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:468 (468.0 B)
```

Step 4: Host firewall configuration

Forward TCP port 80 on the host to TCP port 80 in sandbox, and drop all traffic originated on DMZ network. Also, enable routing on the host. My iptables script is as follows:

```
1  #!/bin/bash
2  # netfilter cleanup
3  iptables --flush
4  iptables -t nat -F
5  iptables -X
6  iptables -Z
7  iptables -P INPUT ACCEPT
8  iptables -P OUTPUT ACCEPT
9  iptables -P FORWARD ACCEPT
10
11 # enable ipv4 forwarding
12 echo "1" > /proc/sys/net/ipv4/ip_forward
13
14 # forward host tcp port 80 to our sandbox
15 iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to
   10.10.20.10:80
16
17 # drop any traffic originated on 10.10.20.0/24 network
18 iptables -A FORWARD -i br0 -m state --state NEW,INVALID -j DROP
19 iptables -A INPUT -i br0 -m state --state NEW,INVALID -j DROP
```

Step 5: Start the sandbox

Start the server using Firejail:

```
# firejail --private --net=br0 --ip=10.10.20.10 \
"/etc/init.d/rsyslog start; \
/etc/init.d/nginx start; \
sleep inf" &
```

The command configures the bridge device and the IP address. It starts a syslog server and nginx inside the sandbox. Use *sleep inf* to keep the session open indefinitely.

Sandbox isolation:

- **Filesystem:** The sandbox has isolated the filesystem by making all directories read-only. Option *–private* removes /root, /home and /tmp directories. Among other things, this also isolates X11 socket and prevents any kind of snooping on X11 sessions.
- **Process space:** The only processes visible in the sandbox are the processes started in the sandbox (syslog and web server). This prevents attacks such as strace attack. You can get a list of the processes running in sandbox using *firejail –list* command:

```
# firejail --list
3867:root:firejail --private --net=br0 --ip=10.10.20.10 /etc/init...
  3868:root:bash -c /etc/init.d/rsyslog start; /etc/init.d/nginx st...
    3885:root:/usr/sbin/rsyslogd -c5
    3912:root:nginx: master process /usr/sbin/nginx
      3913:www-data:nginx: worker process
      3914:www-data:nginx: worker process
      3916:www-data:nginx: worker process
      3917:www-data:nginx: worker process
    3915:root:sleep inf
#
```

- **Network stack:** The sandbox uses a separate network stack, with different interfaces, its own routing table and firewall, and its own set of socket connections. The host firewall

was set to forward TCP port 80 traffic to our sandbox, and to drop any connections originated on 10.10.20.0/24 network segment.

Step 5: Monitoring the web server

Check the server using the log files in /var/log directory inside the sandbox. To reach them you would have to join the sandbox using *firejail –join* command:

```
# firejail --join=3868
[root@debian ~]$
```

Specify the PID number for one of the processes running in sandbox as *–join* argument. The option will only work on Linux kernels 3.8 or newer, and it is equivalent to a terminal login. If you are using an earlier kernel, add an ssh server to your sandbox:

```
# firejail --private --net=br0 --ip=10.10.20.10 \
"/etc/init.d/rsyslog start; \
/etc/init.d/ssh start; \
/etc/init.d/nginx start; \
sleep inf" &
```

Conclusion

I start with a mainstream web server (nginx) running on one of the most popular web server platforms (Debian). This provides a secure baseline for my setup. I sandbox the server using Linux namespaces feature in Linux kernel, thus increasing the security of the setup.

This is a comparison of a regular server setup and a restricted server setup in a security sandbox:

	Regular setup	Security sandbox setup
Filesystem	Read-write	Configurable, mostly read-only

Process table	Access to all running processes	Access only to processes running in the sandbox
Network	Full local network access	Controlled local network access

The same solid server security practices are required for both the regular and the sandbox case. An attacker gaining unauthorized root access is bad in both cases.

Source : http://l3net.wordpress.com/2014/06/08/securing-a-web-server-using-a-linux-namespaces-sandbox/