Hiding the fact that your site runs Drupal

Site builders may see questions about how someone can "hide the fact that their site runs Drupal" or "remove the meta generator header." People often want to do this because they feel it will make their site more secure: "if the attacker doesn't know I'm running Drupal, they will have one less piece of data about what attack methods might work. An automated attack script that detects Drupal sites might not find me and therefore might not attack me."

Dealing with the consequences of hiding

Although it would seem to make sense that hiding your use of Drupal would make you secure, it may not be worth the required effort, especially since it can take a very long time to complete the actions in this article. If you do determine to hide your use of Drupal, prepare for the following consequences:

- Every time there is a new Drupal release, your deployment process will take more time.
- You'll need to fully test your site to ensure that everything that you have changed still works, and that no new `http/html` headers, `txt`, `css`, `jss`, user-facing text, themeable functions, or `tpl.php` files were added. If they were added, you'll have to figure out how to override them.
- Debugging will become enormously more difficult, because Drupal developers, site builders, and themers rely on the site "behaving like Drupal." By making the system different, you may remove the tricks they commonly rely on to identify common problems.
- Developers that are new to your team will probably have to spend additional time getting used to the changes to your site.

Hiding that you're using Drupal

Here are some things that you can do to hide the fact you are running Drupal (and the amount of effort it can take):

- Remove the meta generator
- Remove Drupal-specific text files
- Examine your JavaScript and CSS files
- Check the Expires header
- Probe pages and directories for HTTP 200/404/403 status codes
- Look for default text messages
- Look at the HTML

NOTE

It's a significant amount of work to do this both up-front and on an ongoing basis. It would be far

better to simply focus on the real security of your site rather than hiding whether or not you're using Drupal.

Remove the meta generator

Drupal 7 added a meta generator header to declare that the site runs Drupal. It's visible in the source of a standard webpage, and includes only the major version of Drupal:

```
<meta name="Generator" content="Drupal 7 (http://drupal.org)" />
```

To remove it (and add more code and comments to your theme that you'll have to maintain), adding a theme alter function to your `template.php` similar to the following:

```php
<?php
/**
* Implements theme_html_head_alter().
* Removes the Generator tag from the head for Drupal 7
*/
function YOURTHEME_html_head_alter(&$head_elements) {
  unset($head_elements['system_meta_generator']);
}
?>
```

Remove Drupal-specific text files

Drupal contains many text files, and a quick look through the Drupal 7 install finds 27 specific filenames, including:

- `COPYRIGHT.txt`
- `MAINTAINERS.txt`
- `sites/all/README.txt`
- `modules/simpletest/tests/common_test_info.txt`
- `modules/simpletest/files/html-1.txt`
- `modules/simpletest/files/javascript-1.txt`
- `modules/simpletest/files/php-1.txt`

The default behavior for web servers is to send these files directly to visitors without hiding the contents. If your site allows the upload of `.txt` files as attachments, you'll have to remove all of these Drupal-specific text files. To do this, if you're on a Mac or Linux, you can use the following simple command:

IMPORTANT

The following command works recursively, and should be run from your website's docroot. If users

have uploaded files with a `.txt` extension, it will delete them!

```
find . -name '*.txt' -exec rm {} \;
```

Best practices are to remove these Drupal-specific files on a staged or checked-out (`git fetch;git checkout release-name`) version of your site instead of your production website to ensure that you don't require any of the files. For example, Drupal core's SimpleTest relies on some of these files, and you'll need to find an alternate solution for those (or not run tests on production).

Examine your JavaScript and CSS files

JavaScript and CSS files are served as plain text files to site visitors, and can provide additional information about your site. To evaluate your site's code, you can use the Blind Elephant project to identify what code is running.

One method to obfuscate your code is to use the Advanced Aggregation module on a staging site to rebuild your site's aggregated JavaScript and CSS on a controlled basis. You can then move the aggregated files to your production site and then remove all of the production site's CSS and JavaScript files. Note that this process adds significant overhead to each deployment, and attackers can easily identify running modules by looking for specific patterns in the aggregated files.

The most secure (and time-consuming) solution is to remove all CSS and JavaScript from the site, and then rewrite them from scratch so that they're different.

Check the Expires header

Drupal has a default `Expires` header, which you can grep your code for. For example, the following command on an example server:

```
/usr/bin/lwp-request -dem GET drupal.org | grep Expires
```

provides the following results:

```
Expires: Sun, 19 Nov 1978 05:00:00 GMT
```

One method that you can use to set a different Expires (and add more code and comments to your theme that you'll have to maintain) is to use drupal_add_html_head :

```php
<?php
/**
* Implements hook_init().
*/
function mymodule_init() {
  // Change to a different but still "old" value.
  drupal_add_http_header('Expires', 'Sun, 26 June 1978 05:00:00 GMT');
```

```
}
?>
```

Probe pages and directories for HTTP 200/404/403 status codes

Hackers sometimes probe common pages and directory structures to see if they return a 200, 404, or 403 HTML code. Compare /user, /sites , and /totalandutternonsense . If you visit /logout or /user/logout, you can see that the website is running on Drupal 7.

To hide this behavior, use the Rename Admin Paths module to enable you to hide some well-known 200's like `/user` and `/admin`. You'll need to expand on that concept for other well-known 200's (such as `/node`). It may be sufficient to configure your 404 and 403 pages to return the same http header code and content, which hides your directory structure and makes directory probes return less information.

Look for default text messages

If you go to `/user`, you'll probably see a form that says **Enter your @site-name username** and **Enter the password that accompanies your username**. The Request New Password page will say **Username or e-mail address** and, if someone has a really old site (4.7), it will have separate inputs for email and username. If you use a search engine to search for these common phrases , you can identify many sites that run Drupal — and those are just the messages in one part of Drupal's interface.

Fortunately, using either the String Overrides module or localization system can allow site administrators to override every string in the site with their own versions. Keep in mind that there are 4,620 strings in Drupal 7.9 core alone, not counting strings in any contributed modules that you use.

Look at the HTML

Drupal's HTML is (in)famous. Depending on your theme, a user can identify several key snippets of HTML — for example, here's the Bartik homepage body tag for a signed-out user:

```
<body class="html front not-logged-in one-sidebar sidebar-first page-node" >
```

Many modules and themes contain similar bits of HTML that are easily recognizable.

The solution is to override all theme functions so that your HTML is different. Drupal 7 core has 183 themeable functions and 53 `tpl.php` files that would need to be rewritten, and if you use any contributed themes, you'll have many more functions and templates to override. If a new release of code alters the input or output, you would need to fix yours to match, especially if there is a security fix.

This doesn't address those modules that output their own HTML. In these cases, the best case is to patch each module, file an issue in the modules' queue with your patch, and then theme the output of the modules using your added theme function.

Finally, the form id, form_build_id, form_id, and token (the hash values) are all very indicative of a Drupal

site, and are tied pretty heavily into the proper functioning of the Form API system. Obscuring these would be very difficult. Here's part of the login form on drupal.org:

```
<form id="user-login" method="post" accept-charset="UTF-8" action="/user?
destination=home">
...
<input id="form-86a162e837cebba89a70ca115f7a6313" type="hidden" value="form-
86a162e837cebba89a70ca115f7a6313" name="form_build_id">
<input id="edit-user-login" type="hidden" value="user_login" name="form_id">
...
```