

White Paper:

Obfuscating Code with Zend Guard



By Zend Technologies, Inc.

February 2006

Obfuscating Code with Zend Guard 3

Obfuscation..... 3

Encoding Only 4

Basic 4

Strong..... 4

Choosing an Encoding level 5

Applying Obfuscation to Code 6

Strong Obfuscation 7

Exclude Preferences..... 8

The Suggest Feature 11

Manually Adding Functions to the Exclude List..... 12

Fine Tuning the Exclude List..... 14

The Obfuscate Function Name API..... 15

Testing and Debugging Applications after Obfuscation 16

Obfuscating Code with Zend Guard

Zend Guard's wide range of encoding options makes application source code difficult to understand, modify, and maintain. However, the sole objective is to preserve your Intellectual Property from being misused.

The Zend Guard utilizes the well-known concept - the more effort you apply to protecting your application, the more difficult it will be to reverse-engineer.

When creating encoded PHP files, you will have to consider the amount of effort you want to put into the encoding process. This effort should match the level of protection you are seeking for your application.

Obfuscation

Source Code contains various tags and names defined by the programmer. These names are typically made meaningful to make the code easy to understand and maintain, by developers. Obfuscation converts these tags and names into cryptic names, in the sole effort to make the code hard to understand by others, without affecting the code execution.

For example: The variable *color*, when obfuscated will be changed to something that does not have a meaning, such as *a1*. As you can see from the example the execution logic of the code is maintained, but the code has become difficult to understand.

Several options have been provided to suit various code protection requirements. The Zend Guard obfuscation options support various PHP versions (including PHP 4 and PHP 5). Obfuscation is made through a one-way Hash function that generates the name modifications.

The following obfuscation options are provided:

- Encoding Only
- Basic
- Strong

Encoding Only

The Encoding Only option does not apply any obfuscation to code. This option converts PHP files into encoded binary files (Encoding). Seamless to the developer, encoding does not require any developer involvement, thus providing an out-of-the-box experience. Converting PHP files into encoded binary files makes PHP code unreadable by other developers.

It is recommended to use this option in cases when a relatively low protection for source code is required, yet there is a wish to minimize developer involvement while obfuscating the code with the more powerful obfuscation level.

Basic

Basic obfuscation modifies source code local variables. This option provides improved security and yet it is seamless to the developer. This option improves security measures without generating additional overhead. In addition to being obfuscated, all files are encoded during the Basic obfuscation process.

Note:

The combination of encoding and Basic obfuscation ensures that - even if someone does manage to decode encoded files it prevents a third party from exploiting the code.

Strong

Strong obfuscation modifies all function names, function calls (excluding class method calls), classes and class functions. In addition to all the added security measures applied with Basic obfuscation.

Strong obfuscation includes an additional option for excluding specific entities from being obfuscated. Safeguarding code with Strong obfuscation provides the most efficient security coverage for PHP code. (For additional information about Strong obfuscation see, [Applying Obfuscation to Code](#) page, 6).

Choosing an Encoding level

The following table details the different encoding/obfuscation options, their recommended usage, and possible risks.

Level	When to use	Efforts
None (Encoding Only)	Used if Basic obfuscation causes problems with the Code and you do not want to make any efforts that would be required while implementing more powerful Obfuscation.	None
Basic	To gain enhanced security without typically investing in any additional overhead.	Obfuscated code may require sometimes small code adjustments (i.e. while using <code>isset()</code> ¹ on local variables.
Strong	To protect intellectual property making the additional overhead worthwhile.	Obfuscated code may require some customization and additional testing to ensure issues does not pop-up during production (i.e. application returns “function not defined” errors. In this case, the problematic functions should be added to the Exclude list.)

Note:

There is a direct correlation between the amount of files obfuscated and the difficulty understanding and reverse engineering code. Therefore, complete project obfuscation will best protect your application.

¹ `[bool isset (mixed var [, mixed var [, ...]])]`

Applying Obfuscation to Code

The process of obfuscating code begins with generating a project and configuring the project's settings.

Once you have chosen all the settings and have decided on the PHP files that need to be encoded, you can determine what level of obfuscation you should apply to the code.

Determining the obfuscation level should take the following considerations into account:

- 1) Is the code intended for mass deployment?
- 2) How important is the code? (I.e. is it expensive intellectual property?)

If both the answers are "yes", Strong obfuscation is the most suitable obfuscation level. However, it is up to the user to choose whether they need encoding only or the addition of encoding local variables that is obtained with Basic Obfuscation.

Encoding only (None) and Basic obfuscation do not require any additional intervention, except selecting the appropriate option in the Zend Encoder's Obfuscation tab.

Strong obfuscation requires that the user specify a list of entities that should not be obfuscated. This is done through the using the "Exclude Preferences" feature.

Note:

Encoding files with Strong obfuscation requires the latest version of Zend Optimizer be installed. The Zend Optimizer is available for download free of charge from the [Zend Store](http://www.zend.com/store/), at <http://www.zend.com/store/>.

Strong Obfuscation

Safeguarding code with Strong obfuscation provides the most efficient security coverage for PHP code. In order to successfully obfuscate code there are several preparations that have to be made.

These preparations entail identifying functions that should not be obfuscated called “Exclude Preferences”. Exclude Preferences include entities that should not be obfuscated (Such as function names, function calls, classes and class functions that should not be obfuscated).

Most entities can be identified through a preliminary setup procedure, by using the Suggest feature that automatically generates an initial list of entities that should be included in the “Exclude List”.

Additional entities that for some reason the developer does not want obfuscated can be manually added to the Exclude List.

These entities include:

1. Functions that cannot be automatically identified through the setup process such as indirect functions and concatenated functions
2. Functions located during the debugging/testing stage of the application when using the application that underwent the initial encoding
3. Functions that generated “Function not defined” and “Class not defined” message types that only appeared after obfuscating the code

Note:

Errors that occur in the code before obfuscating indicate a problem in the actual code.

The following section is an in-depth description of each of the preparations. The section also describes how to add (manually and automatically) to the Exclude List.

Exclude Preferences

The Strong obfuscation exclusion feature is a means of analyzing PHP code. This analysis is to detect which components in the code should **not** be obfuscated with Strong obfuscation.

When should you exclude entities?

There are several reasons why certain entities be excluded:

The basic guideline is that if the entity causes an error - add it to the exclude list.

However, it is more efficient to identify entities that should be excluded before receiving errors when running/debugging the code.

Always export:

- Functions defined in un-obfuscated code.
- Indirect function calls. This occurs when referencing function calls through a variable holding the function name

For example:

```
function do_mysql_query($query) { ... }
function do_sqlite_query($query) { ... }
if($db == "mysql") {
    $query_function = "do_mysql_query";
} else {
    $query_function = "do_sqlite_squery";
}
$result = $query_function("SELECT * FROM TABLE");
```

In this code example, we can see that the functions *do_mysql_query* and *do_sqlite_squery* are referenced through a variable holding their name. Therefore, these functions should be added to the Exported Functions list.

- When passing functions to the other functions using arguments i.e. callbacks (the solution is to use

The Obfuscate Function Name API).

For example:

```
function myerror() { ... }
set_error_handler('myerror');
or
function myfunc($data) { ... }
array_walk($array, 'myfunc');
```

In this code example, you can see that the functions *myerror* and *myfunc* are used to pass functions to other functions. Therefore, these functions should be added to the export list.

- Functions that implement external interfaces

For example:

```
class c_iter implements Iterator {
function rewind() { ... }
function valid() { ... }
function current() { ... }
function next() { ... }
function key() { ... }
}
```

In this example, the functions: `rewind`, `valid`, `current`, `next` and `key` should be exported because they implement an external interface.

Note:

A full list of functions for each interface can be found in the PHP manual.

- When Functions are used as object callbacks

For example:

```
class VariableStream {
/.../
}
stream_wrapper_register("var", "VariableStream");
```

In this example, functions that are wrapper callbacks should be exported. (See

<http://www.zend.com/manual/function.stream-wrapper-register.php> for full a complete list of callback names and other such functions, like `xslt_set_object`).

- Autoloading classes will not work since the file on the disk would not match the obfuscated name.

There are two ways to define which components should be excluded:

- 1) Use the Suggest feature to recommend functions to be added to the Exclude list.
- 2) Manually add components that you know should be excluded.

The Suggest Feature

The Suggest feature scans the code and seeks suitable candidates. The Suggest feature identifies any strings and functions with the same name. This option is extremely flexible and gives the user the option to discard the suggestions before finalizing their addition to the Exclude List.

Running the Suggest Feature:

- 1) Open a project
- 2) Configure the settings and the list of PHP files to be obfuscated.
- 3) Go to the Zend Encoder's Obfuscation tab and select Strong obfuscation.
- 4) In the Exclude Preferences section select the Suggest button.

The code is scanned and the suggestions are displayed in a suggestions list in a separate popup.

- 5) Select OK to approve the suggestions or Cancel to discard the suggestions.

The suggestions are transferred from the suggestions list to the Exclude List section of the Obfuscation tab.

The Exclude feature remembers the last suggestion. If the code has been changed, use the Regenerate button to scan the code. The Regenerate option refreshes the contents of the Suggest List. (Obfuscation tab | Exclude Preferences | Suggest button | Regenerate button)
After the suggestions have been applied and code has been obfuscated accordingly, the code should undergo the organization's regular testing cycle to test the application.

Manually Adding Functions to the Exclude List

The Suggest feature can identify almost all functions that should not be obfuscated. However, there are certain situations and circumstances that necessitate manually adding entities to the Exclude list.

The Testing/Debugging Process

Entities, that after obfuscation prevent the application from working properly, should be manually added to the Exclude List.

These entities will generate “Function not defined” and “Class not defined” message types only after the code was obfuscated.

Concatenating Strings into Function Names

The only instance the Suggest function cannot identify is when Concatenating Strings into Function Names. This covers instances where the code calls an indirect function name and not the functions real name. This occurs when the real function names are not identified in the code as functions, but rather as strings. The Suggest feature searches only for functions in the code.

For example:

```
function do_mysql_query($query) { ... }
function do_sqlite_query($query) { ... }

$query_function = "do_".$dbname."_query";

$result = $query_function("SELECT * FROM TABLE");
```

Here the functions *do_mysql_query* and *do_sqlite_query* are not directly referenced as functions. They are mentioned in the code as strings not functions and therefore, these functions should be added to the Exclude List

In order to circumvent this, always manually add concatenated strings to the Exclude List.

Indirect Functions

User functions that are in use indirectly or called from un-obfuscated script should be manually added to the Exclude List.

Manually adding functions to the Exclude List:

- 1) In the Obfuscation tab's Exclude List section, click Add.
The All Excluded Entities dialog opens
- 2) Choose one of the file addition options:
 - a. **Excluded Entities:** type the name of the entity that should be added to the Exclude list.
 - b. **Load from file:** browse to a file containing strings of text (to create this file, use a simple text editor to list entities in a file). Make sure that each string represents an entity.






Note:

The "Load from file" option does not validate the contents of the file and assumes that each new line is a separate function without validation.

Fine Tuning the Exclude List

At any time users can choose to entities in the exclude. Disabling an entity in the list means that the entity will be obfuscated. Only selected entities (with an X next to them) will be remain as-is in the code and not be obfuscated.

The Exclude section of the Obfuscation tab has the following search and add/remove options:

- Search: Search - The partial search option is for searching for specific entities in the Exclude List. Typing the name of the entity in the search area automatically reduces the options in the display to gradually expand the search. Click Backspace to delete the content of the Search field letter-by-letter.
-  Clear - Empties the Search field and refreshes the entity display.
-  Clear all - Un-checks the checkbox for all entities - all of the entities will be obfuscated.
-  Clear selected - Un-checks the checkbox for all the selected entities (CTRL + Select) - the selected entities will be obfuscated.
-  Add to all - Checks the checkbox for all the entities - no entities will be obfuscated.
-  Add to selected - Checks the checkbox for all the selected entities - The selected functions will not be obfuscated.

The Obfuscate Function Name API

This API should be used to obfuscate function names that require coordination between functions and called functions.

`obfuscate_function_name`

`string obfuscate_function_name(string function_name)` obfuscate and return the given function name with the internal obfuscation function.

Important Note:

Developer discretion should be used when implementing the `obfuscate_function_name` API. Only use the API in code that will be entirely obfuscated. Using this API with un-obfuscated code will generate a compatibility problem between the obfuscated code and un-obfuscated code.

Testing and Debugging Applications after Obfuscation

Any code that has been changed or manipulated must be verified and checked to determine that it still works. Code that has undergone Zend Guard encoding/obfuscation is no exception.

No matter what type of encoding or obfuscation is applied to the code, it is necessary to validate the code by running a complete QA (Quality Assurance) cycle on the code. Code validation should be repeated after each time the code is encoded or obfuscated.

Extra attention should be given when using Strong obfuscation. The errors found in the code provide an excellent indication to entities that should not be obfuscated and should be included in the Exclude List.

Contact Information

United States and Canada:

Zend Technologies, Inc.
19200 Stevens Creek Blvd.
Cupertino, CA 95014
Tel: 1-888-PHP-ZEND (1-888-747-9363)
Fax: 1-408-253-8801

Central & Eastern Europe:

Zend Technologies GmbH
Bayerstraße 83
80335 München, Deutschland
Tel: +49-89-516199-0
Fax: +49-89-516199-20

International:

Zend Technologies, Ltd.
7 Abba Hillel Street
Silver Building
Ramat Gan, Israel 52136
Tel: 972-3-613-9665
Fax: 972-3-613-9671

Document Feedback:

E-mail feedback to:
documentation@zend.com
Please state the name of the
document in the subject line.