

PHP LockIt! - an example of a bad product

This report focuses on one of the weird products which you may find on the Internet and its main purpose is to share some of my knowledge in the field of breaking simple and weak encryption methods.

Introduction

One day, one of my friends asked me, if I can look to a product for encryption PHP files which seems a lot more financially attractive (\$29.99) than the *Zend Encoder* (\$960 per year or \$2.400 for lifetime license). I haven't had too much free time, so I told him to check it out on his own, encode some files, and I will just look to the output. So he did, and I got back an encoded file. For my surprise, the file was a usual PHP file and I instantly knew that this encryption could be easily broken. When I told him, that this product is useless, and a beginner can break it, he did not believe me. So I decoded the encoded file. And it took less than 10 minutes.

The software

Trial version

You may get a 30 day fully featured evaluation and demonstration version of the product at <http://www.phplockit.com/demo.php>. After the installation you can encode your files (if you click to OK in a dialog informing you that you are using the demo version everytime you start encoding).

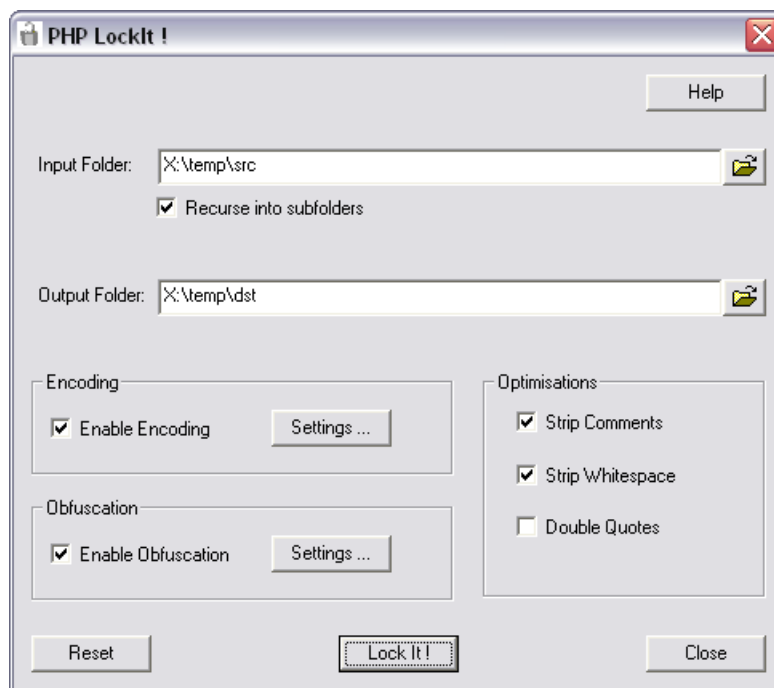


Fig.1: The user interface

Full version

The following note is from the shopping page (<http://www.phplockit.com/buy.php>):

Once your payment has been made you will be sent an email giving a URL, allowing you to download the software immediately.

I thought that there will be some hashed names, but my friend was quicker - he gave me this link: <http://www.phplockit.com/downloads/>. When I went there, I saw a list of files with hashed names. Maybe somebody forgot to turn off the *Indexes...* or they just want to keep the quality of the product and the web presentation at the same level ;-)

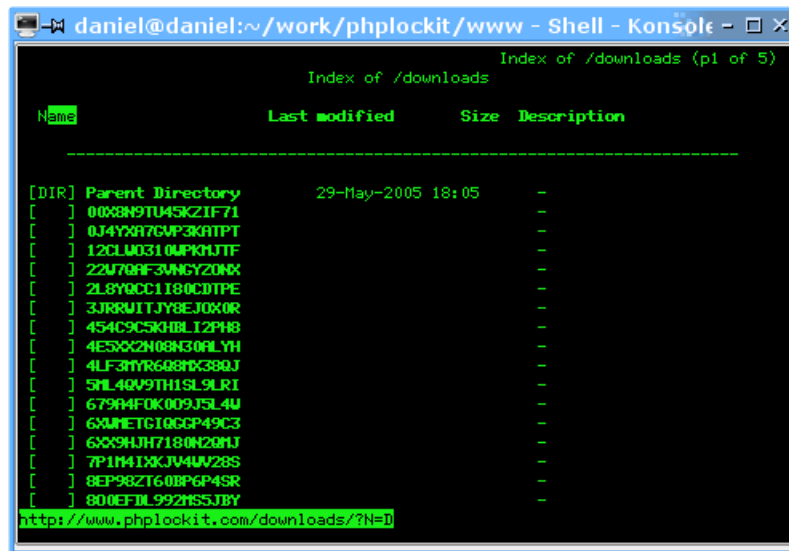


Fig.2: Proof of availability of full version

Another proof that getting the full version is possible (no, we haven't paid for it - if you're thinking that we bought it):

```
$ ls Php* -l
-rw-rw-r-- 1 daniel users 288256 Jun 26 22:20 PhpLockIt.msi
-rw-rw-r-- 1 daniel users 292352 Jun 26 22:20 PhpLockIt_demo.msi
```

However, we haven't installed the full version. This decision was taken because the web site says that the demo is full featured, and also because we are really just evaluating the software. Anyway, would you use this *crap*, even when you could get it for free?

Encoding

Let's start the evaluating. We made an example file:

```
<?php
    $hello = "Hello world!";
    echo $hello;

?>
```

After encoding, we got this:

```
<?php // This file is protected by copyright law and provided under license.
Reverse engineering of this file is strictly prohibited.
$00000000=__FILE__; $00000000=__LINE__; $00000000=76; eval((base64_decode('JE8w
MDBPME8wMD1mb3BlbigkT09PME8wTzAwLCdyYicpO3doawxlKC0tJE8wME8wME8wMClmZ2V0cygkTzA
wME8wTzAwLDEwMjQpO2ZnZXRzKCRPMDAwTzBPMDA5NDA5Nik7JE9PMDBPMDBPMDOcYmFzZTY0X2RlY2
9kZShzdHJ0cihmcmVhZCgkTzAwME8wTzAwLDM3MiksJ0VudGVyew9ld2toUkhZS05XT1VUQWFCYkNjR
GRGZkdnSWlkakxsTWlQcFFxU3NWdlh4WnowMTIzNDU2Nzg5Ky89JywnQUJDREVGR0hJSkMtMTU5PUFFS
U1RVVldYWVphYmNkZWZnaGlqa2xtbm9wcXJzdHV2d3h5ejAxMjM0NTY3ODkrLycpKSk7ZXZhbCgkT08
wME8wME8wKTs='))); return;?>
kr9NHenNhenNhe1lFMamb3klFoxiC2APk19gOLlHOa9gkZXJkZwVkr9NTznNhr8Xht4JkZwShokiF2A
2Yy9LcBYvcoAPF3OZfUwPcmklCBWPkr8XHenNhr8XhtXLt08Xhr8XHeEXhUXmOB50cbk5d3a3D2iUUY
lRtlfNaaOnCAkJW2YrcrcMO2fkdApQToxYdanXAbYTF1c2BuidGjExHjh0YTC3KeLqRz0mRtfnWLYrO
AcuUrhU0xYTL9WAakTayaBalicBMyJC20lcMfPDBpqdolVd3nxFmY0fbc3Gul6HerZHZw1YjF4KUSv
kZLphUL7cMysd3YlhtONHeEXTznNheEpK2a2CBXPKr9NHenNhenNhtL7eWPLUAlkUalkUalkUalkwe0
IwLildoxvwufvFMxLwUw7eWplC2iVwtOkUalkUalkUalkUAL7eWP=
```

Yep, it's *a bit* bigger than the source (1084 bytes vs. 63 bytes). But when you try running it, it works well, so there's no problem yet.

I can say that I am an experienced PHP programmer, who often uses a lot more features of that given language than others. One of the advanced features of PHP is that you can dynamically execute some code or evaluate expressions, as is shown in the following example:

```
<?php
    $name = 'hello';
    $$name = 'Hello world!';

    echo $hello;

?>
```

When you encode this *simple* script (with obfuscation turned on) it will not work anymore, but that was more or less predictable, that the obfuscation will fail on dynamic evaluation.

Decoding

When I saw the construct composed from `eval` and `base64_decode` functions, it was clear, that it is quite easy to decode such a script. That was mostly from my experience with web pages, where the JavaScript code is hidden with the same method - by transforming the source code (script) into a sort of transport encoding. On the client side the string is decoded and evaluated with a function typical for interpreted languages - `eval` (could be found in both PHP and JS).

Legal issues

Every encoded file contains a comment, that the reverse engineering is prohibited:

This file is protected by copyright law and provided under license. Reverse engineering of this file is strictly prohibited.

Moreover, the readme.txt which came from the installation contains a more general rule:

6. You may not decompile, reverse engineer, disassemble, or otherwise reduce any files generated by the Software.

But... we're just evaluating the software and that (*disassemble for study purposes*) should be allowed by the law. I am not a lawyer, I could be wrong, but here is one similarity - you build a feet tall fence around your house and you write a note to it - "Do not step over". Would you feel safe? If we are evaluating the solution, we need to do it properly.

On the other side, when you look to today's data encryption algorithms, they are all well documented and the *cipher-text only attack* is possible only with brute force - there are no other methods which would be more effective.

Let's decode

Decoding an encrypted file is always a fun. You could expect that some features will be present, but as the opposite (encoding) side is unknown, there are always surprises.

The file encoded by *PHP LockIt* consists from two or three parts, depending how are you looking at it. On the first line, a PHP code block is opened and followed directly by a comment about the prohibition of the reverse engineering. On the second line is some PHP code with a closing tag so the third line is not parsed, and it seems that is somehow encoded.

If we want to see more, we need to decode the following code (indented for better idea about the structure):

```
<?php // This file is protected by copyright law and provided ...

    $00000000=__FILE__;
    $00000000=__LINE__;
    $00000000=76;
    eval((base64_decode('...some random chars...')));
    return;

?>
```

The task is relatively simple, because we can easily show, what is the `eval` evaluating:

```
$ php
<?php echo base64_decode('...the random chars...'); ?>
```

The above script's output gave the following code (again, indented):

```
$000000000=fopen($000000000,'rb');
while(--$000000000)fgets($000000000,1024);
fgets($000000000,4096);
$000000000=(base64_decode(strtr
( fread($000000000,372)
, 'EnteryouwkhRHYKNWOUTAaBbCcDdFfGgIiJjLlMmPpQqSsVvXxZz0123456789+/'=
, 'ABCDEFGHIJKLMNopqrstuvwxyz0123456789+'
)));
eval($000000000);
```

Now we can tell that the executing of the script is not a one pass job, because we have another dynamic evaluation here. The code before works this way:

- open the current file for binary safe reading
- read lines which contains comments - the variable inside the `while` condition contains "2", as it was assigned on the second line via the `__LINE__` preprocessor expression, the `while` loop is therefore executed only once (`--2` evaluates to 1, loop is executed, `--1` evaluates to 0, loop is not executed)
- skip the first pass script by reading out a line from the input file
- read the rest of the file (actually it is the non-PHP part of the file)
- decipher a substitution cipher (with `strtr` function)
- decode from the base64 encoding
- running the result string with `eval`

To get the contents of the evaluated string, we can again use a simple PHP script:

```
$ php
<?php echo base64_decode(strtr(substr('...the random thing...',0,372)
, 'EnteryouwkhRHYKNWOUTAaBbCcDdFfGgIiJjLlMmPpQqSsVvXxZz0123456789+/'=
, 'ABCDEFGHIJKLMNopqrstuvwxyz0123456789+')); ?>
```

And the result is (a bit indented):

```
$000000000=ereg_replace('__FILE__'," ".$000000000." "
,(base64_decode(strtr(fread($000000000,$000000000)
, 'EnteryouwkhRHYKNWOUTAaBbCcDdFfGgIiJjLlMmPpQqSsVvXxZz0123456789+/'=
, 'ABCDEFGHIJKLMNopqrstuvwxyz0123456789+')))));
fclose($000000000);
eval($000000000);
```

The functionality of that fragment is replacing the `__FILE__` strings in the evaluated code, which is a replacement of the preprocessor's job, so it tells us, that we are quite close to the end. The `fread` function reads as many bytes, as was specified directly (not encoded) in the encoded file. These bytes are then again deciphered (simple substitution cipher) and then decoded from the base64 encoding. When we do one more round:

```
$ php
<?php
echo base64_decode(strtr(substr('... the random thing...',372,76)
, 'EnteryouwkhRHYKNWOUTAaBbCcDdFfGgIiJjLlMmPpQqSsVvXxZz0123456789+/'=
, 'ABCDEFGHIJKLMNopqrstuvwxyz0123456789+')); ?>
```

We get the final result:

```
$IIIIIIIIIIII = "Hello world!";
echo $IIIIIIIIIIII;
```

Yep, that is the first source code, just a bit obfuscated.

PHP UnLockIt!

The process of decoding can be automated, so I wrote a little script which will decode the given file. The decoding itself is done with an increased level of debugging, because the obfuscated names are replaced by a semantically equivalent identifier (such as handle, offset, filename).

```
#!/usr/bin/php
<?php

/*
 *   PHP LockIt! unlocker
 *   by Daniel Rozsnyo [ daniel@rozsnyo.com ]
 *
 *   "a proof that security through obscurity doesn't work"
 *
 */

$input = file_get_contents( $filename = $argv[1] );

if (ereg( "[$]([^=]+)=[_][_]FILE[_][_][]"
        . "[$]([^=]+)=[_][_]LINE[_][_][]"
        . "[$]([^=]+)=[][0-9]+)[]"
        . "eval.*(base64_decode.[^']*['].)", $input, $x) {

    // first pass
    $filename = $x[1];
    $linenum  = $x[2];
    $offset   = $x[3];
    $offset   = $x[4];
    $decoder  = $x[5];

    // get the decoder
    $decoder = eval("return $decoder;");

    // decipher the decoder
    ereg("[$]([^=]+)=[.]*[$]([^=]+).base64", $decoder, $x);
    $ihandle = $x[1];
    $iout    = $x[2];

    $decoder = str_replace($iout      , 'output'  , $decoder);
    $decoder = str_replace($ihandle  , 'handle'   , $decoder);
    $decoder = str_replace($linenum  , 'line'    , $decoder);
    $decoder = str_replace($filename , 'filename', $decoder);

    // the decoder is our slave!
    $decoder = str_replace('eval($output);', 'return $output;', $decoder);
    $decoder = '$line=2;'.$decoder;

    // sandbox
    $next = eval($decoder);

    // remove expiration time
    $next = ereg_replace("^if[^;]+;", '', $next);

    // some more deciphering
    ereg('^[$]([^=]+)=', $next, $x);
    $next = str_replace($ihandle, 'handle', $next);
    $next = str_replace($filename, 'filename', $next);
    $next = str_replace($x[1], 'output', $next);
    $next = str_replace($ioffset, 'offset', $next);

    // the decoder is our slave, again!
    $next = str_replace('eval($output)', 'return $output', $next);

    // final pass
    $code = eval($next);

    // finish
    echo '<', '?php', $code, '>', "\n";

} else die("Failed. No base64_decode anymore.\n");

?>
```

Using the un-locker

We can very, very easily decode the files:

```
$ ./unLockIt.php ./demo.php
<?php
$IIIIIIIIIIIII = "Hello world!";
echo $IIIIIIIIIIII;
?>
```

As I mentioned, the functionality of a script can be broken by obfuscation - and here is the proof that it is broken:

```
$ ./unLockIt.php ./advanced.php
<?php
$name = 'hello';
$$name = 'Hello world!';
echo $IIIIIIIIIIII;
?>
```

There are some options in the encoder, but in real world they are unusable, dynamic evaluation can not be described by a list of exceptions.

Opinion of the authors

Quoting again some *lies* from the web site of the vendor:

```
.....
No security system is 100% secure. PHP LockIt! will deter the vast majority
of potential "crackers". Breaking into a PHP LockIt! encoded script would
require considerable expertise and effort.
.....
```

Could you agree with the authors, that the few copy paste actions with the mouse on the *nix shell into a command line version of the PHP shown above is *a considerable expertise and effort*? I don't think so.

Conclusion

So there is only one serious conclusion: **DO NOT USE PHP LockIT!**

If your PHP scripts need protection, go rather for products which are using PHP byte-code encodings, like *Zend Encoder*. They are in different price range, but they are really capable of fulfilling security related features which they promise.

When you can not afford this kind of protection, there is an alternative solution of different kind. Instead of spreading encoded scripts keep them un-encoded on a safe place - on one of your servers and try to use a service oriented business model - when you will offer only the functionality (service), not the source code itself. I know that this is not usable in all cases, but gives you a good level of protection of your intellectual property.