Radius Vlan

Description of the mechanism used in WR switch
October 2020 (wr-switch-sw-v6.0-58-g0f08e51)

A. Rubini

Table of Contents

Introduction

This document describes a new feature of the White Rabbit Switch, to be used in GSI, which is a subset of IEEE 802.1X.

When a devices is detected on a port which is configured as "access", a Radius server is queried for authorization.

The features is called *radiusvlan* or *rvlan* when a shorter name is to be preferred. It relies on *radclient*, which was added to *buildroot* in a new *freeradius-utils* package. The package installs *radtest*, *radclient* and a minimal dictionary (the full dictionary is more than 1MB worth of data).

I chose to install an older version of radclient because the current version requires a special allocator (libtalloc), that had to be added to buildroot too.

1 Related Kconfig Items

Like most features in White Rabbit Switch, *radiusvlan* is configured through *Kconfig*. The *dot-config* file is used both at build time and at run time (where it lives in */wr/etc*).

This is the list of configuration items related to *radiusvlan*. None of them has effects on the firmware build, they are only used at runtime.

`RVLAN_DAEMON`
: The boolean option selects whether the tool is to be run or not. If disabled, the tool will not run and the related *monit* rule won't be activated. No further config option has any effect if this flag is false.

`RVLAN_PMASK`
: A port mask. If any bit in the mask is 0, the associated port will not be monitored by the tool. Port *wri1* is associated to bit 0, and so on until *wri18* associated to bit 17. Bits 18-31 are ignored. The default value is all-1.

`RVLAN_AUTH_VLAN`
: A temporary VID to be used during port authorization. Defaults to 4094.

`RVLAN_NOAUTH_VLAN`
: The VID to be used for ports that are not authorized. Defaults to 4094.

`RVLAN_OBEY_DOTCONFIG`
: A boolean option. If set, *radiusvlan* will obey the VID value set forth in *dot-config* rather than what the Radius server returned. Thus, the Radius server's reply is only used to authorize or not the port (if not, `NOAUTH_VLAN` is applied).

`RVLAN_RADIUS_SERVERS`
: A comma-separated list of the names or IP addresses of a set of Radius servers. See Chapter 4 [Multiple Servers], page 3.

`CONFIG_RVLAN_RADIUS_SECRET`
: The string used to encrypt radius frames, called "secret" in radius documentation.

2 Service Activation and Monitoring

The tool is a standalone program that ignores its own command line; all configuration information comes from dot-config, as described above.

The service is executed at boot from */etc/init.d/radiusvlan*, which has the same structure of all other similar scripts.

In */etc/rcS*, the symbolic link must be after *vlan* configuration.

The service, like most other *wrs* services, is monitored by *monit*, with the same parameters as all other services. Working *monit* setup can be confirmed by running

```
while true; do killall radiusvlan; sleep 3; done; done
```

which will properly trigger a *monit-triggered* reboot.

Both invocation and monitoring depend on *dot-config*: if `RVLAN_DAEMON` is false, neither of them is activated.

3 Internal Design

The tools enumerates all *wri** interfaces in */sys/net/class*. It opens a *netlink* socket to get notification of any change in interface up/down status, and then checks whether each of them is up or down (thus, no change can get undetected).

Only ports configured as `VLAN_PORTxx_MODE_ACCESS` are monitored, and only if the corresponding bit in `RVLAN_PMASK` is set.

For each monitored port, the tool runs a state machine, where the initial state is *DOWN* or *JUSTUP*. Whenever a state change is reported by *netlink*, the port is moved to either *JUSTUP* or *GODOWN*.

This is the list of states. No state is blocking, so operation on one port does not stop operation on other ports (the engine is based on *select()*).

`RVLAN_GODOWN`
: This state is the default state for enumerated ports, and it is entered whenever *netlink* reports that the interface went down. This state turns the port to *vlan_auth* and forgets the peer's mac address it. It also closes any open file descriptor and kills the child process, if it exists. Thus, no remaining garbage remains in the system even if the fiber is unplugged and re-plugged quickly several times.

`RVLAN_DOWN`
: The port is quietly down.

`RVLAN_JUSTUP`
: The port was just reported as "up" and we must start authentication. The first step is identifying the MAC address of the peer. Thus, the tool starts sniffing the port, but opening a raw socket listening to this port.

`RVLAN_SNIFF`
: Get a frame from the port. If the frame is sent from the switch itlsef, it is ignored. Any address configured in the switch is considered as *self* (i.e., also the *eth0* mac address, used by *ppsi* as sender address, is ignored). When a foreign frame is received, the tool saves the MAC address of the peer, it closes the sniffing socket and it moves to the next state.

`RVLAN_RADCLIENT`
: The tool selects a radius server (see Chapter 4 [Multiple Servers], page 3, and runs *radclient*, feeding data to its *stdin* and collecting its *stdout+stderr*. The *pid* of the child process is retained for later cleanup.

`RVLAN_AUTH`

radclient returned some data. This state collects it until EOF. If *radclient* reports an error in communication, the server is marked as "recently faulty" and *radiusvlan* moves back to `RVLAN_RACLIENT`, where a different server will be selected. If the server replied, the tool looks for "`Framed-User`" and "`Tunnel-Private-Group-Id`". If both exist authentication succeeded. The *chosen_vlan* is either the one returned by the Radius Server or the one set forth in dot-config, according to the `OBEY_DOTCONFIG` parameter.

`RVLAN_CONFIG`

This state calls "`wrs_vlans --port <port> --pvid <pvid>''`", where *pvid* is *noauth_vlan* if authorization failed. We then move to `CONFIGURED` state. The external *wrs_vlans* tool is lazily executed with *system(3)*, and thus this state is blocking. However, *wrs_vlans* completes in no time, so this is acceptable in my opinion.

`RVLAN_CONFIGURED`

The port is quietly running, no action is performed.

`RVLAN_WAIT`

Wait for the child process to terminate (if we killed it in `GODOWN`). This is a transient state that leads to `DOWN`.

4 Multiple Servers

To support multiple Radius servers, *radiusvlan* creates an internal list of possible servers (by splitting the comma-separated list it gets from dot-config).

When a port needs to call *radclient*, it asks for the "best" server. At the beginning, this is the first server listed in the dot-config line.

Then, Whenever *radclient* returns, if it exited in error **and** the reply begins with "`radclient:`", then this is considered a communication error (which is different from an authorization denial). Please note that *radiusvlan* merges *stdout* and *stderr* to the same file descriptor, due to developer laziness.

Communication errors can happen because the server name cannot be resolved by DNS; because the shared secret is wrong, or because the radius server is not running at the selected address.

When such an error happens, the server name is marked as "recently faulty", and another (or the same) server is selected, to resend the same query. The selection process returns the radius server whose failure is oldest, among the list of known servers; this ensures that if two servers are out of order at different times, *radiusvlan* sticks to the one that is currently working, but can get back to the other server when needed.

Thus, in a dynamic environment where ports feature "link up" and "link down" over time, we always query the right server, if any in the list is off-service. However, when several port go up at the same time (e.g. at power on time), we may query the wrong server for all ports, because no failure is yet known to the system when we see the "link up" events.

What follows is an example, in verbose mode, using the string `tornado,gsi.de,192.168.16.201,192.168.16.200` as `CONFIG_RVLAN_RADIUS_SERVERS`. Here, "`tornado`" is a host name but it can't be resolved by DNS (fast error reporting); "`gsi.de`" is not responding (slow error); host 201 does not respond either (slow error); host 200 replies with an authorization error – i.e. it does "*exit 1*", but for a different reason. The log is augmented with timestamps (minutes:seconds), so we see that the timeout of *radtest* is 16 seconds.

```
13:49: Pmask = 0xffffffff
13:49: Radius server: "tornado"
```

```
13:49: Radius server: "gsi.de"
13:49: Radius server: "192.168.16.201"
13:49: Radius server: "192.168.16.200"
13:49: Interface "wri1": not access mode
13:49: Check wri2: down
[...]
13:50: FSM: wri3: justup -> sniff
13:51: recvfrom(wri3): 0800-90e2ba456c6b
13:51: dev wri3 queries server tornado
13:51: FSM: wri3: sniff -> auth
13:52: dev wri3, got 63 bytes so far
13:52: wri3: reaped radclient: 0x00000100
13:52: wri3: server failed
13:52: FSM: wri3: auth -> radclient
13:52: dev wri3 queries server gsi.de
13:52: FSM: wri3: radclient -> auth
14:08: dev wri3, got 55 bytes so far
14:08: wri3: reaped radclient: 0x00000100
14:08: wri3: server failed
14:08: FSM: wri3: auth -> radclient
14:09: dev wri3 queries server 192.168.16.201
14:09: FSM: wri3: radclient -> auth
14:25: dev wri3, got 55 bytes so far
14:25: wri3: reaped radclient: 0x00000100
14:25: wri3: server failed
14:25: FSM: wri3: auth -> radclient
14:25: dev wri3 queries server 192.168.16.200
14:25: FSM: wri3: radclient -> auth
14:27: dev wri3, got 46 bytes so far
14:27: wri3: reaped radclient: 0x00000100
14:27: dev wri3: vlan 4094
14:27: FSM: wri3: auth -> config
14:28: FSM: wri3: config -> configured
```

After the above events, if we plug *wri2*, only the right server is queried:

```
14:34: FSM: wri2: down -> sniff
14:34: recvfrom(wri2): 0026-0008546f9863
14:34: dev wri2 queries server 192.168.16.200
14:34: FSM: wri2: sniff -> auth
14:36: dev wri2, got 46 bytes so far
14:36: wri2: reaped radclient: 0x00000100
14:36: dev wri2: vlan 4094
14:36: FSM: wri2: auth -> config
14:36: FSM: wri2: config -> configured
```

5 Robustness

The tool is designed to be robust. All possible errors are reported back to the caller (and to *stderr*) and no blocking operation is performed. The only exception is the call to *wrs_vlans*, which is blocking.

Any errors in the state machine leaves the port in the same state, so *wrs_vlans* is re-run if it fails, and so on. Failure in reading replies from *radclient* turn the FSM to *GODOWN*, so the procedure is started again – because the port is up.

If the Radius server is not reachable *radclient* will time out, as shown, so the state machine is not stuck.

The startup script uses `CONFIG_WRS_LOG_OTHER` as a destination for its own output, and it is verified to work with my local *rsyslog* server.

The only weak point is in understanding *radclient*'s replies. A sane tool would *exit(1)* or *exit(2)* to mean different things, but *radclient* always does *exit(1)*, so we ar forced to rely on the output strings, which might change from one version to he next.

6 Diagnostic Tools

6.1 Checking the Current Status

You can always see the current configuration by running `rvlan-status`. This example is taken in a running switch,where port *wri1* is in trunk mode (and thus not monitored), and only port *wri17* is connected to a slave, which was authorized:

```
nwt0075m66# /wr/bin/rvlan-status
wri2 (70b3d591e346 <-> ): state down, vlan 0, pid 0, fd -1
wri3 (70b3d591e347 <-> ): state down, vlan 0, pid 0, fd -1
wri4 (70b3d591e348 <-> ): state down, vlan 0, pid 0, fd -1
wri5 (70b3d591e349 <-> ): state down, vlan 0, pid 0, fd -1
wri6 (70b3d591e34a <-> ): state down, vlan 0, pid 0, fd -1
wri7 (70b3d591e34b <-> ): state down, vlan 0, pid 0, fd -1
wri8 (70b3d591e34c <-> ): state down, vlan 0, pid 0, fd -1
wri9 (70b3d591e34d <-> ): state down, vlan 0, pid 0, fd -1
wri10 (70b3d591e34e <-> ): state down, vlan 0, pid 0, fd -1
wri11 (70b3d591e34f <-> ): state down, vlan 0, pid 0, fd -1
wri12 (70b3d591e350 <-> ): state down, vlan 0, pid 0, fd -1
wri13 (70b3d591e351 <-> ): state down, vlan 0, pid 0, fd -1
wri14 (70b3d591e352 <-> ): state down, vlan 0, pid 0, fd -1
wri15 (70b3d591e353 <-> ): state down, vlan 0, pid 0, fd -1
wri16 (70b3d591e354 <-> ): state down, vlan 0, pid 0, fd -1
wri17 (70b3d591e355 <-> 00267b0003d4): state configured, vlan 31, pid 0, fd -1
wri18 (70b3d591e356 <-> ): state down, vlan 0, pid 0, fd -1
```

This works by sending *SIGUSR1* to the running *radiusvlan*, which creates */tmp/rvlan-status* with the above information. If *radiusvlan* is not running, *rvlan-status* will report "`radiusvlan: no process found`".

6.2 Looking at Authorization Strings

Communication with *radclient* happens using *stdin* and *stdout*. Currently *radiusvlan* saves in */tmp* both files, to help tracing any errors. The file names are port-specific, so only the last iteration will be visible.

There are two example: a successful *wri17* authentication and a failed *wri3*authentication:

```
nwt0075m66# grep . /tmp/radclient-wri17-*
/tmp/radclient-wri17-in:User-Name = "00267b0003d4"
/tmp/radclient-wri17-in:User-Password = "00267b0003d4"
/tmp/radclient-wri17-out:Received response ID 93, code 2, length = 50
/tmp/radclient-wri17-out:       Tunnel-Type:0 = 13
/tmp/radclient-wri17-out:       Tunnel-Medium-Type:0 = IEEE-802
/tmp/radclient-wri17-out:       Framed-Protocol = PPP
/tmp/radclient-wri17-out:       Service-Type = Framed-User
/tmp/radclient-wri17-out:       Tunnel-Private-Group-Id:0 = "2984"

wrs# grep . /tmp/radclient-wri17-*
/tmp/radclient-wri3-in:User-Name = "90e2ba456c6b"
/tmp/radclient-wri3-in:User-Password = "90e2ba456c6b"
/tmp/radclient-wri3-out:radclient: no response from server for ID 98 socket 4
```

6.3 Verbose Operation

If you set `RVLAN_VERBOSE` to a non-empty value in the tool's environment, initial enumeration and state machine changes are reported to *stdout*.

This "verbose" mode can also be entered (or left) by seding `SIGUSR2`, see below. This is an example on a running switch where *radiusvlan* was already automatically run:

```
wrs# export RVLAN_VERBOSE=y; killall radiusvlan; /wr/bin/radiusvlan
device wri3 left promiscuous mode
```

```
Pmask = 0xffffffff
Interface "wri1": not access mode
Check wri2: up
Check wri5: down
Check wri6: down
Check wri7: down
Check wri3: up
Check wri4: down
Check wri8: down
[...]
FSM: device wri3 entered promiscuous mode
wri2: justup -> sniff
FSM: wri3: justup -> sniff
vfrom(wri2): 0026-0008546f9863
FSM: wri2: sniff -> auth
recvfrom(wri3): 0800-90e2ba456c6b
FSM: wri3: sniff -> auth
dev wri2, got 55 bytes so far
wri2: reaped radclient: 0x00000100
dev wri2: vlan 4094
FSM: wri2: auth -> config
FSM: wri2: config -> configured
dev wri3, got 54 bytes so far
wri3: reaped radclient: 0x00000100
dev wri3: vlan 4094
FSM: wri3: auth -> config
FSM: wri3: config -> configured
```

In the above example, two interface were up and authorization failed for both (as seen, *radclient* did `exit(1)`). Both interfaces are configured in vlan 4094.

6.4 Forcing Re-Authorizazion

By sending *SIGUSR2* to a running *radiusvlan* all state machines are turned to `JUSTUP` so all authorization is retried, and verbose mode is toggled. Please note that this is pretty raw, and should only be run in a quiet system where all interfaces are *configured* or *down* (the cleanup of state `GODOWN` is not performed).

The simple script `rvlan-debug` can be used to send `SIGUR2` and check the new value of verbosity. This example is in a system where *radiusvlan* was automatically started at boot:

```
wrs#rvlan-debug
radiusvlan verbose level is now 1
```

Diagnostic messages are then sent to syslog, using my `CONFIG_WRS_LOG_OTHER` and related configuration choices. To turn off verbosity, run the command again:

```
wrs#rvlan-debug
radiusvlan verbose level is now 0
```

Let me repeat this trivial diagnostic feature is not meant for production use because it may leave some garbage in the system (e.g. a zombie *radclient* process).

7 Bugs and Missing Features

A few, unfortunately

- It is not expected that MAC addresses change. Both identification of self frames and blessing of peers (for authorization) has an ever-lasting effect. Clearly, if you change client in a port, the link-down and link-up events will force authentication on the new mac address, but if you change the mac address of a PTP slave while it runs, authorization is not re-run.
- Vlan configuration only happens with `--pvid` configuration, and no action is performed on the routing table.

The last item is tricky. The White Rabbit Switch must be informed about vlan-sets, in order to correctly route frames, but those sets sometimes cannot just be derived by the individual *vid* settings.

I was told that for the current application (an "obey-dotconfig" one) I should not touch the routing table, but I'm sure this is not correct for a real multi-vlan setup (especially a dynamic radius-driven environment). This should be investigated when new use cases get real.