Guide to API Privilege Escalation - The Admin Module Method

Overview

In cPanel & WHM version 86 and later, cPanel, L.L.C. recommends that you develop Perl admin modules as subclasses of the `Cpanel::Admin::Base` class. The admin module method improves creating API escalation methods in several ways. This method is:

- **Easier.** You do not need a separate configuration file for this method. All you need is an ordinary Perl module.
- **Faster.** The `cpsrvd` service loads admin modules directly and within the initiated process with the admin module method. The previous Call method would run a new command on each admin function call. This resulted in a slower process.
- **More powerful.** Admin modules can receive a file handle. This allows for streaming between the privileged and unprivileged processes.
- **More secure.** If a thrown exception escapes the admin function, the caller receives a generic error rather than details about the uncaught exception.

We recommend that you write **all** admin functions in Perl. If you cannot, use the Standard method to write admin functions in other code languages.

Create a new module

To create a new admin module, perform the following steps:

1. Create a new namespace and module in the `/var/cpanel/perl/Cpanel/Admin/Modules` directory. For example, if you create the `GreatHosting` namespace, then create the `GreatThings` module, that module will exist in the `/var/cpanel/perl/Cpanel/Admin/Modules/GreatHosting/GreatThings.pm` file path.
2. Make the new module a subclass of the `Cpanel::Admin::Base` class.
3. Create a function named `_actions()` in the new module.

> ⚠ **Note:**
>
> - This function **must** return a list of names of functions that your module exposes to callers.
> - This module's callers will **not** see any function **not** included in the `_actions()` function list.

4. Create your functions in the module:
   - By convention, names of such functions are in all uppercase (ALLCAPS) characters.
   - Add each function's name to the `_actions()` function's return.
   - Each function receives a context object and a list of parameters. To pass one or more values back to the caller, return those values as your function's return.

Example

The following is an example of the admin module's basic usage:

```
package Cpanel::Admin::Modules::GreatHosting::GreatThings;

use strict;
use warnings;

use parent 'Cpanel::Admin::Base';

use constant _actions => (
    'SAY_HI',
    'GET_INFO',
);

# This function simply returns the string “hello”.
sub SAY_HI {
    return 'hello';
}

# This function returns a list of values. The first-returned value is an array reference that
# contains all of the values that the caller sent.
sub GET_INFO {
    my ($self, @args) = @_;

    return (
        \@args,
        $self->get_cpuser_domains(),
        $self->get_caller_username(),
    );
}

1;
```

How to call admin modules

To call your admin module, use the following example:

```
use Cpanel::AdminBin::Call;

# $result will be “hello”:
my $result = Cpanel::AdminBin::Call::call('GreatHosting', 'GreatThings', 'SAY_HI');

# @results will contain the 3 values that GET_INFO() returns.
# $results[0] will be ['foo', 'bar'].
my @results = Cpanel::AdminBin::Call::call('GreatHosting', 'GreatThings', 'GET_INFO', 'foo', 'bar');
```


Note:

The called admin function runs in the same context (`void`, `scalar`, or `list`) as the `Cpanel:AdminBin::Call:call()` function. For example, if the unprivileged code calls the `GET_INFO` function in scalar context, the caller receives the **last** element of the list.

Exception handling

If an untrapped exception happens within an admin function, an exception gets thrown in the calling process. This exception will contain an error ID, and the error details and the same error ID get written to the `/usr/local/cpanel/logs/error_log` file. You can use this log file to review specific errors via the error ID.

To indicate details of an error (for example, validation failures) to a caller, return the details as status codes.

Streaming

Admin modules let you send a file handle as part of a call to an admin function. This allows you to reduce the overhead of the exchange of large amounts of data between privileged and unprivileged processes. An admin function that accepts a filehandle might look like the following example:

```
sub TAKES_FILEHANDLE {
    my ($self, @args) = @_;

    my $fh = $self->get_passed_fh();

    # ... now do whatever is needed with $fh
}
```

A call to this function might look like the following example:

```
use Cpanel::AdminBin::Call;

Cpanel::AdminBin::Call::stream( $filehandle, 'GreatHosting', 'GreatThings', 'TAKES_FILEHANDLE' );
```

The admin function can also pass a filehandle back to the caller. To do this, perform the following steps:

1. Have the caller create a pair of UNIX-domain sockets via a `socketpair` call.
2. Have the caller pass one of those sockets to the admin function.
3. Have the admin function pass its intended filehandle back to the caller via the socket it received from the caller. You can use CPAN modules such as IO::FDPass to simplify this.

Note:

Because the `stream()` function uses blocking I/O, real-time communication between the unprivileged and privileged processes is not currently possible.

Security concerns

- It is possible for an unprivileged caller to call an admin function directly. Because of this, admin functions **must** fully validate all inputs. The caller **should** ordinarily perform that same validation itself before calling the admin function.
- Ordinary inputs to and outputs from an admin function **must** use **only** the following values:
  - `undef`
  - Strings.
  - Numbers.
  - Array references.
  - Hash references.
- There is no centralized method to handle Webmail username validation. If you call an admin function on behalf of a Webmail user, that function **must** accept the Webmail username as an argument.

Public methods of the `Cpanel::Admin::Base` class

Warning:

Any `Cpanel::Admin::Base` methods not documented below are subject to removal or change at any time **without** prior notice. Do **not** use any undocumented methods in your code.

cPanel, L.L.C. documents the following methods for use by integrators when writing admin functions:

- `get_caller_username()` — Returns the name of the cPanel user who called the admin function.
- `get_cpuser_domains()` — Returns an array reference of the calling cPanel user's domains, including the cPanel user's primary domain.
- `get_passed_fh()` — Returns the Perl file handle that the caller passed. If the caller did not pass a file handle, this returns the `undef` value.

Access control and cPanel demo mode users

By default, cPanel demo mode users cannot access **any** admin functions. To expose an admin function to demo mode users, create a `_demo_actions()` function that returns the names of the functions you want to make available to these users.

Note:

You do **not** need to create the `_demo_actions()` method for any other reason.