Tunneldigger Documentation

Release latest

wlan slovenija

Mar 08, 2021

Contents

Tunneldigger is one of the projects of wlan slovenija open wireless network. It is a simple VPN tunneling solution based on the Linux kernel support for L2TPv3 tunnels over UDP.

Tunneldigger consists of a client and a server portion.

The client is written in C for minimal binary size and optimized to run on embedded devices such as wireless routers running OpenWrt.

The server portion, referred to as the broker, is written in Python.

CHAPTER 1

Contents

1.1 Server (Broker) Installation

The installation of Tunneldigger's server side (broker) is pretty straightforward and is described in the following sections.

1.1.1 OpenWrt Package

If you want to run Tunneldigger's server side on OpenWrt, you can use the opkg package.

You can add the whole repository as an OpenWrt feed and add package to your firmware.

1.1.2 Getting the Source

If you want to run Tunneldigger from source, it can be retrieved from its GitHub repository by running the following command:

```
git clone git://github.com/wlanslovenija/tunneldigger.git
```

This will give you a `tunneldigger` directory which contains the broker and the client in separate directories. Server installations only need the broker.

Warning: `master` branch is not necessary stable and you should not be using it in production. Instead, use the latest release. See history for the list of changes.

1.1.3 Operating System

The first thing you need is a recent (>= 5.2.17, 5.3.1, 5.4+) Linux kernel that supports L2TPv3 tunnels. You can find out your linux kernel version using the command `uname -a`. Older kernels unfortunately have a bug that makes

them not work properly. For those kernel, we are maintaining the legacy branch.

We assume the following instructions to work on the distributions listed below. You are welcome to add your distribution if the instructions work and to edit them to make them work.

- Debian
- Fedora with the package `kernel-modules-extra`
- *add your distribution*

1.1.4 Kernel Modules

The following modules are required for Tunneldigger operation:

- `l2tp_core`
- `l2tp_eth`
- `l2tp_netlink`

Note: Fedora and RHEL/CentOS blacklist some of these modules by default for security reasons. Ensure you remove the blacklisting from `/etc/modprobe.d/l2tp_eth-blacklist.conf` and `/etc/modprobe.d/l2tp_netlink-blacklist.conf`.

Kernel Module Activation on Boot

For the activation of the kernel modules, we recommend adding a file `/etc/modules-load.d/tunneldigger.conf` with the following content:

```
l2tp_core
l2tp_eth
l2tp_netlink
```

Manual Activation of the Kernel Modules

You can also activate the modules using `modprobe`. A system restart will *not* load the modules again if you do not create the file as described above.

```
$ sudo modprobe l2tp_core
$ sudo modprobe l2tp_eth
$ sudo modprobe l2tp_netlink
```

Check if Modules are Loaded

You can find out if these modules are loaded by running `lsmod | grep l2tp`. If you get no output, they are not activated. If the modules were loaded successfully, your listing of the modules might look like this:

```
$ lsmod | grep l2tp
l2tp_eth               16384  0
l2tp_netlink           24576  1 l2tp_eth
l2tp_core              32768  2 l2tp_eth,l2tp_netlink
```

(continues on next page)

(continued from previous page)

```
ip6_udp_tunnel         16384  1 l2tp_core
udp_tunnel             16384  1 l2tp_core
```

1.1.5 System Packages

Also the following Debian packages are required:

- `iproute`
- `bridge-utils`
- `python-dev`
- `libevent-dev`

If you would like to use the already supplied hook scripts to setup the network interfaces, you also need the following packages:

- `ebtables`

Since `ebtables` is also a kernel module, please activate it as described in the *Kernel Modules* Section.

Note that the best way to run any Python software is in a virtual environment (virtualenv), so the versions you have installed on your base system should not affect the versions that are installed for Tunneldigger.

You can install all of the above simply by running on Debian:

```
sudo apt-get install iproute bridge-utils python-dev libevent-dev ebtables python-
↪virtualenv
```

and for Fedora you can use this command:

```
sudo yum install iproute bridge-utils python-devel libevent-devel ebtables libnl-
↪devel python-pip python-virtualenv
```

1.1.6 Installation

If we assume that you are installing Tunneldigger under `/srv/tunneldigger` (the scripts provided with Tunneldigger assume that as well), you can do:

```
cd /srv/tunneldigger
virtualenv -p /usr/bin/python3 env_tunneldigger
```

Note: Tunneldigger only supports Python 3.

Using the above command ensures the virtualenv is created using a Python 3 interpreter. In case the Python 3 interpreter you would like to use is not located at `/usr/bin/python3` you will have to adjust the path accordingly. If your distribution uses Python 3 by default you can usually omit the `-p` parameter.

This creates a virtual Python environment in the `env_tunneldigger` folder. This folder can be deleted and recreated as needed using the virtualenv command, should this be required. You can then checkout the Tunneldigger repository into `/srv/tunneldigger/tunneldigger` by doing:

```
cd /srv/tunneldigger
git clone https://github.com/wlanslovenija/tunneldigger.git
```

Next you have to enter the environment and install the broker alongside its dependencies:

```
source env_tunneldigger/bin/activate
cd tunneldigger/broker
python setup.py install
```

1.1.7 Configuration

The broker must be given a configuration file as first argument, an example of which is provided in `l2tp_broker.cfg.example`. There are some options that must be changed and some that can be left as default:

- **address** should be configured with the external IP address that the clients will use to connect with the broker.
- **port** should be configured with the external port (or ports separated by commas) that the clients will use to connect with the broker.
- **interface** should be configured with the name of the external interface that the clients will connect to.
- Hooks in the **hooks** section should be configured with paths to executable scripts that will be called when certain events occur in the broker. They are empty by default which means that tunnels will be established but they will not be configured.

Hook scripts that actually perform interface setup. Examples that we use in production in *wlan slovenija* network are provided under the `scripts/` directory. The configuration file must contain absolute paths to the hook scripts and the scripts must have the executable bit set.

There are currently four different hooks, namely:

- `session.up` is called after the tunnel interface has been created by the broker and is ready for configuration at the higher layers. (Example of such a script is found under `scripts/session.up.sh`.)
- `session.pre-down` is called just before the tunnel interface is going to be removed by the broker. Notice that hooks are executed asynchonously, so by the time this script runs, the interface may already be gone. Thus you probably want to use *session.down* instead.
- **`session.down` is called after the tunnel interface has been destroyed and is no longer available.** (Example is found under `scripts/session.down.sh`.)
- `session.mtu-changed` is called after the broker's path MTU discovery determines that the tunnel's MTU has changed and should be adjusted. (Example is found under `scripts/mtu_changed.sh`.)
- `broker.connection-rate-limit` is called when a IP address tries to connect `connection_rate_limit_per_ip_count` times within `connection_rate_limit_per_ip_time` seconds. (Example is found under `scripts/broker.connection-rate-limit.sh`.)

Please look at all the example hook scripts carefully and try to understand them before use. They should be considered configuration and some things in them are hardcoded for our deployment. You will probably have some different network configuration and so you should modify the scripts to suit your setup. The examples also document the command-line argument passed to the hooks.

Example hook scripts present in the `scripts/` subdirectory are set up to create one bridge device per MTU and attach L2TP interfaces to these bridges. They also configure a default IP address to newly created tunnels, set up `ebtables` to isolate bridge ports and update the routing policy via `ip rule` so traffic from these interfaces is routed via the `mesh` routing table.

- Each tunnel established with the broker will create its own interface. Because we are using OLSRv1, we cannot dynamically add interfaces to it, so we group tunnel interfaces into bridges.

- We could put all tunnel interfaces into the same bridge, but this would actually create a performance problem. Different tunnels can have different MTU values – but there is only one MTU value for the bridge, the minimum of all interfaces that are attached to that bridge. To avoid this problem, we create multiple bridges, one for each MTU value – this is what the example scripts do.
- We also configure some `ip` policy rules to ensure that traffic coming in from the bridges gets routed via our `mesh` routing table and not the main one (see `bridge_functions.sh`). Traffic between bridge ports is not forwarded (this is achieved via `ebtables`), otherwise the routing daemons at the nodes would think that all of them are directly connected – which would cause them to incorrectly see a very large 1-hop neighbourhood. This file also contains broker-side IP configuration for the bridge which should really be changed.

Note that you do not actually need to have the same configuration, this is just something that we are using at the moment in *wlan slovenija* network. The scripts should be very flexible and you can configure them to do anything you want/need.

The example hook scripts require that the routing daemon (like `olsrd`) be configured with the Tunneldigger bridge interfaces.

1.1.8 Running

After you configured Tunneldigger, you can run the broker:

```
cd /srv/tunneldigger
/srv/env_tunneldigger/bin/python -m tunneldigger_broker.main /srv/tunneldigger/broker/
↪l2tp_broker.cfg
```

1.2 Client Installation

1.2.1 Getting the Source

Tunneldigger source can be retrieved from its Github repository by running the following command:

```
git clone git://github.com/wlanslovenija/tunneldigger.git
```

This will give you a `tunneldigger` directory which contains the broker and the client in separate directories. Client code can be compiled into a stand-alone program.

1.2.2 OpenWrt Package

Currently supported way to compile and deploy a client is through an OpenWrt package. Source code for such OpenWrt package can be found here.

You can add the whole repository as an OpenWrt feed and add package to your firmware.

1.2.3 Configuration

- **MAX_BROKERS** (default: 10): Maximum number of brokers that can be handled in a single process.

  make CFLAGS="-D MAX_BROKERS=20"

CHAPTER 2

Source Code and Issue Tracker

Development happens on GitHub and issues can be filed in the Issue tracker.

CHAPTER 3

License

Tunneldigger is licensed under AGPLv3.

CHAPTER 4

Contributions

We welcome code and documentation contributions to Tunneldigger in the form of Pull Requests on GitHub where they can be reviewed and discussed by the community. We encourage everyone to check out any pending pull requests and offer comments or ideas as well.

Tunneldigger is developed by a community of developers from many different backgrounds.

You can visualize all code contributions using GitHub Insights.

CHAPTER 5

Indices and Tables

- genindex
- search

CHAPTER 6

Related Work

- German Tunneldigger Tutorial by Freifunk Franken