How To Use SuExec in Apache to run CGI Scripts on an Ubuntu

Authored by: ***ASPHostServer Administrator*** *[asphostserver@gmail.com]*
Saved From: *http://faq.asphosthelpdesk.com/article.php?id=233*

Introduction

The Apache web server is the most popular web server in the world. It can be used to deliver static and dynamic web content to visitors in a multitude of different contexts.

One of the most common ways of generating dynamic content is through the use of `CGI`, or the common gateway interface. This provides a standard way of executing scripts that generate web content that can written in a variety of programming languages.

Running any kind of executable code within a web-space comes with a certain amount of risk. In this guide, we will demonstrate how to implement CGI scripting with the `suexec` module, which allows you to run scripts in a way that doesn't elevate privileges unnecessarily.

Prerequisites

In this guide, we will be configuring an Ubuntu 12.04 server with a standard LAMP (Linux, Apache, MySQL, PHP) installation. We assume that you have already installed these basic components and have them working in a basic configuration.

We will be referencing the software as it is in its initial state following that tutorial.

How To Enable CGI Scripts

In Ubuntu's Apache configuration, CGI scripts are actually already configured within a specific CGI directory. This directory is empty by default.

CGI scripts can be any program that has the ability to output HTML or other objects or formats that a web browser can render.

If we go to the Apache configuration directory, and look at the modules that Apache has enabled in the `mods-enabled` directory, we will find a file that enables this functionality:

```
less /etc/apache2/mods-enabled/cgi.load
```

```
LoadModule cgi_module /usr/lib/apache2/modules/mod_cgi.so
```

This file contains the directive that enables the CGI module. This allows us to use this functionality in our configurations.

Although the module is loaded, it does not actually serve any script content on its own. It must be enabled within a specific web environment explicitly.

We will look at the default Apache virtual host file to see how it does this:

```
sudo nano /etc/apache2/sites-enabled/000-default
```

While we are in here, let's set the server name to reference our domain name or IP address:

```
<VirutalHost *:80> ServerName your_domain_or_IP_address ServerAdmin
your_email_address. . .
```

We can see a bit down in the file the part that is applicable to CGI scripts:

```
ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/ <Directory "/usr/lib/cgi-bin">
AllowOverride None Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch Order
allow,deny Allow from all </Directory>
```

Let's break down what this portion of the configuration is doing.

The `ScriptAlias` directive gives Apache permission to execute the scripts contained in a specific directory. In this case, the directory is `/usr/lib/cgi-bin/`. While the second argument gives the file path to the script directory, the first argument, `/cgi-bin/`, provides the URL path.

This means that a script called "script.pl" located in the `/usr/lib/cgi-bin` directory would be executed when you access:

```
your_domain.com/cgi-bin/script.pl
```

Its output would be returned to the web browser to render a page.

The `Directory` container contains rules that apply to the `/usr/lib/cgi-bin` directory. You will notice an option that mentions CGI:

```
Options +ExecCGI ...
```

This option is actually unnecessary since we are setting up options for a directory that has already been declared a CGI directory by `ScriptAlias`. It does not hurt though, so you can keep it as it is.

If you wished to put CGI files in a directory outside of the ScriptAlias, you will have to add these two options to the directory section:

```
Options +ExecCGI AddHandler cgi-script .pl .rb [extensions to be treated as CGI
scripts]
```

When you are done examining the file, save and close it. If you made any changes, restart the web server:

```
sudo service apache2 restart
```

Make a Test CGI Script

We will create a basic, trivial CGI script to show the steps necessary to get a script to execute correctly.

As we saw in the last section, the directory designated in our configuration for CGI scripts is `/usr/lib/cgi-bin`. This directory is not writeable by non-root users, so we will have to use sudo:

```
sudo nano /usr/lib/cgi-bin/test.pl
```

We gave the file a ".pl" extension because this will be a Perl script, but Apache will attempt to run any file within this directory and will pass it to the appropriate program based on its first line.

We will specify that the script should be interpreted by Perl by starting the script with:

```
#!/usr/bin/perl
```

Following this, the first thing that the script must output is the content-type that will be generated. This is necessary so that the web browser knows how to display the output it is given. We will print out the HTML content type, which is "text/html", using Perl's regular print function.

```
print "Content-type: text/html\n\n";
```

After this, we can do whatever functions or calculations are necessary to produce the text that we want on our website. In our example, we will not produce anything that wouldn't be easier as just plain HTML, but you can see that this allows for dynamic content if your script was more complex.

The previous two components and our actual HTML content combine to make the following script:

```
#!/usr/bin/perl print "Content-type: text/html\n\n"; print
"<html><head><title>Hello There...</title></head>"; print "<body>"; print
"<h1>Hello, World.</h1><hr>"; print "<p>This is some regular text.</p>"; print
"<p>The possibilities are great.</p>"; print "</body></html>";
```

Save and close the file.

Now, we have a file, but it isn't marked as executable. Let's change that:

```
sudo chmod 755 /usr/lib/cgi-bin/test.pl
```

Now, if we navigate to our domain name, followed by the CGI directory (/cgi-bin/), followed by our script name (test.pl), we should see the output of our script.

Point your browser to:

```
your_domain.com/cgi-bin/test.pl
```

You should see something that looks like this:

Not very exciting, but rendered correctly.

If we choose to view the source of the page, we will see only the arguments given to the print functions, minus the content-type header:

```
<html><head><title>Hello There...</title></head>
<body><h1>Hello, World.</h1><hr><p>This is some
regular text.</p><p>The possibilities are great.</p>
</body></html>
```

How To Enable SuExec

There are some security concerns implicit in setting a script as executable by anybody. Ideally, a script should only be able to be executed by a single, locked down user. We can set up this situation by using the `suexec` module.

We will actually install a modified suexec module that allows us to configure the directories in which it operates. Normally, this would not be configurable without recompiling from source.

Install the alternate module with this command:

```
sudo apt-get install apache2-suexec-custom
```

Now, we can enable the module by typing:

```
sudo a2enmod suexec
```

Next, we will create a new user that will own our script files. If we have multiple sites being served, each can have their own user and group:

```
sudo adduser script_user
```

Feel free to enter through all of the prompts (including the password prompt). This user does not need to be fleshed out.

Next, let's create a scripts directory within this new user's home directory:

```
sudo mkdir /home/script_user/scripts
```

Suexec requires very strict control over who can write to the directory. Let's transfer ownership to the script_user user and change the permissions so that no one else can write to the directory:

```
sudo chown script_user:script_user /home/script_user/scripts sudo chmod 755
/home/script_user/scripts
```

Next, let's create a script file and copy and paste our script from above into it:

```
sudo -u script_user nano /home/script_user/scripts/attempt.pl

#!/usr/bin/perl print "Content-type: text/html\n\n"; print
"<html><head><title>Hello There...</title></head>"; print "<body>"; print
"<h1>Hello, World.</h1><hr>"; print "<p>This is some regular text.</p>"; print
```

```
"<p>The possibilities are great.</p>"; print "</body></html>";
```

Make it executable next. We will only let our script_user have any permissions on the file. This is what the suexec module allows us to do:

```
sudo chmod 700 /home/script_user/scripts/attempt.pl
```

Next, we will edit our Apache virtual host configuration to allow scripts to be executed by our new user.

Open the default virtual host file:

```
sudo nano /etc/apache2/sites-enabled/000-default
```

First, let's make our CGI directory. Instead of using the `ScriptAlias` directive, as we did above, let's demonstrate how to use the regular `Alias` directory combined with the `ExecCGI` option and the `SetHandler` directive.

Add this section:

```
Alias /scripts/ /home/script_user/scripts/ <Directory
"/home/script_user/scripts"> Options +ExecCGI SetHandler cgi-script </Directory>
```

This allows us to access our CGI scripts by going to the "/scripts" sub-directory. To enable the suexec capabilities, add this line outside of the "Directory" section, but within the "VirtualHost" section:

```
SuexecUserGroup script_user script_user
```

Save and close the file.

We also need to specify the places that suexec will consider a valid directory. This is what our customizable version of suexec allows us to do. Open the suexec configuration file:

```
sudo nano /etc/apache2/suexec/www-data
```

At the top of this file, we just need to add the path to our scripts directory.

```
/home/script_user/scripts/
```

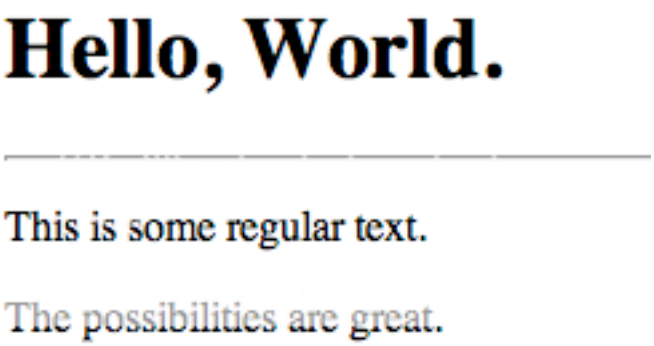
Save and close the file.

Now, all that's left to do is restart the web server:

```
sudo service apache2 restart
```

If we open our browser and navigate here, we can see the results of our script:

```
your_domain.com/scripts/attempt.pl
```



Please note that with suexec configured, your normal CGI directory will not work properly, because it does not pass the rigorous tests that suexec requires. This is intended behavior to control what permissions scripts have.

Conclusion

You can now create scripts and execute them in a relatively secure way. CGI scripts are very helpful for quickly including dynamic content on your site. Suexec allows you to lock down this ability for greater security.