Developer's Guide

for PacketFence version 5.0.0

Developer's Guide

by Inverse Inc.

Version 5.0.0 - Mar 2015

Table of Contents

About this Guide

This guide will help you modifying PacketFence to your particular needs. It also contains information on how to add support for new switches.

The latest version of this guide is available at http://www.packetfence.org/documentation/.

Other sources of information

Network Devices Configuration Guide	Covers switch, controllers and access points configuration.
Administration Guide	Covers PacketFence installation, configuration and administration.
`NEWS`	Covers noteworthy features, improvements and bugfixes by release.
`UPGRADE`	Covers compatibility related changes, manual instructions and general notes about upgrading.
`ChangeLog`	Covers all changes to the source code.

These files are included in the package and release tarballs.

Documentation

The in-depth or more technical documentation is always as close to the code as possible. Always look at the `POD` doc [1]. To do so, the prefered way is using the `perldoc` command as follows:

```
perldoc conf/authentication/ldap.pm
```

[1] Perl's Plain Old Documentation: http://perldoc.perl.org/perlpod.html

Code conventions

Code style

Caution

Work in progress.

We are slowly migrating away from an automated `perltidy` code style. The reason we are not doing another pass of tidy is that it messes up code history and makes maintainer's job more complicated than it should be. Every new change uses the new guidelines so over time the old code style will slowly disappear.

- Lines of 120 character width maximum (lower encouraged)
- No tab characters
- Stay consistent with surrounding white spaces
- Document each subroutine in POD format (`perldoc perlpod`)
- Use constants instead of hardcoded strings or numbers (use `constant` or `Readonly` modules)
- in object-oriented modules we use CamelCase [1] notation (ex: `$radiusRequest->getVoIpAttributes();`)
- in procedural modules we use perl's usual notation (ex: `$node_info{'pid'} = $current_request{'pid'};`)
- regular expressions should be documented (with the `/x` modifier)

```
if ($phone_number =~ /
    ^\(?([2-9]\d{2})\)?  # captures first 3 digits allows parens
    (?:-|.|\s)?          # separator -, ., space or nothing
    (\d{3})              # captures 3 digits
    (?:-|.|\s)?          # separator -, ., space or nothing
    (\d{4})$             # captures last 4 digits
    /x) {
        return "$1$2$3";
}
```

- `SQL` should be capitalized, properly indented and always use named fields (no *)

```
$node_statements->{'node_add_sql'} = get_db_handle()->prepare(<<'SQL');
    INSERT INTO node (
        mac, pid, category_id, status, voip, bypass_vlan,
        detect_date, regdate, unregdate, lastskip,
        user_agent, computername, dhcp_fingerprint,
        last_arp, last_dhcp,
        notes,
    ) VALUES (
        ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?
    )
SQL
```

Customizing PacketFence

Captive Portal

Presentation

XHTML Templates

Captive portal content use Template Toolkit templates. All the template files are located in `/usr/local/pf/html/captive-portal/templates`. You can freely edit the `HTML` code in these files. However, if you want to customize the pages beyond the `HTML` template (for example by adding new variables to it), you'll need to look into the `/usr/local/pf/lib/pf/web/custom.pm` Perl module. This module allows you to overload the behavior of the default `/usr/local/pf/lib/pf/web.pm` module.

Each template relies on `header.html` and `footer.html` for the common top and bottom portion of each file.

CSS

Most of the branding should be possible by only changing the `CSS`. Here are the various `CSS` files used by PacketFence:

Usual one	`/usr/local/pf/html/captive-portal/content/styles.css`
Mobile	`/usr/local/pf/html/captive-portal/content/responsive.css`
Print	`/usr/local/pf/html/captive-portal/content/print.css`

Workflow

When a `HTTP` request is received by the Apache web server, the following workflow happens:

1. URL is compared against the redirection instructions in `/usr/local/pf/conf/httpd.conf.d/captive-portal-cleanurls.conf`
2. Requested CGI script in `/usr/local/pf/html/captive-portal/` is executed
3. CGI script calls a `generate_<type>` which is defined in `/usr/local/pf/lib/pf/web.pm`
4. The `generate_<type>` function populate the proper template in `/usr/local/pf/html/captive-portal/templates` in order to render the page

Remediation Pages

The remediation page shown to the user during isolation are specified through the URL parameter of the given violation in `/usr/local/pf/conf/violations.conf`. In its default configuration, PacketFence uses Template Toolkit to render text provided in the directory `/usr/local/pf/html/captive-portal/templates/violations` and obeys to everything mentionned in the Presentation section.

Translations

The language of the user registration pages is selected through the `general.locale` configuration parameter. Translatable strings are handled differently for the Remediation pages and the rest of the captive portal:

- Remediation pages

  Strings defined in the violation pages (in `/usr/local/pf/html/captive-portal/templates/violations`) will be looked up in the translation files in `/usr/local/pf/conf/locale/..` and if a translation is available the translated string will be the one visible on the captive portal.

  Also, if you create a violation template with the name of your locale in `/usr/local/pf/html/captive-portal/templates/violations` in the format: `<template_name>.<locale_name>.html`. It will be loaded instead of the default `<template_name>.html` and so you can put strings and HTML directly in your target language without the hassle of escaping everything properly as you would need to do with `gettext`.

  For example, if `malware.es_ES.html` exists and you are using the `es_ES` (Spanish) locale then it will be loaded instead of `malware.html` on a violation set to load the `malware` template.

- Rest of the captive portal

  In the templates, if a string is in a `i18n()` call it will be translated. Also `pf::web` takes care of performing some of the other translations.

Adding custom fields to the database

You can, if needed, add additional fields to the PacketFence database. Keep in mind though that this might lead to more work when you upgrade to the next PacketFence version. Depending on the degree of integration of these fields with PacketFence, you'll have to execute one or more of the following steps

Adding a field to the database only

In this case, the field is part of one of the main PacketFence tables, but PacketFence is unaware of it. PacketFence won't consult the field and won't be able to modify it. A possible usage scenario would be a 3rd party application which maintains this field.

Since PacketFence doesn't have to know about the field, all you have to do is execute your `SQL ALTER TABLE` query and you are done.

Adding a field and giving PacketFence read-only access

In this case, PacketFence can show the contents of the table using both `pfcmd` but won't be able to modify the contents of the field.

Start by modifying the database table using an `SQL ALTER TABLE` query.

Then, modify the Perl module having the same name as the table you have added the field to, i.e. If you added the field to the node table, then edit `/usr/local/pf/lib/pf/node.pm`. You'll have to modify the `SQL SELECT` queries at the beginning of the file to include your new field and, possibly the functions using these queries. If your new field should be used in reports, the dashboard or graphs, you'll also have to modify the queries in `/usr/local/pf/lib/pf/pfcmd/graph.pm`, `/usr/local/pf/lib/pf/pfcmd/report.pm` and `/usr/local/pf/lib/pf/pfcmd/dashboard.pm`.

Adding a field and giving PacketFence read-write access

Start by creating the read-only field as described above.

Then, modify the 'SQL UPDATE` and `INSERT` queries in the database tables' Perl module, as well as the associated functions.

The last step is to make PacketFence's grammar aware of the new field. Modify `/usr/local/pf/lib/pf/pfcmd/pfcmd.pm` and then re-generate the precompiled grammar (which is used by the `pfcmd` CLI) with:

```
cd /usr/local/pf
/usr/bin/perl -w -e '
  use strict; use warnings;
  use Parse::RecDescent; use lib "/usr/local/pf/lib";
  use pf::pfcmd::pfcmd;
  Parse::RecDescent->Precompile($grammar, "pfcmd_pregrammar");
'
mv pfcmd_pregrammar.pm /usr/local/pf/lib/pf/pfcmd/pfcmd_pregrammar.pm
```

VLAN assignment

pfsetvlan uses the `getNormalVlan` function defined in `pf::vlan::custom` to determine a node's VLAN. Here's the default function:

```
sub getNormalVlan {
    #$switch is the switch object (pf::Switch)
    #$ifIndex is the ifIndex of the computer connected to
    #$mac is the mac connected
    #$node_info is the node info hashref (result of pf::node's node_view on $mac)
    #$conn_type is set to the connnection type expressed as the constant in
 pf::config
    #$user_name is set to the RADIUS User-Name attribute (802.1X Username or MAC
 address under MAC Authentication)
    #$ssid is the name of the SSID (Be careful: will be empty string if radius
 non-wireless and undef if not radius)
    my ($this, $switch, $ifIndex, $mac, $node_info, $connection_type, $user_name,
 $ssid) = @_;

    my $logger = Log::Log4perl->get_logger();

    return $switch->getVlanByName('normalVlan');
}
```

As you can see, the function receives several parameters (such as the switch and full node details) which allow you to return the VLAN in a way that matches exactly your needs!

SNMP

Introduction

Good places to start reading about SNMP are http://en.wikipedia.org/wiki/SNMP and http://www.net-snmp.org/.

When working with SNMP, you'll sooner or later (in fact more sooner than later) be confronted with having to translate between OIDs and variable names. When the OIDs are part of the Cisco MIBs, you can use the following tool to do the translation: http://tools.cisco.com/Support/SNMP/public.jsp. Otherwise, you'll have to use `snmptranslate` for example and setup your own collection of MIBs, provided (hopefully) by the manufacturer of your network equipment.

Obtaining switch and port information

Below are some example of how to obtain simple switch and port information using SNMP. We'll assume that your switch understands SNMP v2, has the read community `public` defined and is reachable at `192.168.1.10`.

Switch Type

```
snmpwalk -v 2c -c public 192.168.1.10 sysDescr
```

Switchport indexes and descriptions

```
snmpwalk -v 2c -c public 192.168.1.10 ifDescr
```

Switchport types

```
snmpwalk -v 2c -c public 192.168.1.10 ifType
```

Switchport status

```
snmpwalk -v 2c -c public 192.168.1.10 ifAdminStatus
snmpwalk -v 2c -c public 192.168.1.10 ifOperStatus
```

Supporting new network hardware

PacketFence is designed to ease the addition of support for new network hardware referred to as Network Devices. All supported network devices are represented through Perl objects with an extensive use of inheritance. Adding support for a new product comes down to extending the `pf::Switch` class (in `/usr/local/pf/lib/pf`).

The starting point to adding support for a new network device should be the vendor's documentation! First of all, you'll have to figure out the exact capabilities of the switch and how these capabilities will fit into PacketFence. Is it a Switch, an Access-Point or a Wireless Controller?

Switch

Will you be able to use only link change traps? Does your switch allow you to use MAC notification traps? Port Security? MAC Authentication? 802.1X?

Link change capabilities

You need to define a new class which inherits from `pf::Switch` and defines at least the following functions:

- `getMacAddrVlan`
- `getVersion`
- `getVlan`
- `getVlans`
- `isDefinedVlan`
- `parseTrap`
- `_getMacAtIfIndex`
- `_setVlan`

The 'parseTrap` function will need to return an hash with keys `trapType` and `trapIfIndex`. The associated values must be `up` or `down` for `trapType` and the traps' `ifIndex` for `trapIfIndex`. See a similar switch's implementation for inspiration. Usually recent modules are better coded than older ones.

MAC notification capabilities

In addition to the functions mentioned for link change, you need to define the following function:

- `isLearntTrapsEnabled`

Also, your `parseTrap` function will need to return `trapOperation`, `trapVlan` and `trapMac` keys in addition to `trapType` equals `mac`. See a similar switch's implementation for inspiration. Usually recent modules are better coded than older ones.

Port security capabilities

In addition to the functions mentioned for link change, you need to define the following functions:

- `isPortSecurityEnabled`
- `authorizeMAC`

In this case, the `parseTrap` function needs to return `secureMacAddrViolation` for the `trapType` key. See a similar switch's implementation for inspiration. Usually recent modules are better coded than older ones.

MAC Authentication

Note

Work in progress

NAS-Port translation

Often the `ifIndex` provided by the switch in a RADIUS `Access-Request` is not the same as it's real world physical equivalent. For example in Cisco requests are in the 50xxx while physical ifIndex are 10xxx. In order for PacketFence to properly shut the port or request re-authentication a translation between the two is required. To do so provide an implementation of the following interface:

- `NasPortToIfIndex`

MAC Authentication re-evaluation

MAC Authentication re-evaluation is necessary in order to provoke a VLAN change in the PacketFence system. This happens for instance when a node is isolated based on an IDS event or when the user succesfully authenticates through the captive portal. The default implementation in `pf::Switch` will bounce the port if there is no Voice over IP (VoIP) devices connected to the port. Otherwise it will do nothing and send an email. If your device has specific needs (for example it doesn't support RADIUS Dynamic VLAN Assignments) override:

- `handleReAssignVlanTrapForWiredMacAuth`

Please note that the default implementation works 99% of the time. If you are unsure whether to override, it means you don't need to override.

Once the MAC Authentication works, add the Wired MAC Auth capability to the switch's code with:

```
sub supportsWiredMacAuth { return $TRUE; }
```

802.1X

Note

Work in progress

NAS-Port translation

Often the `ifIndex` provided by the switch in a RADIUS `Access-Request` is not the same as it's real world physical equivalent. For example in Cisco requests are in the 50xxx while physical ifIndex are 10xxx. In order for PacketFence to properly shut the port or request re-authentication a translation between the two is required. To do so provide an implementation of the following interface:

- `NasPortToIfIndex`

So far the implementation has been the same for MAC Authentication and 802.1X.

Force 802.1X re-authentication

802.1X re-authentication is necessary in order to provoke a VLAN change in the PacketFence system. This happens for instance when a node is isolated based on an IDS event or when the user succesfully authenticates through the captive portal. The default implementation in `pf::Switch` uses SNMP and the standard `IEEE8021-PAE-MIB` and is generally well supported. If the default implementation to force 802.1X re-authentication doesn't work override:

- `dot1xPortReauthenticate`

Proper 802.1X implementations will perform re-authentication while still allowing traffic to go through for supplicants under re-evaluation.

Once the `802.1X` works, add the Wired Dot1X capability to the switch's code with:

```
sub supportsWiredDot1x { return $TRUE; }
```

RADIUS Dynamic Authorization (RFC3576)

Note

RADIUS Dynamic Authorization implementation is not recommended on the wired side at this point.

RADIUS Dynamic Authorization also known as RADIUS Change of Authorization (CoA) or RADIUS Disconnect Messages is supported by PacketFence starting with version 3.1.

On wired network devices CoA can be used to change the security posture of a MAC and perform other functions like bounce a port. So far we only encountered support for CoA on the wired side on the Cisco hardware. For an implementation example check `_radiusBounceMac` in `pf::Switch::Cisco`.

Floating Network Devices Support

Floating Network Devices are described in the Administration Guide under "Floating Network Devices" in the "Optional Components" section. Refer to this documentation if you don't know what Floating Network Devices are.

In order to support Floating Network Devices on a switch, you need to implement the following methods:

- `setPortSecurityEnableByIfIndex($ifIndex, $enable)`
- `isTrunkPort($ifIndex)`
- `setModeTrunk($ifIndex, $enable)`
- `setTaggedVlans($ifIndex, $switch_locker_ref, @vlans)`
- `removeAllTaggedVlans($ifIndex, $switch_locker_ref)`

You might need to implement the following:

- `enablePortConfigAsTrunk($mac, $switch_port, $switch_locker, $taggedVlans)`

  Provided by `pf::Switch` core as the glue between `setModeTrunk()`, `setTaggedVlans()` and `removeAllTaggedVlans()`. Override if necessary.

- `disablePortConfigAsTrunk($switch_port)`

  Provided by `pf::Switch` core as the glue between `setModeTrunk()`, `setTaggedVlans()` and `removeAllTaggedVlans()`. Override if necessary.

- `enablePortSecurityByIfIndex($ifIndex)`

  Provided by `pf::Switch` core as a slim accessor to `setPortSecurityEnableByIfIndex()`. Override if necessary.

- `disablePortSecurityByIfIndex($ifIndex)`

  Provided by `pf::Switch` core as a slim accessor to `setPortSecurityEnableByIfIndex()`. Override if necessary.

- `enableIfLinkUpDownTraps($ifIndex)`

  Provided by `pf::Switch` core as a slim accessor to `setIfLinkUpDownTrapEnable`. Override if necessary.

- `disableIfLinkUpDownTraps($ifIndex)`

  Provided by `pf::Switch` core as a slim accessor to `setIfLinkUpDownTrapEnable`. Override if necessary.

Once all the required methods are implemented, enable the capability in the switch's code with:

```
sub supportsFloatingDevice { return $TRUE; }
```

Wireless Access-Points or Controllers

Minimum hardware requirements

PacketFence's minimum requirements regarding Wireless hardware is:

- definition of several SSID with several VLANs inside every SSID (minimum of 2 VLANs per SSID)
- RADIUS authentication (MAC Authentication / 802.1X)
- Dynamic VLAN assignment through RADIUS attributes
- a means to de-associate or de-authenticate a client through CLI (Telnet or SSH), SNMP, RADIUS Dyn-Auth [1] or WebServices

Most of these features are available on enterprise grade Access Points (AP) or Controllers. Where the situation starts to vary wildly is for deauthentication support.

De-authentication techniques

CLI (SSH or Telnet)

An error prone interface and requires preparation for the SSH access or is insecure for Telnet. Not recommended if you can avoid it.

SNMP

SNMP de-authentication works well when available. However Vendor support is not consistent and the OID to use are not standard.

RADIUS Dynamic Authorization (RFC3576)

RADIUS Dynamic Authorization also known as RADIUS Change of Authorization (CoA) or RADIUS Disconnect Messages is supported by PacketFence starting with version 3.1. When supported it is the preferred technique to perform de-authentication. It is standard and requires less configuration from the user.

An actual implementation can be found in `pf::Switch::Aruba`.

Template module

Start with a copy of the template module `pf/lib/pf/Switch/WirelessModuleTemplate.pm` and fill in appropriate documentation and code.

Required methods

You need to implement at least:

`getVersion()`	Fetches firmware version

`parseTrap()`	Parses the SNMP Traps sent by the hardware. For wireless hardware an empty method like the one in `pf::Switch::WirelessModuleTemplate` is ok.
`deauthenticateMac()`	Performs deauthentication
`supportsWirelessMacAuth()`	Return `$TRUE` if MAC-Authentication is supported
`supportsWirelessDot1x()`	Return `$TRUE` if 802.1X (aka WPA-Enterprise) is supported

Override methods

If default implementation of the following methods doesn't work you will need to override them:

`extractSsid()`	Extract SSID from RADIUS Request

Special case: bridged versus tunneled modes and deauthentication

It is important to validate the Access-Point (AP) to Controller relationship when operating in bridged mode versus when operating in tunneled mode. For example, some hardware will send the RADIUS `Access-Request` from the AP when in bridged mode even though it is controlled by a controller. This behavior impacts deauthentication because it still needs to be performed on the controller. To support this behavior a `switches.conf` parameter was introduced: `controller_ip`.

When adding a new Wireless module try to validate the bridged versus tunneled behavior and modify `deauthenticateMac()` to honor `controller_ip` if required.

The "adding a new network device module in PacketFence" checklist

Here's a quick rundown of the several files you need to edit in order to add a new switch into PacketFence. There's a plan to reduce this amount of work in progress see issue #1085.

- Tested model and firmware version should be documented in module's POD
- Any bugs and limitations should be documented in module's POD
- Add it to `pf/html/admin/configuration/switches_add.php` and `switches_edit.php`
- Make sure that all tests pass
- Add configuration documentation to the Network Devices Guide
- Add switch to the Network Devices Guide's switch chart
- Add switch to the chart in `README.network-devices`

Developer recipes

Running development version

Bleeding edge

For day to day development one can run a checkout of the current development branch in `/usr/local/pf/` and develop there within a working setup.

Care should be taken not to commit local configuration files changes and files not in the repository.

Not so bleeding edge

Using the development `yum` repository and upgrade packetfence often is a good way to proceed. Check our snapshots download page for instructions.

Make sure you read the `UPGRADE` document after every upgrades to avoid any surprises.

Debugging PacketFence grammar

PacketFence uses a parser to validate user input. This parser is referred to as the grammar. When you see errors like

```
Command not understood. (pfcmd grammar test failed at line 217.)
```

it means that you faced a problem in the command you are trying to send or in the grammar itself.

The parsing of a command is a tricky process. First the command is interpreted in the `pf::pfcmd` module using traditional regular expressions. Then some of the commands will trigger the parser `pf::pfcmd::pfcmd_pregrammar` which is a precompiled module that is generated from `pf::pfcmd::pfcmd` when packetfence is built.

To help troubleshoot a failing command, you can enable tracing on the parser by removing the comment from the following line in `pfcmd`: `#our $RD_TRACE = 1;`

New Exception handling techniques under testing

Little attention was given to error-handling in PacketFence's early design. This is understandable as it wasn't probably the most bang-for-the-buck thing to do. However we must now live with a large code base that explodes at runtime or that doesn't differentiate an erroneous condition from an undefined or 0 value. Refactoring to improve error-handling will be gradual but new code should follow these tips:

1. use `Try::Tiny`
2. wrap stuff in `try {...} catch {...};` (and optionally a `finally {...};`)
3. in the code use `die(...);` to throw an exception and make the error message meaningful
4. in the catch block, use `$logger->logcarp("explanation: $_")` if I want output to the CLI, otherwise, choose wisely

This catches a lot of errors (including runtime crashers) and allows us to recover from these conditions.

So far, it is mandatory to wrap the Web Services enabled network devices modules' code since `SOAP::Lite` will die on you if host is unreachable for example (actually it's `LWP::UserAgent` who will).

Contributing

Here are some golden rules of contributing to PacketFence:

- Be active on the developer mailing list

The place to be if you want to contribute to the PacketFence project is our developers mailing list: https://lists.sourceforge.net/lists/listinfo/packetfence-devel. Let us know your issues, what you are working on and how you want to solve your problems. The more you collaborate the greater the chances that your work will be incorporated in a timely fashion.

- Use the issue tracker: http://www.packetfence.org/bugs/

Good chances that the bug you want to fix or the feature you want to implement is already filed and that information in the ticket will help you.

- Please provide small, focused and manageable patches or pull-requests

If you plan on doing a lot of code, use `git` and track our current stable branch called `stable`. Develop the feature in small chunks and stay in touch with us. This way it'll be merged quickly in our codebase. Ideally there would be no big code dumps after finishing a feature.

Creating patches

Note

Since we migrated to git / github, using these tools is recommended over sending patches by hand.

Patches should be sent in unified diff format. This can be obtained from the `diff` or `git` tools.

```
diff -u oldfile newfile
```

or from a checkout of the PacketFence source code from `git`:

```
git diff
```

Translations

The internationalization process uses `gettext`. If you are new to `gettext`, please consult http://www.gnu.org/software/gettext/manual/gettext.html#Overview for a quick introduction.

The PO files are stored in `/usr/local/pf/conf/locale`. List that directory to see the languages we currently have translations for.

Online using Transifex

We use the hosted service Transifex to translate PacketFence's PO files. It offers the possibility to translate all the strings online as well as providing a command-line tool to push your changes. It's very convenient.

To use Transifex, you must first sign up for a free account here: https://www.transifex.net/plans/signup/free/

- Once registered, request a new team for your language
- Once authorized, you'll be able to start/continue translating PacketFence in your language

If you need further help about using Transifex, you might want to have a look here.

Using traditional method

If you want to add support for a new language, please follow these steps:

1. create a new language subdirectory in `/usr/local/pf/conf/locale`
2. change into your newly created directory
3. create a new subdirectory `LC_MESSAGES`
4. change into your newly created directory
5. copy the file `/usr/local/pf/conf/locale/en/LC_MESSAGES/packetfence.po` into your directory
6. translate the message strings in `packetfence.po`
7. create the MO file by executing:

```
/usr/bin/msgfmt packetfence.po
```

Submit your new translation to the PacketFence project by contacting us at packetfence-devel@lists.sourceforge.net.

Additional Information

For more information, please consult the mailing archives or post your questions to it. For details, see:

- packetfence-announce@lists.sourceforge.net: Public announcements (new releases, security warnings etc.) regarding PacketFence
- packetfence-devel@lists.sourceforge.net: Discussion of PacketFence development
- packetfence-users@lists.sourceforge.net: User and usage discussions

Commercial Support and Contact Information

For any questions or comments, do not hesitate to contact us by writing an email to: support@inverse.ca.

Inverse (http://inverse.ca) offers professional services around PacketFence to help organizations deploy the solution, customize, migrate versions or from another system, performance tuning or aligning with best practices.

Hourly rates or support packages are offered to best suit your needs.

Please visit http://inverse.ca/ for details.

GNU Free Documentation License

Please refer to http://www.gnu.org/licenses/fdl-1.2.txt for the full license.