PHP ENCRYPTION
A Pure PHP Solution
By Kurt Payne
April 10, 2004

General Project Description

Two of the biggest challenges facing software developers involve piracy. Software piracy by end users results in income loss for developers. Another problem is the piracy of intellectual property by other developers. This problem is especially prevalent amongst developers of web-based software using open source technologies like PHP.

PHP is a high level programming language employed by many companies to power their website. In the past few years PHP has gained considerable notoriety as a powerful, scalable language with very fast development time. Recently Yahoo switched from its proprietary scripting language, yScript, to PHP. For more information, please see http://news.com.com/2100-1023-963937.html.

Any PHP script is interpreted and compiled on demand by the Zend (PHP) engine whenever a script is requested. PHP scripts, therefore, are distributed in source form and not in compiled form. Distributing software in source form makes software piracy and intellectual property theft very easy.

This project allows developers to distribute their source code in an encrypted format. Because the source code is encrypted, it is much harder to edit and read. If you can’t edit the source code, you can’t disable license enforcement measures. If you can’t read the source code, you can’t look for security exploits, you can’t steal proprietary algorithms, and you can’t release any “plug-ins” that are not authorized by the original developer.
Reasons for Hiding Source Code

Having the source code available for a program makes it very easy to disable any license enforcement measures and distribute it illegally. It would also be very easy to steal any proprietary algorithms. Other developers could also make significant changes to a program’s interface and resell the application under a different name.

Project source code can also compromise the security of a piece of software. For only the purchase price of the software, anyone can audit the code for potential exploits and use those exploits against other customers.

One final reason that developers would like to protect source code is to prevent any unauthorized “plug-ins” for their software. A plug-in is a piece of software that interacts with a parent software application to perform another function or extend an existing function. For example, if a developer creates banner-ad management software that shows and tracks the views / clicks of GIF and JPEG banners, someone may release a plug-in that allows the software to show rich media banners made in Flash or Java.

The original developer may not want any plug-ins released for the software. Customers may place the burden of support for plug-ins on the original developer and may complain if a plug-in is not forward compatible. The developer can also make more profit by releasing features satisfied by these plug-ins as part of the core software package.

Different Approaches

In researching this project, four methods for hiding PHP source code were found.

Obfuscation
This method involves removing all comments and unnecessary whitespace from the original source code. Then, all variable names, class names, and function names are replaced with random character combinations.
**Benefits**
Obfuscation makes code very unreadable. Obfuscation requires no modification to the environment, produces smaller files, and gives slightly faster parse times (as a result of smaller files). There are also obfuscation tools available for free.

**Liabilities**
Obfuscation can be partially reversed. While comments can’t be restored, the formatting can be restored using a code beautification tool. Careful code analysis coupled with the search & replace feature on any text editor can, once again, give meaningful names to variables, functions, and classes. However, if you’re interested only in editing the code, you don’t need to reverse it completely. Code analysis can guide you to a particular location in the software where you need to make your change.

Distributing software patches / upgrades is also difficult. You must make sure that each file uses the same variable / function names as all of the other files. For example, if you write an application with a function called “main()” and you distribute an obfuscated copy with the same function renamed to “faC933oZ2()” you must make sure that any future upgrades / patches that rely on “main()” make calls instead to “faC933oZ2().” (Good obfuscation software will use checksums to generate the new variable, class, and function names and should solve this problem.)

**Binary Encryption**
This method involves encrypting PHP files on a byte level. The binary file is then distributed with one or more “loaders” which are loaded by PHP at runtime and provide the decryption routines necessary to decrypt and run the files.

**Benefits**
This type of encryption is difficult to reverse. Code cannot be directly edited nor can it be read without reversing the encryption. Since the decryption routines are compiled, the overall execution time is only slightly slower than an unencrypted file. For a runtime comparison of several existing PHP binary encryption products and unencrypted code please see [http://www.ioncube.com/benchmarks.php](http://www.ioncube.com/benchmarks.php).

**Liabilities**
This type of encryption can be reversed. Using a PHP-CLI binary with debugging options enabled can restore the original source code completely. More information on this method can be found at
http://www.ioncube.com/source_display.php. One of the biggest liabilities for this type of encryption is that you are required to alter the software environment. The loaders required to decrypt and run the encrypted files are compiled for specific operating systems and specific versions of PHP. Some versions of PHP with debugging and thread safety options will not allow the loading of other libraries at runtime causing some customers to not be able to use the software. Developers using this method often distribute all available loaders with their application – adding up to 2 megabytes to the distribution. The applications are also restricted to the operating systems and PHP versions supported by the loaders.

**Source Compilation**

As mentioned above, PHP scripts are compiled and stored in memory when requested. One company, ionCube, has released software which saves PHP code in its compiled form and then encrypts the compiled code.

*Benefits*
This type of compilation is impossible to reverse. Reading and editing the compiled code is also impossible. Further, since scripts are decrypted to a compiled form, there is no compilation overhead and the scripts actually run faster as a result. For a runtime comparison please see [http://www.ioncube.com/benchmarks.php](http://www.ioncube.com/benchmarks.php).

*Liabilities*
This software is expensive, costing up to $350 for a single license. When compared to the average cost of PHP software ($10 - $50), this can be a big expense. This type of encryption is also subject to the same environmental limitations as the binary encryption method mentioned above.

**PHP Encryption**
This method uses pure PHP code to decrypt and execute other PHP files.

*Benefits*
Since the decryption routine is written in pure PHP, there is no need to alter the environment. Since this method encrypts the original code, it is impossible to read or edit the code without first decrypting the file. However, if a project is planned poorly, it is possible to inject malicious code into an encrypted file.
Liabilities
Because the decryption code is written in PHP, it must be interpreted and run before the original code can be interpreted and run. This results in slower execution time. Further, since any encrypted code must be able to be decrypted by PHP, it is impossible to write a solution in pure PHP that cannot be manually decrypted. This means that any encrypted code can be completely reversed.

Project Goals
The goals of this project were to create a relatively easy and inexpensive way to create self-decrypting PHP scripts with license enforcement using pure PHP code.

License Enforcement
Any files encrypted by this project can have license enforcement code inserted into each file. All of the following methods can work in combination. The current licensing options available are:

- Domain Name License – The script will only run on a certain domain name. The licensing is checked by comparing the last “n” characters of the domain to the licensed domain, where “n” is the string length of the licensed domain. For example, a script licensed to run on “nmu.edu” would run on “euclid.nmu.edu” as well.

- Server IP License – The script will only run on a server with a certain IP address.

- Client IP License – The script will only run when requested by a client with a certain IP address.

- Expiration Date – The script will only run before a certain date. This date relies on the server’s time, and must be set absolutely and not relatively. For example, a script can be set to expire on April 30th, 2004 but not 30 days from when it is first run. However, using the API, an expiration date can be set automatically.
• File Checksum – The script performs an MD5 checksum on itself and will not run if its checksum does not match the checksum it recorded at the time of its encryption.

**Failures**
License failures are handled in one of two ways. Upon failure, a script can display a message, or a script can forward the user to a URL. After the message is displayed or after the user is forwarded, the script will exit and cease execution.

When a user is forwarded to a URL, certain parameters are appended to the URL. This is done so that any script that the user is forwarded to knows everything about the license failure. For example, if a script that is licensed to “euclid.nmu.edu” is run on “cs.nmu.edu” the user will be forwarded to a URL that looks this:

```
```

In this case, the script “violation.php” would know that a client with an IP address of 172.16.4.110 visited [http://cs.nmu.edu/protected-script.php](http://cs.nmu.edu/protected-script.php). This script caused a domain violation and was licensed to run on the domain “euclid.nmu.edu.” The server where the script was “cs.nmu.edu”, had an IP address of 198.110.193.9, and the clock on the server was set to 2004-04-07 22:06:47 (transmitted above as a Unix timestamp).

This information is useful for detecting attempted theft and user configuration error. This project includes a script that logs license violations.

**Extra Options**

Scripts can optionally encrypt any output using JavaScript. This is achieved by inserting code into the original source which turns on output buffering and runs all output through a function that will encrypt the output into self-decrypting JavaScript. This option is available to prevent users from viewing the source code of the HTML output of the scripts and stealing any design code.

This feature should be disabled if the target script produces any non-HTML output or uses heavy JavaScript in the HTML output. This option
is also incompatible with the output disabling technique mentioned in the “Challenges” section below. In any scripts that try to use both options, the output encryption option takes precedence and automatically disables the output disabling code.

Scripts can also include a nag screen. This method has been used for years by shareware developers to prompt users to pay for the application they’re using. Since HTTP is a stateless protocol, displaying a nag screen once per session is achieved through the use of a session cookie. When the user requests a script, that script looks for a specific cookie on the user’s system. If the cookie is not found, a nag screen is displayed with a message defined at encryption time and a cookie is set to let the script know that the user has already seen the nag screen. The cookie disappears once the user closes his/her browser.

**Challenges**

The biggest challenge of this project was making the encrypted file as difficult to decrypt as possible. Since any file must be able to be decrypted by PHP, the original source code can be exposed if a user knows enough PHP to perform the decryption routine manually. However, several methods were employed to make the files as difficult to read as possible:

- Encryption can be used recursively making manual decryption a long and arduous process.

- Each script is bundled with its own decryption routine. This means that every file is encrypted with a different key and there is no single entry point for an entire project. If there was one file which handled all decryption purposes, then reverse engineering that file would provide a way to restore the original source code of the entire project.

- The output of the encryption is runnable PHP code. This code has a set structure, but almost all of the variable names are generated as random 33 character strings. There are some variable / function names which must be preserved across all encrypted files and must have a static name. This employs the techniques used in obfuscation to create code that is very difficult to read.
Conclusions

Encryption using pure PHP is an effective way to distribute smaller projects in relative safety. While the code can be completely reversed, this requires a very high understanding of programming and of the PHP language.

Firetank Software (www.firetanksoftware.com) has used this technology to distribute its Marketing Manager product to over 200 customers with no compatibility problems and no reported theft. Marketing Manager relies on loadable modules and, to date, no unauthorized third-party modules have been released.

However, for larger projects, where performance and security are of greater concern, ionCube’s PHP encoder currently provides the best security and best performance of any other solutions.

API

This project is object oriented. The objects can be used in other scripts to generate encrypted PHP code automatically. For example, a developer may wish to offer a downloadable demo for a certain application. Using this API, the developer’s website can ask the user for a domain name and then generate an encrypted application which expires 30 days in the future and licensed only to that particular domain. This script is shown below in the “Example” section.

Usage

The API can be accessed via a settings file or via a set of project instructions.

A line in a settings file looks like this:

Listing 1: SettingName|Value|Loadable|JS_Name

Project setting instructions look very similar:

Listing 2: $project->setValue('SettingName', 'Value', Loadable, 'JS_Name');

Where:

- $project – The instance of a Project object
- Setting name – The name of the setting (e.g. UseGzip)
- Value – The value of the setting (e.g. 1)
• Loadable – 1 or 0, determines if the setting can be loaded into the Encryptor object. (e.g. ‘UseGzip’ is a loadable setting, ‘SnippetStrength’ is not).
• JS_Name – The name of the form element on the HTML page where the setting is stored. For example, if the ‘UseGzip’ setting is stored in a drop down menu with the name of ‘gzip’, this field would be ‘gzip.’ This field is used when loading settings from a file via the HTML form.

Example
The following example takes 2 variables passed on the query string (UseGzip and DomainLockDomain), and encrypts “input_script.php” with settings contained in “ftpe_settings.ini” along with licensing “input_script.php” for the domain specified by DomainLockDomain and setting the expiration date 30 days in the future. The “ftpe_settings.ini” file was generated through the program’s user interface.
<?php

/******************************
/* Set up the project and load our settings */
******************************

// Get necessary definitions
include('../objects/Encryptor.class.php');
include('../objects/Setting.class.php');
include('../objects/Project.class.php');
include('../objects/Zip.class.php');

// Create the objects we'll use
$enc = new Encryptor();
$project = new Project();
$zip = new Zip();

// Load the settings into the project & object
$project->setConfig('./ftpe_settings.ini', true);

// Override any default settings here
$project->setValue('DateLockDate', strtotime('+30 days'), 1, 'date_date');
$project->setValue('UseGzip', $_GET['UseGzip'], 1, 'gzip');
$project->setValue('DomainLockDomain', $_GET['DomainLockDomain'], 1, 'domain_domain');

// Load the settings into the encryptor object
$project->loadConfig($enc);

/******************************
/* Now we're ready to encrypt files */
******************************

// Read the file in
$fp = fopen('./input_script.php', 'r');
$page = fread($fp, filesize('./input_script.php'));
fclose($fp);

// Give the encryptor the unencrypted page
$enc->Page = $page;

// Set the strength
$enc->Strength = 2;

// Encrypt the page
$page = $enc->getPage();

// Add the files to the zip file
$zip->addFile('script.php', $page);

/******************************
/* Trigger the download header and send the file */
******************************

$zip->sendDownloadHeader('software.zip');
echo $zip->getFile();
exit();
?>
File listing for “ftpe_settings.ini”

UseGzip|1|1|gzip
GzipLevel|9|1|gzip_level
Checksum|1|1|checksum
ChecksumErrorMethod|msg|1|checksum_error_method
ChecksumURL||1|checksum_error_url
ChecksumMessage|This file is corrupt.|1|checksum_error_msg
DomainLock|1|1|domain
DomainLockDomain|euclid.nmu.edu|1|domain_domain
DomainLockErrorMethod|url|1|domain_error_method
DomainLockURL|http://euclid.nmu.edu/~kpayne/ftpe/violations/violation.php|1|domain_error_url
DomainLockMessage||1|domain_error_msg
DateLock|1|1|date
DateLockDate|1041397200|1|date_date
DateLockErrorMethod|msg|1|date_error_method
DateLockURL||1|date_error_url
DateLockMessage|This file has expired.|1|date_error_msg
ServerIPLock|0|1|serverip
ServerIPLockIP||1|serverip_ip
ServerIPLockErrorMethod|msg|1|serverip_error_method
ServerIPLockURL||1|serverip_error_url
ServerIPLockMessage||1|serverip_error_msg
ClientIPLock|0|1|clientip
ClientIPLockIP||1|clientip_ip
ClientIPLockErrorMethod|msg|1|clientip_error_method
ClientIPLockURL||1|clientip_error_url
ClientIPLockMessage||1|clientip_error_msg
NagScreen|1|1|nagscreen
NagScreenMessage|Shareware. <A HREF="[link]">Ok</A>|1|nagscreen_message
EncryptOutput|1|1|encoutput
UseOutputSecurity|1|1|outputsecurity
UseFileSecurity|1|1|filesecurity
InputType|file|0|inputtype
IncludeSettings|No|0|include_settings
SnippetStrength|0|0|SnippetStrength
Reference

// Gzip Settings
UseGzip – 1 or 0 – Should the script be compressed after encoding
GzipLevel – 0 through 9 – The level of compression to use

// Checksum settings
Checksum – 1 or 0 – Should the script include a self checksum test
ChecksumErrorMessage – ‘msg’ or ‘url’ – What happens upon failure
ChecksumURL – String – URL to forward to upon failure
ChecksumMessage – String – Message to display upon failure

// Domain settings
DomainLock – 1 or 0 – Should the script check its domain
DomainLockDomain – String – Licensed domain name
DomainLockErrorMessage – ‘msg’ or ‘url’ – What happens upon failure
DomainLockURL – String – URL to forward to upon failure
DomainLockMessage – String – Message to display upon failure

// Date settings
DateLock – 1 or 0 – Should the script expire after a certain date
DateLockDate – Date when the script expires (Unix integer timestamp)
DateLockErrorMessage – ‘msg’ or ‘url’ – What happens upon failure
DateLockURL – String – URL to forward to upon failure
DateLockMessage – String – Message to display upon failure

// Server IP settings
ServerIPLock – 1 or 0 – Should the script check the server’s IP address
ServerIPLockIP – String – The IP address of the licensed server
ServerIPLockErrorMessage – ‘msg’ or ‘url’ – What happens upon failure
ServerIPLockURL – String – URL to forward to upon failure
ServerIPLockMessage – String – Message to display upon failure

// Client IP settings
ClientIPLock – 1 or 0 – Should the script check the client’s IP address
ClientIPLockIP – String – The IP address of the licensed client
ClientIPLockErrorMessage – ‘msg’ or ‘url’ – What happens upon failure
ClientIPLockURL – String – URL to forward to upon failure
ClientIPLockMessage – String – Message to display upon failure

// Nag screen
NagScreen – 1 or 0 – Should the script display a nag screen
NagScreenMessage – String – The message to display (cannot contain “\r” or “\n” characters, should escape all single and double quotes, and should contain a link to continue, e.g. <A HREF=”[link]”>Continue</A>)

// Browser output settings
EncryptOutput – 1 or 0 – Should the script encrypt its output

// Output security settings
UseOutputSecurity – 1 or 0 – Should the decryption script hide its output

// File security settings
UseFileSecurity – 1 or 0 – Should the decryption key be based on the file contents