# rVRRPd Documentation

**Nicolas Chabbey**

**Mar 05, 2021**

# Contents

**rVRRPd** is a fast, secure and standard compliant implementation of the high-availability VRRP protocol. It is very scalable, and can run on multiple platforms and operating systems.

As its name implies, **rVRRPd** can run as a Unix daemon or as a standalone program. It can also exposes a RESTful API for monitoring and configuration purposes, enabling Software Defined Networking (SDN) applications.

CHAPTER 1

# Features

**rVRRPd supports a number of innovative features:**

- Secure software architecture leveraging the Rust programming language
- Highly scalable; up to several hundreds of concurrent VRRP groups
- Supports standard RFC3768 and RFC2338, `Simple Password` authentication
- Supports additional *proprietary authentication* methods
- Supports multiple operating systems and processors architectures
- Provides a RESTful Client Application Programming Interface (API)
- Provides a plain-text HTTP or SSL/TLS HTTPS interface to the Client API
- Leverages additional features such as `macvlan` and `Linux Socket Filters`

## 1.1 Features Support Matrix

| Supported Features | Linux | FreeBSD |
|---|---|---|
| Multiple Listeners Threads | Yes | Yes |
| RESTful Client API | Yes | Yes |
| Socket Filters (eBPF) | Yes | No |
| MAC-Based Virtual LAN Interface (`macvlan`) | Yes | No |
| Static Routing | Yes | No |

CHAPTER 2

# Configuration Guide

This part of the documentation focuses on the step-by-step installation instructions of the daemon and on how to configure the latter for various network and high-availability scenarios.

## 2.1 Introduction

By default, **rVRRPd** reads the `/etc/rvrrpd.conf` configuration file. This file holds all the configuration elements needed for the proper operation of the daemon, the virtual routers, and their related functions.

At this time of writing, both TOML (default) and the JSON formats are supported for the main configuration file. The former is usually simpler to understand and to write, greatly reducing human errors. JSON based configurations however, are harder to write and to parse for some people, but may be more practical when used with automation tools or with an HTTP based Application Programming Interface (API).

If you don't know which configuration file format to use, we recommend to stick with TOML, unless you want to use the Client API extensively.

The **rVRRPd** daemon runs one `virtual-router` per `interface, group` pair, which means you can configure the same VRRP groups id or `virtual-router` id across several physical interfaces. The daemon can scale to hundreds if not thousands of active virtual-routers if the CPU and memory resources permit.

The initial *developer* of **rVRRPd** has chosen to build the daemon entirely using the Rust programming language. Rust is a language, aimed primarily at security and speed. You get all the benefits of a modern object-oriented programming language such as Java or C++, without their respective performance penalty and inherent security risks.

We tried to keep `unsafe` blocks as small as possible in order to provides a clean interface to unsafe functions. However, we cannot removes all of them as they are necessary to implement functions calls to the standard C library, and to the various interfaces (such as IOCTLs) to the operating system kernel.

We hope that you will enjoy running **rVRRPd** and you would be able to solve your current network and high-availability challenges in less time and thus without the hassles commonly found in commercial solutions.

This project wouldn't be live without the dedication of its developers, and the open source community. Any contributions are greatly welcome, and will help us developing new features and a more stable and secure implementation.

### 2.1.1 Developpers

- Nicolas Chabbey
  - Keybase: @e3prom
  - PGP Public Key Fingerprint: `DBD4 3BD8 81F3 C3E2 37E1 9E54 D7FF 004E 2E22 CF1C`

### 2.1.2 Sponsorship

You can help us directly by donating to the project.

Every single penny will cover the development cost of **rVRRPd**, which is comprised of a lot of coffee, and the power bill of the bare-metal servers running the interoperability and testing labs.

You can donate by Paypal, or by using a crypto-currency listed below:

| Crypto Currency | Wallet Address |
|---|---|
| Bitcoin (BTC) | 3Pz7PQk5crAABg2MsR6PVfUxGzq2MmPd2i |
| Etherum (ETH) | 0x0686Dd4474dAA1181Fc3391035d22C8e0D2dA058 |

### 2.1.3 Software License

```
Lightweight, fast, and highly secure VRRP daemon
Copyright (C) 2019-2021  Nicolas Chabbey

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program <LICENSE-GPLv3>.
If not, see <https://www.gnu.org/licenses/>.

Additional permission under GNU GPL version 3 section 7

If you modify this Program, or any covered work, by linking or
combining it with OpenSSL (or a modified version of that library),
containing parts covered by the terms of OpenSSL <LICENSE-OpenSSL>,
the licensors of this Program grant you additional permission
to convey the resulting work.
```

## 2.2 Install rVRRPd

**rVRRPd** can be installed from source or by using pre-compiled binaries packages. The latter is recommended for production uses, as the executables have been previously tested for stability.

### 2.2.1 Software Requirements

- The Linux or FreeBSD operating system (64 bits)
- The OpenSSL library
- The Netlink Protocol Library Suite library *(Linux)*

### 2.2.2 Hardware Requirements

- An Intel IA-64 (x86_64) or ARMv8 (aarch64) processor
- At least **one** Ethernet interface

### 2.2.3 Source Installation

#### Getting Started

To install **rVRRPd** from source, first of all, make sure you have all the required build dependencies (see *Building Dependencies* section below).

Then download the source tarball files (tar.gz) from our release page or use git to clone the source repository.

Below we will describe the step-by-step instructions on how to install a stable release of the daemon and its utilities:

#### Building Dependencies

**To build rVRRPd from source you must have several programs and libraries installed on your system (preferably system-wide):**

- Rust Cargo (v1.33.0 or later), to build the project and its related dependencies (crates).
- The OpenSSL development headers
- The Netlink Protocol Library Suite development headers *(Linux)*

On Debian and derivatives, all three libraries' headers files can be installed with the below command:

```
$ sudo apt-get install libnl-3-dev libnl-route-3-dev libssl-dev
```

#### Cloning Source Repository

We will now clone the source from our official github repository:

```
$ git clone https://github.com/e3prom/rvrrpd
Cloning into 'rvrrpd'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 1301 (delta 4), reused 12 (delta 4), pack-reused 1285
Receiving objects: 100% (1301/1301), 347.88 KiB | 0 bytes/s, done.
Resolving deltas: 100% (831/831), done.
```

### Switching to Stable Release

We move to the `rvrrpd` directory just created by git and we will switch to the latest stable release (here version `0.1.3`):

```
$ cd rvrrpd
$ git checkout tags/0.1.3
[...]
```

### Invoking the Build Process

Enter the `make` command to start the build process. Rust Cargo will automatically fetch and build all the required dependencies and will start the build process of the **rVRRPd** daemon and related utilities such as `rvrrpd-pw`:

```
$ make
Updating crates.io index
[...]
Compiling rVRRPd v0.1.3 (/var/tmp/rvrrpd)
    Finished release [optimized] target(s) in 2m 40s
```

Once the build process is completed, you can find the daemon executable in `target/release/rvrrpd`. The latter can be executed as-is or can be installed system-wide (recommended).

### Installing System-wide

We will now install `rvrrpd`, its accompanying configuration file `/etc/rvrrpd.conf`, and the `rvrrpd-pw` utility in our system paths by using the `make install` command (requires root privileges):

```
$ sudo make install
make -C utils/rvrrpd-pw install
make[1]: Entering directory 'utils/rvrrpd-pw'
if [ ! -d /usr/bin ]; then \
    mkdir -p /usr/bin; \
fi
cp target/release/rvrrpd-pw /usr/bin/rvrrpd-pw
chmod 755 /usr/bin/rvrrpd-pw
make[1]: Leaving directory 'utils/rvrrpd-pw'
if [ ! -d /usr/sbin ]; then \
    mkdir -p /usr/sbin; \
fi
cp target/release/rvrrpd /usr/sbin/rvrrpd
chmod 755 /usr/sbin/rvrrpd
if [ ! -d /etc/rvrrpd ]; then \
    mkdir -p /etc/rvrrpd; \
fi
```

### Configuring

Prior to running the daemon, you must edit the main configuration file according to your network or high-availability environment. See *Configure* below for a basic sample configuration example.

#### Running

**rVRRPd** supports multiple operating modes: it can run in `foreground` mode from a terminal or in `background` mode as a standard Unix daemon, using the `-m1` and `-m2` switches, respectively.

> **Warning:** The daemon requires root privileges to run successfully. The daemon must have access to raw sockets, and to privileged kernel functions to create virtual interfaces, IP addresses and routes.

In the below example, we are running the daemon in `foreground` mode using the `-m1` switch:

```
$ sudo rvrrpd -m1
```

### 2.2.4 Binary Package Installation

**rVRRPd** could also be installed directly from binaries packages. This is the recommended way of installing the VRRP daemon for production uses as we are testing every executable for stability prior to shipping the releases to the public.

#### Getting Binary Archives

Visit the official release page on github and download the latest package in `tar.xz` format.

You can download directly from the command-line using the `wget` utility:

```
$ wget "https://github.com/e3prom/rVRRPd/releases/download/0.1.3/rvrrpd-0.1.3-linux-
↪amd64.tar.xz"
```

#### Verifying the Archives Integrity

Prior to unpacking the archive, we strongly suggest to verify the file checksum to ensure it has not be tempered by a third party.

```
$ wget "https://github.com/e3prom/rVRRPd/releases/download/0.1.3/SHA256SUMS"
$ sha256sum --check SHA256SUMS
rvrrpd-0.1.3-linux-amd64.tar.xz: OK
```

#### Unpacking Archives

Untar the downloaded archive using `tar`:

```
$ tar -xvf rvrrpd-0.1.3-linux-amd64.tar.xz
rvrrpd-0.1.3-linux-amd64/
rvrrpd-0.1.3-linux-amd64/README.md
rvrrpd-0.1.3-linux-amd64/conf/
rvrrpd-0.1.3-linux-amd64/conf/rvrrpd.conf
rvrrpd-0.1.3-linux-amd64/conf/rvrrpd.json.conf
rvrrpd-0.1.3-linux-amd64/rvrrpd
rvrrpd-0.1.3-linux-amd64/LICENSE
```

### Configuring

Move into the release `rvrrpd-<version>-<os>-<arch>/` directory just created above:

```
$ cd rvrrpd-0.1.3-linux-amd64/
```

*Edit* the sample configuration file in `etc/rvrrpd.conf` and run the daemon from the current directory:

### Running

> **Warning:** The daemon requires root privileges to run successfully. The daemon must have access to raw sockets, and to privileged kernel functions to create virtual interfaces, IP addresses and routes.

```
$ sudo ./rvrrpd -m1 -c conf/rvrrpd.conf
```

See our configuration reference for more information about the available configuration options.

## 2.2.5 Basic Configuration Example

rVRRPd read its configuration file from the default `/etc/rvrrpd.conf`. The later, must be configured to match your current network and high-availability configuration. You can also overwrite the config file path using the `-c` or `--conf` command-line switches.

Below a sample TOML configuration file of a basic VRRP first-hop router:

Listing 1: rvrrpd.conf

```
debug = 5
pid = "/var/tmp/rvrrpd.pid"
working_dir = "/var/tmp"
main_log = "/var/tmp/rvrrpd.log"
error_log = "/var/tmp/rvrrpd-error.log"
client_api = "http"

[[vrouter]]
group = 1
interface = "ens192.900"
vip = "10.100.100.1"
priority = 254
preemption = true
rfc3768 = true
netdrv = "libnl"
iftype = "macvlan"
vifname = "vrrp0"
auth_type = "rfc2338-simple"
auth_secret = "thissecretnolongeris"

[protocols]
  [[protocols.static]]
    route = "0.0.0.0"
    mask = "0.0.0.0"
    nh = "10.240.0.254"

```

(continues on next page)

(continued from previous page)

```
[api]
  tls = false
  host = "0.0.0.0:7080"
  users = [ "{{SHA256}}admin:0:1eb7ac761a1201f9:095820af..." ]
```

**The above configuration do the following:**

- Starts the daemon in foreground mode with a debug level of `5`(extensive).
- Enable the Client API with the `http` listener (listen by default on `tcp/7080`).
- Runs one virtual-router with group id `1` on interface `ens192.900`, with the below parameters:
  - Uses the virtual IP address `10.100.100.1`.
  - Is configured with the highest priority of `254`.
  - Has preemption enabled.
  - Has compatibility with RFC3768 turned on (may be required to fully interoperate with some equipment vendors).
  - Uses the network driver `libnl` which leverage the netlink protocol. Alternatively, you can use the `ioctl` driver, which is simpler but will removes the interface's IP addresse(s) for the VIP when in Master state.
  - Is configured for a `macvlan` type interface, a MAC-based virtual interface.
  - Name the child virtual interface `vrrp0`, the latter will be used to hold the virtual router IP address.
  - Set authentication to the RFC2338 , `Simple Password` authentication method.
  - Set the secret key (or password) to be shared between the virtual routers.
- When Master, install a static default route with a next-hop of `10.240.0.254`.
- The Client API only authorizes queries from the users listed in the `users` list under the `[api]` section. The users must authenticate prior to accessing the virtual router's information.
  - You can generate users passwords hashes using the rvrrpd-pw utility.

You can consult our configuration guide to have more details and explanation about all the available configuration options.

## 2.3 Configuration Reference

### 2.3.1 General Directives

#### debug

**Description** The verbose or debugging level.

**Value type** Decimal

**Default** 0

The `debug` directive sets the debugging (or verbosity) level of the daemon.

**Possible values are:**

- `0` Information

- `1` Low
- `2` Medium
- `3` High
- `5` Extensive

### time_zone

**Description** The timestamps reference time zone

**Value type** String

**Default** local

The `time_zone` directive sets the reference time zone for the various daemon timestamps.

**Possible values are:**

- **`local` for Local Time (LT)** This setting uses the locally configured time zone of the operating system.
- **`utc` for Coordinated Universal Time (UTC)** Timestamps will be given in UTC or Zulu time.

### time_format

**Description** The timestamps time format

**Value type** String

**Default** disabled

The `time_format` directive sets the reference time format for the various daemon timestamps.

**Possible values are:**

- `disabled` for no particular time format (use the default time format)
- `short` for a shortened, more concise time format
- `rfc2822` for the standard RFC2822, Internet Time Format

### pid

**Description** The daemon's PID file path

**Value type** String

**Default** /var/run/rvrrpd.pid

The `pid` directive sets the full or relative path to the daemon's PID file.

### working_dir

**Description** The daemon's working directory

**Value type** String

**Default** /tmp

The `working_dir` directive sets the daemon's working directory. The daemon's user must have read access to this directory.

#### main_log

**Description** Path to the daemon's main log file

**Value type** String

**Default** /var/log/rvrrpd.log

The `main_log` directive sets the path to the daemon's main log file.

#### error_log

**Description** Path to the daemon's error log file

**Value type** String

**Default** disabled

The `error_log` directive sets the path to the daemon's error log file. Any errors occuring during the runtime are written to this log file.

#### client_api

**Description** Client API interface type

**Value type** String

**Default** http

The `client_api` directive sets the Client API interface type.

**Possible values are:**

- **`http` for the RESTful HTTP interface** This value enable a plain-text HTTP or HTTPS (SSL/TLS) interface to the client API. It does include user authentication and a secure communication channel when SSL/TLS is enabled.

New in version 0.1.3: Directive added with Client API Support

### 2.3.2 Virtual Routers Directives

#### group

**Description** Virtual Router Group ID (VRID)

**Value type** Integer

**Default** *none*

The `group` directive sets the VRRP group id or virtual-router id (VRID).

**Valid values are:**

- `0-255` The VRRP group id or virtual-router id. Usually matches the sub-interface unit number or interface's vlan id.

### interface

**Description** Interface to run VRRP on

**Value type** String

**Default** *none*

The `interface` directive sets the VRRP virtual-router's interface. Only Ethernet interfaces are supported.

### iftype

**Description** Interface type

**Value type** String

**Default** *none*

The `iftype` directive sets the VRRP virtual-router's interface type. By default, the daemon will directly work with the configured running interface, and therefore may change its IP and/or MAC address(es).

**Valid values are:**

- `macvlan` Use a MAC-Based Virtual LAN interface.

New in version 0.1.1: Directive added with MAC-Based Virtual LAN Interface Support

### vip

**Description** Virtual IP Address

**Value type** String

**Default** *none*

The `vip` directive sets the VRRP standby address or virtual-router address. Only IPv4 addresses are currently supported at this time.

### priority

**Description** Virtual Router Priority

**Value type** Integer

**Default** 100

The `priority` directive sets the virtual-router VRRP priority.

**Valid values are:**

- `1-254` The VRRP virtual router priority. Values 0 and 255 are reserved as per RFC3768 and cannot be configured manually.

### preemption

**Description** Preemption Support

**Value type** Boolean

**Default** false

The `preemption` directive sets if preemption is enabled. By default, preemption is turned off; a higher-priority virtual router cannot preempt an active Master.

**Valid values are:**

- `true` Preemption is turned on, a higher-priority Standby virtual router can preempt the current Master virtual router.
- `false` Preemption is turned off.

### auth_type

**Description** Authentication Type

**Value type** String

**Default** *none*

The `auth_type` directive sets the VRRP group's authentication type. Authentication allow to authenticate VRRP messages and with some types allow to verify their integrity. Authentication can prevent a misconfigured VRRP virtual router to take over the Master, resulting in the blackhole or interception of the user network traffic.

**Valid values are:**

- `rfc2338-simple` for RFC2338 Simple Password Authentication.
- `p0-t8-sha256` for proprietary P0 Authentication. Uses a SHA256 HMAC of the VRRP messages. This type provides both messages authentication and integrity.
- `p1-t8-shake256` for proprietary P1 Authentication. Uses the SHAKE256 Extendable-Output Function (XOF). This type provides both messages authentication and integrity.

### auth_secret

**Description** Authentication Secret

**Value type** String

**Default** *none*

The `auth_secret` directive sets the VRRP group's authentication secret or password. Ensure all virtual routers among the configured group share the same secret and that the latter has been transmitted securely.

> **Warning:** Keep in mind that the configuration file holds the secret, therefore only authorized users should be able to read it.

### rfc3768

**Description** RFC3768 Compatibility Warning Flag

**Value type** Boolean

**Default** true

The `rfc3768` directive allow you to force the compatibility flag. The meaning of this flag may be confusing, and can be safely ignored most of the time. When this flag is set to `true`, it indicates the

virtual router may **NOT** operates entirely according to the applicable VRRP RFCs. In particular regarding to the authentication and to the length of some VRRP PDUs header fields. When this flag is `true`, the virtual router may not be interoperable with third-party, standard-compliant devices or softwares.

---

**Note:** Enabling proprietary features such as the proprietary authentication types, will automatically turn this flag on.

---

**Valid values are:**

- `true` to forcibly enable non-standard operations.
- `false` to forcibly disable non-standard operations whenever possible.

### netdrv

**Description** Network Driver

**Value type** String

**Default** ioctl

The `netdrv` directive specify which network driver to uses for the virtual-router. The available drivers depend on the operating system and slight differences do exists between them. The driver is used partially or entirely to; add the virtual IP addresses, create the virtual interface, change the interface's MAC address, or to update the kernel routes.

**Valid values are:**

- `ioctl` for using IOCTLs. This option should be supported in all Linux based operating systems, even with the presence of an old kernel.
- `libnl` for using the Netlink Protocol Library which is an intermediate API to communicate with the Linux Netlink protocol. The latter is a modern and robust way of configuring and interrogating the kernel.

  ---

  **Note:** We strongly suggest to keep using this driver whenever possible. When using `macvlan` interfaces, this driver is automatically enabled.

  ---

### vifname

**Description** Virtual Interface Name

**Value type** String

**Default** standby<*group-id*>

The `vifname` directive sets the virtual-router's virtual interface name. By default, the virtual interface is named using the `standby<group-id>` format, where `group-id` correspond to the virtual-router's VRRP group id or VRID.

---

**Note:** This directive is only used when virtual interface support is activated. (e,g. by having the *iftype* directive set to `macvlan`).

---

New in version 0.1.1: Directive added with MAC-Based Virtual LAN Interface Support

#### socket_filter

**Description** Socket Filter Support

**Value type** String

**Default** true

The `socket_filter` directive allow you to enable or disable the use of Socket Filters. On Linux, eBPF based Socket Filters allow every virtual-router raw sockets to only receives VRRP traffic matching their interface and VRRP group, thus greatly improving performance.

**Valid values are:**

- `true` for enabling support for socket filters. Drastically improves the listener threads performance by allowing the kernel to filter out unwanted traffic not to be processed by the listening thread.
- `false` for disabling support for socket filters.

New in version 0.1.2: Directive added with Linux Socket Filters Support

### 2.3.3 API Directives

#### users

**Description** API Users

**Value type** List of Strings

**Default** *none*

The `users` directive lists the user accounts authorized for the Client API. Every string in the list must adhere to strict formatting rules and can be easily generated using the `rvrrpd-pw` utility.

#### secret

**Description** API Secret

**Value type** String

**Default** 128-bits random number

The `secret` directive sets the API secret. This secret is used for a number of cryptogrphic functions and must be kept secret.

By default, at every start of the daemon, a random 128 bits unsigned integer is generated from a secure PRNG. This number is large enough and *SHOULD* have sufficient entropy to provides good security.

You can overwrite this secret by specifiy your own. The secret will be maintained across restart of the *rVRRPd* daemon.

> **Warning:** Improper setting of the secret string can open up vulnerabilities or security holes, such as authentication bypass.

**Note:** If setting the secret manually, please ensure your string is long and random enough to provides *sufficient* security. We strongly recommend to use a random number generator to generate it.

### host

**Description** Listening Host

**Value type** String

**Default** 0.0.0.0:7080

The `host` directive sets the IP address(es) and port for the API interface to listen on. By default it listens on all interfaces on port `7080`.

When setting the Client API Interface to `http` this directive will specify which interfaces and port the HTTP or HTTPS service will listen on.

### tls

**Description** Transport Layer Security (TLS) Support

**Value type** Boolean

**Default** false

The `tls` directive allow you to enable or disable support for SSL/TLS. When using the `http` *Client API Interface*, it will allow you to enable secure HTTPS communication with the API clients.

**Valid values are:**

- `true` for activating Transport Layer Security (TLS) on the API interface.
- `false` for disabling the TLS support.

### tls_key

**Description** SSL/TLS Key File

**Value type** String

**Default** /etc/rvrrpd/ssl/key.pem

The `tls_key` directive allow you to set the full or relative path to the TLS key file.

### tls_cert

**Description** SSL/TLS Certificate File

**Value type** String

**Default** /etc/rvrrpd/ssl/cert.pem

The `tls_key` directive allow you to set the full or relative path to the certificate chain file. At this time of writting, only a valid X.509 server's certificate is necessary.

CHAPTER 3

# Client API Guide

This guide covers the Client Application Programming Interface (API), how to configure it, how to make requests and interprets their various responses.

## 3.1 Introduction to the API

**rVRRPd** provides an Application Programming Interface (API) that allow remote tasks to be performed on the daemon and on the running virtual routers.

The Client API can be accessed by various means, but at this time of writing, only supports the Hypertext Transfer Protocol (HTTP), in plain-text or securely by using a SSL/TLS channel. The use of the latter is highly recommended for integrity and confidentiality purposes.

### 3.1.1 RESTful HTTP Interface

The Client API can be accessed over HTTP using the Representational State Transfer or REST model, which provides a simple and uniform access model to the various data coming from the daemon instance, the VRRP virtual routers, and from the operating system such as interfaces information and kernel routes.

The API not only allow to read data and to parse it efficiently, but also to make modification to the running instance of **rVRRPd**, such as adding a new virtual router, or changing its priority so it can take over a Master router.

---

**Note:** As of version 0.1.3, the Client API is only providing read-only access. Modifications are not yet supported but will be introduced in a later release.

---

To query **rVRRPd** for information, such as the current role of a running VRRP virtual router, a simple HTTP GET request can be made to a specific resource path. If the query can be honored, the API will return a JSON formatted body response with all the attributes and values corresponding to your query.

The responses can be easily and efficiently parsed by both a human and a machine, thus providing a uniform and standardize interface that can be used as a *console*, as an automation interface for SDN applications and much more.

## 3.2 API Reference

**Todo:** The API is still under **active** development. The reference documentation will be available when the API will be stable and ready for production use.

## 3.3 Client API Queries Examples

### 3.3.1 Getting Virtual Router States

#### Getting Started

You can get running information directly from an instance of **rVRRPd** using the HTTP Client API, but first you must authenticate using an HTTP `POST` request to the `auth/` path.

#### Authenticating

The below example shows how to authenticate to the daemon running on `10.0.0.1`, using the `curl` utility:

```
$ curl -k -c /tmp/rvrrpd-api-cookie -d "user=admin passwd=banana" -X POST https://10.
↪0.0.1:7080/auth
```

The above command will send an HTTP `POST` request to the API, and if successful will store the resulting session cookie to `/tmp/rvrrpd-api-cookie`.

#### Requesting VRRP Information

Once authenticated, you can query the router for the current VRRP running information by sending an HTTP `GET` request to the `run/vrrp` resource path:

```
$ curl -k -s -b /tmp/rvrrpd-api-cookie -X GET https://10.0.0.1:7080/run/vrrp | jq
```

You should get a JSON formatted response like below:

```
[
    {
        "virtual_ip": "10.100.100.1",
        "group": 1,
        "interface": "standby1",
        "priority": 254,
        "preempt": true,
        "state": "Master"
    },
    {
        "virtual_ip": "10.100.101.1",
        "group": 2,
        "interface": "standby2",
        "priority": 254,
        "preempt": true,
        "state": "Master"
```

(continues on next page)

(continued from previous page)

```
    }
]
```

CHAPTER 4

# Additional resources

- Github Repository