Exploitable PHP functions

KEEP YOUR CLOUD TO YOURSELF.
Private servers. Private clouds. Private network. Private. Private. Private.
SOFTLAYER an IBM Company
LEARN MORE >

I'm trying to build a list of functions that can be used for arbitrary code execution. The purpose isn't to list functions that should be blacklisted or otherwise disallowed. Rather, I'd like to have a `grep` -able list of *red-flag* keywords handy when searching a compromised server for back-doors.

The idea is that if you want to build a multi-purpose malicious PHP script -- such as a "web shell" script like c99 or r57 -- you're going to have to use one or more of a relatively small set of functions somewhere in the file in order to allow the user to execute arbitrary code. Searching for those those functions helps you more quickly narrow down a haystack of tens-of-thousands of PHP files to a relatively small set of scripts that require closer examination.

Clearly, for example, any of the following would be considered malicious (or terrible coding):

```
<? eval($_GET['cmd']); ?>

<? system($_GET['cmd']); ?>

<? preg_replace('/.*/e',$_POST['code']); ?>
```

and so forth.

Searching through a compromised website the other day, I didn't notice a piece of malicious code because I didn't realize `preg_replace` could be made dangerous by the use of the `/e` flag (*which, seriously? Why is that even there*?). Are there any others that I missed?

Here's my list so far:

Shell Execute

- `system`
- `exec`
- `popen`
- *`backtick operator`*
- `pcntl_exec`

PHP Execute

- `eval`
- `preg_replace` (with `/e` modifier)
- `create_function`
- `include` [`_once`] / `require` [`_once`] (*see mario's answer* for exploit details)

It might also be useful to have a list of functions that are capable of modifying files, but I imagine 99% of the time exploit code will contain at least one of the functions above. But if you have a list of all the functions capable of editing or outputting files, post it and I'll include it here. (And I'm not counting `mysql_execute` , since that's part of another class of exploit.)

php security grep

edited Oct 19 '10 at 17:28

community wiki
4 revs, 2 users 65%
tylerl

locked by George Stocker ♦ Jun 28 '12 at 2:07

This question exists because it has historical significance, but **it is not considered a good, on-topic question for this site**, so please do not use it as evidence that you can ask similar questions here. This question and its answers are frozen and cannot be changed. More info: help center.

43 as a sidenote, I'd like to see that list published in the near future, if possible :) – yoda Jun 25 '10 at 4:40

16 @yoda: published where? I'll keep the list updated here, since SO is the Source of All Knowledge. – tylerl Jun 25 '10 at 8:36

3 What does the `/e` modifier do? – Billy ONeal Sep 13 '10 at 4:13

6 @Billy: the `e` modifier makes the replacement string to be evaluated as PHP code. – nikc.org Sep 13 '10 at 6:20

1 It has to be said: executing the code in the regex is something Perl and possibly Python do too, not something exclusive to PHP. I don't know the details, though. – Adriano Varoli Piazza Sep 20 '10 at 18:21

|

23 Answers

To build this list I used 2 sources. A Study In Scarlet and RATS. I have also added some of my own to the mix and people on this thread have helped out.

Edit: After posting this list I contacted the founder of RIPS and as of now this tools searches PHP code for the use of every function in this list.

Most of these function calls are classified as Sinks. When a tainted variable (like $_REQUEST) is passed to a sink function, then you have a vulnerability. Programs like RATS and RIPS use grep like functionality to identify all sinks in an application. This means that programmers should take extra care when using these functions, but if they where all banned then you wouldn't be able to get much done.

"With great power comes great responsibility."

--Stan Lee

Command Execution

```
exec           - Returns last line of commands output
passthru       - Passes commands output directly to the browser
system         - Passes commands output directly to the browser and returns last line
shell_exec     - Returns commands output
`` (backticks) - Same as shell_exec()
popen          - Opens read or write pipe to process of a command
proc_open      - Similar to popen() but greater degree of control
pcntl_exec     - Executes a program
```

PHP Code Execution

Apart from `eval` there are other ways to execute PHP code: `include` / `require` can be used for remote code execution in the form of Local File Include and Remote File Include vulnerabilities.

```
eval()
assert()  - identical to eval()
preg_replace('/.*/e',...) - /e does an eval() on the match
create_function()
include()
include_once()
require()
require_once()
$_GET['func_name']($_GET['argument']);
$func = new ReflectionFunction($_GET['func_name']); $func->invoke(); or $func-
>invokeArgs(array());
```

List of functions which accept callbacks

These functions accept a string parameter which could be used to call a function of the attacker's choice. Depending on the function the attacker may or may not have the ability to pass a parameter. In that case an `Information Disclosure` function like `phpinfo()` could be used.

```
Function                    => Position of callback arguments
'ob_start'                  =>  0,
'array_diff_uassoc'         => -1,
'array_diff_ukey'           => -1,
'array_filter'              =>  1,
'array_intersect_uassoc'    => -1,
'array_intersect_ukey'      => -1,
'array_map'                 =>  0,
'array_reduce'              =>  1,
'array_udiff_assoc'         => -1,
'array_udiff_uassoc'        => array(-1, -2),
'array_udiff'               => -1,
'array_uintersect_assoc'    => -1,
'array_uintersect_uassoc'   => array(-1, -2),
'array_uintersect'          => -1,
'array_walk_recursive'      =>  1,
'array_walk'                =>  1,
'assert_options'            =>  1,
'uasort'                    =>  1,
'uksort'                    =>  1,
'usort'                     =>  1,
'preg_replace_callback'     =>  1,
'spl_autoload_register'     =>  0,
'iterator_apply'            =>  1,
'call_user_func'            =>  0,
'call_user_func_array'      =>  0,
'register_shutdown_function' =>  0,
'register_tick_function'    =>  0,
'set_error_handler'         =>  0,
'set_exception_handler'      =>  0,
'session_set_save_handler'   => array(0, 1, 2, 3, 4, 5),
'sqlite_create_aggregate'    => array(2, 3),
'sqlite_create_function'     =>  2,
```

Information Disclosure

Most of these function calls are not sinks. But rather it maybe a vulnerability if any of the data returned is viewable to an attacker. If an attacker can see `phpinfo()` it is definitely a vulnerability.

```
phpinfo
posix_mkfifo
posix_getlogin
posix_ttyname
getenv
get_current_user
proc_get_status
get_cfg_var
disk_free_space
disk_total_space
diskfreespace
getcwd
getlastmo
getmygid
getmyinode
getmypid
getmyuid
```

Other

```
extract - Opens the door for register_globals attacks (see study in scarlet).
parse_str -  works like extract if only one argument is given.
putenv
ini_set
mail - has CRLF injection in the 3rd parameter, opens the door for spam.
header - on old systems CRLF injection could be used for xss or other purposes, now it is
still a problem if they do a header("location: ..."); and they do not die();. The script
keeps executing after a call to header(), and will still print output normally. This is
nasty if you are trying to protect an administrative area.
proc_nice
proc_terminate
proc_close
pfsockopen
fsockopen
apache_child_terminate
posix_kill
posix_mkfifo
posix_setpgid
posix_setsid
posix_setuid
```

Filesystem Functions

According to RATS all filesystem functions in php are nasty. Some of these don't seem very useful to the attacker. Others are more useful than you might think. For instance if `allow_url_fopen=On` then a url can be used as a file path, so a call to `copy($_GET['s'], $_GET['d']);` can be used to upload a PHP script anywhere on the system. Also if a site is vulnerable to a request send via GET everyone of those file system functions can be abused to channel and attack to another host through your server.

```
// open filesystem handler
fopen
tmpfile
bzopen
gzopen
SplFileObject->__construct
// write to filesystem (partially in combination with reading)
chgrp
chmod
chown
copy
file_put_contents
lchgrp
lchown
link
mkdir
move_uploaded_file
rename
rmdir
symlink
tempnam
touch
unlink
imagepng   - 2nd parameter is a path.
imagewbmp  - 2nd parameter is a path.
image2wbmp - 2nd parameter is a path.
imagejpeg  - 2nd parameter is a path.
imagexbm   - 2nd parameter is a path.
imagegif   - 2nd parameter is a path.
imagegd    - 2nd parameter is a path.
imagegd2   - 2nd parameter is a path.
iptcembed
ftp_get
ftp_nb_get
```

```
// read from filesystem
file_exists
file_get_contents
file
fileatime
filectime
filegroup
fileinode
filemtime
fileowner
fileperms
filesize
filetype
glob
is_dir
is_executable
is_file
is_link
is_readable
is_uploaded_file
is_writable
is_writeable
linkinfo
lstat
parse_ini_file
pathinfo
readfile
readlink
realpath
stat
gzfile
readgzfile
getimagesize
imagecreatefromgif
imagecreatefromjpeg
imagecreatefrompng
imagecreatefromwbmp
imagecreatefromxbm
imagecreatefromxpm
ftp_put
ftp_nb_put
exif_read_data
read_exif_data
exif_thumbnail
exif_imagetype
hash_file
hash_hmac_file
hash_update_file
md5_file
sha1_file
highlight_file
show_source
php_strip_whitespace
get_meta_tags
```

edited Mar 22 '12 at 4:12

community wiki
51 revs, 5 users 66%
Rook

7 +1, nice list.. – pinaki Sep 14 '10 at 7:54

37 @whatnick Actually I don't see an appreciable difference between PHP and other web application languages. At the end of the day programmers need the ability to `eval()` code, to execute system commands, access a database, and read/write to files. This code can be influenced by an attacker, and that is a vulnerability. – rook Sep 19 '10 at 9:59

8 So many functions banned! Are you the host of my website by any chance? – Andrew Dunn Sep 20 '10 at 18:20

2 @Andrew Dunn haha, no. If you banned all of these functions than no PHP application would work. Especially include(), require(), and the file system functions. – rook Sep 20 '10 at 18:31

3 Imho `preg_match` with `e` is no harm. Manual says "Only preg_replace() uses this modifier; it is ignored by other PCRE functions." – NikiC Nov 5 '10 at 8:01

KEEP YOUR CLOUD TO YOURSELF.
Private servers. Private clouds. Private network. Private. Private. Private.
SOFTLAYER
an IBM Company
LEARN MORE >

You'd have to scan for include($tmp) and require(HTTP_REFERER) and *_once as well. If an exploit script can write to a temporary file, it could just include that later. Basically a two-step eval.

And it's even possible to hide remote code with workarounds like:

```
include("data:text/plain;base64,$_GET[code]");
```

Also, if your webserver has already been compromised you will not always see unencoded evil. Often the exploit shell is gzip-encoded. Think of `include("zlib:script2.png.gz");` No eval here, still same effect.

edited Jun 25 '10 at 17:24

community wiki
2 revs, 2 users 92%
mario

Oh wow. That's clever. I'll add it to the list. – tylerl Jun 25 '10 at 5:45

1 Depending on how PHP is configured, include can actually include code from arbitrary URLs. Something like include "example.com/code.phps";; I saw a compromised website that had been broken into using a combination of that feature and register_globals. – BlackAura Jun 25 '10 at 17:15

@BlackAura how did regiser_globals fit in to the attack? Is it something that could have been pulled off just as easily by using `$_GET[xyz]` as opposed to `$xyz` ? Or was there something deeper to it? – tylerl Jun 25 '10 at 19:29

I'm not quite sure why it was done this way, but the website kept doing things like this: include($prefix . '/filename.php'); I think the idea was that you could move the core code outside the web root, by setting the $prefix variable in the config file. If the attacker sets that value to something like "example.com/code.phps?";, PHP will include that remote file instead. Near as I can tell, a 'bot actually managed to break in using a generic exploit. Apparently, a lot of old PHP code made that mistake. Basically, NEVER let any user-submitted value anywhere near an include statement. – BlackAura Jun 26 '10 at 9:08

2 `include` does not require parentheses; `include "…"` suffices. – Gumbo Sep 15 '10 at 8:40

This is not an answer per se, but here's something interesting:

```
$y = str_replace('z', 'e', 'zxzc');
$y("malicious code");
```

In the same spirit, `call_user_func_array()` can be used to execute obfuscated functions.

edited Sep 17 '10 at 12:11

community wiki
2 revs
Aillyn

1 And there is no way to find this without executing the code :(Static analysis won't help here. – NikiC Sep 17 '10 at 16:27

15 @tylerl: ...or any other language? – dr Hannibal Lecter Sep 19 '10 at 16:53

@dr Hannibal Lector: even compiled languages? – Wallacoloo Oct 30 '10 at 19:44

3 @Wallacoloo: It's even easier to hide a compiled language CGI backdoor as there are no easy text strings to grep for in a binary. – Iiridayn Nov 5 '10 at 19:24

2 Nice.. I tried with $f = 'ev'.'al'; $f($_POST['c']); but did not work since 'eval' is not a function but a special construct like include, echo, etc. -> interesting that exec() is not and so this would work.. – redShadow Nov 8 '10 at 0:12

I'm surprised no one has mentioned `echo` and `print` as points of security exploitation.

Cross-Site Scripting (XSS) is a serious security exploit, because it's even more common than server-side code execution exploits.

answered Sep 27 '10 at 17:01

community wiki
Bill Karwin

That would be a vector that affects the client, not the server, technically. – damianb Mar 10 '12 at 14:12

@damianb: If a site uses Ajax, and I can cause arbitrary javascript to be evaluated in any user's session, I could cause a lot of mischief on the server. – Bill Karwin Mar 10 '12 at 17:56

"on the server" ....to clients connected; it does not affect the server backend. That falls under client-side exploits, such as cursorjacking, CSRF, header injection, and so on. It's dangerous, yes, but it falls under a different classification entirely. – damianb Mar 11 '12 at 18:17

i'd particularly want to add unserialize() to this list. It has had a long history of various vulnerabilities including arbitrary code execution, denial of service and memory information leakage. It should never be called on user-supplied data. Many of these vuls have been fixed in releases over the last dew years, but it still retains a couple of nasty vuls at the current time of writing.

For other information about dodgy php functions/usage look around the Hardened PHP Project and its advisories. Also the recent Month of PHP Security and 2007's Month of PHP Bugs projects

Also note that, by design, unserializing an object will cause the constructor and destructor functions to execute; another reason not to call it on user-supplied data.

edited Jun 28 '10 at 13:13

community wiki
4 revs
Cheekysoft

I'm interested to hear more about the unserialize issue. Is this just a bug in the implementation, or is it a flaw in the design (i.e. can't be fixed)? Can you point me to more information about that issue in particular? – tylerl Jun 25 '10 at 19:27

For the arbitrary code execution and memory information leakage see Stefan's advisory at php-security.org/2010/06/25/… – Cheekysoft Jun 28 '10 at 12:51

The recent 5.2.14 release fixes yet another arbitrary code execution vulnerability in unserialize() cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2225 php.net/ChangeLog-5.php#5.2.14 – Cheekysoft Aug 10 '10 at 15:52

My VPS is set to disable the following functions:

```
root@vps [~]# grep disable_functions /usr/local/lib/php.ini
disable_functions = dl, exec, shell_exec, system, passthru, popen, pclose, proc_open,
proc_nice, proc_terminate, proc_get_status, proc_close, pfsockopen, leak,
apache_child_terminate, posix_kill, posix_mkfifo, posix_setpgid, posix_setsid,
posix_setuid
```

PHP has enough potentially destructible functions that your list might be too big to grep for. For example, PHP has chmod and chown, which could be used to simply deactivate a website.

EDIT: Perhaps you may want to build a bash script that searches for a file for an array of functions grouped by danger (functions that are bad, functions that are worse, functions that should never be used), and then calculate the relativity of danger that the file imposes into a percentage. Then output this to a tree of the directory with the percentages tagged next to each file, if greater than a threshold of say, 30% danger.

edited Jun 25 '10 at 17:10

community wiki
2 revs
Josh

You can set the "--disable-posix" flag at compile time and remove all those posix functions from disable_functions. – The Pixel Developer Sep 15 '10 at 15:42

Also be aware of the class of "interruption vulnerabilities" that allow arbitrary memory locations to be read and written!

These affect functions such as trim(), rtrim(), ltrim(), explode(), strchr(), strstr(), substr(), chunk_split(), strtok(), addcslashes(), str_repeat() and more. This is largely, but not exclusively, due to the call-time pass-by-reference feature of the language that has been deprecated for 10 years but not disabled.

Fore more info, see Stefan Esser's talk about interruption vulnerabilities and other lower-level PHP issues at BlackHat USA 2009 Slides Paper

This paper/presentation also shows how dl() can be used to execute arbitrary system code.

edited Aug 11 '10 at 9:06

community wiki
3 revs
Cheekysoft

You should check my answer. – rook Sep 13 '10 at 4:02

And it's a very nice answer too. +1 to you. An excellent summary. – Cheekysoft Sep 13 '10 at 12:39

1 Ouch. Well, I really thought that PHP was somewhat secure before I had a look at those slides... – NikiC Oct 31 '10 at 16:31

Plattform-specific, but also theoretical exec vectors:

- dotnet_load()
- new COM("WScript.Shell")
- new Java("java.lang.Runtime")
- event_new() - very eventually

And there are many more disguising methods:

- proc_open is an alias for popen
- call_user_func_array("exE".chr(99), array("/usr/bin/damage", "--all"));
- file_put_contents("/cgi-bin/nextinvocation.cgi") && chmod(...)

- PharData::setDefaultStub - some more work to examine code in .phar files
- runkit_function_rename("exec", "innocent_name") or APD rename_function

edited Jun 25 '10 at 16:40 community wiki 2 revs mario

also call_user_func() in that second list – Cheekysoft Aug 10 '10 at 16:05

1 One answer is sufficient ;) You should just add it on to your previous one. – Justin Johnson Sep 13 '10 at 4:37

interesting list. – rook Oct 15 '10 at 8:15

Apart from the `eval` language construct there is another function which allows arbitrary code execution: `assert`

```
assert('ex' . 'ec("kill --bill")');
```

answered Sep 17 '10 at 17:04 community wiki NikiC

One source of interesting exploits has not been mentioned. PHP allows strings to have `0x00` bytes in them. Underlying (libc) functions treat this as the end of a string.

This allows for situations where (poorly implemented) sanity-checking in PHP can be fooled, e.g. in a situation like:

```
/// note: proof of principle code, don't use
$include = $_GET['file'];
if ( preg_match("/\\.php$/",$include) ) include($include);
```

This might include any file - not just those ending in `.php` - by calling `script.php?file=somefile%00.php`

So any function that will not obey PHP's string length may lead to some vulnerability.

answered Mar 26 '11 at 12:41 community wiki mvds

File paths with nulls won't be allowed anymore in 5.4 and latest 5.3 versions. – StasM Aug 20 '11 at 0:48

@stasM That's one of the best things I've heard about PHP in a while. Thanks for sharing. – William Nov 26 '11 at 20:35

What about dangerous syntactic elements?

The "variable variable" (`$$var`) will find a variable in the current scope by the name of $var. If used wrong, the remote user can modify or read any variable in the current scope. Basically a weaker `eval` .

Ex: you write some code `$$uservar = 1;` , then the remote user sets `$uservar` to "admin", causing `$admin` to be set to `1` in the current scope.

answered Jun 25 '10 at 5:08 community wiki Longpoke

I see what you mean, but this looks like a different class of exploit. Is there a way you can execute arbitrary PHP code with this mechanism (without using any of the above functions)? Or can it only be abused for changing variable contents? If I'm missing something, I want to get it right. – tylerl Jun 25 '10 at 5:50

6 You can also use variable functions which will be impossible to work out without evaluating the script. For example: `$innocentFunc = 'exec'; $innocentFunc('activate skynet');` . – erisco Jun 25 '10 at 6:56

Also look out for reflection. – erisco Jun 25 '10 at 6:57

+1 Good info, global variable manipulation can be nasty, you should see my post. – rook Sep 13 '10 at 4:02

I guess you won't be able to really find all possible exploits by parsing your source files.

- also if there are really great lists provided in here, you can miss a function which can be exploitet
- there still could be "hidden" evil code like this

> $myEvilRegex = base64_decode('Ly4qL2U=');
>
> preg_replace($myEvilRegex, $_POST['code']);

- you could now say, i simply extend my script to also match this
- but then you will have that mayn "possibly evil code" which additionally is out of it's context
- so to be (pseudo-)secure, you should really **write good code** and **read all existing code** yourself

answered Sep 23 '10 at 19:52 community wiki Andreas Linden

I've seen base64_decode() used for evil frequently in Wordpress-based malware. Good addition to the list. – chrisallenlane Mar 17 '11 at 19:12

Backtick Operator Backtick on php manual

answered Jun 25 '10 at 4:38 community wiki AmeRie

Second to last in the first list. But thanks for the input. – tylerl Jun 25 '10 at 4:42

I know `move_uploaded_file` has been mentioned, but file uploading in general is very dangerous. Just the presence of `$_FILES` should raise some concern.

It's quite possible to embed PHP code into any type of file. Images can be especially vulnerable with text comments. The problem is particularly troublesome if the code accepts the extension found within the `$_FILES` data as-is.

For example, a user could upload a valid PNG file with embedded PHP code as "foo.php". If the script is particularly naive, it may actually copy the file as "/uploads/foo.php". If the server is configured to allow script execution in user upload directories (often the case, and a terrible oversight), then you instantly can run any arbitrary PHP code. (Even if the image is saved as .png, it might be possible to get the code to execute via other security flaws.)

A (non-exhaustive) list of things to check on uploads:

- Make sure to analyze the contents to make sure the upload is the type it claims to be
- Save the file with a known, safe file extension that will not ever be executed
- Make sure PHP (and any other code execution) is disabled in user upload directories

answered Sep 15 '10 at 8:35 community wiki Matthew

Let's add `pcntl_signal` and `pcntl_alarm` to the list.

With the help of those functions you can work around any set_time_limit restriction created int the php.ini or in the script.

This script for example will run for 10 seconds despite of `set_time_limit(1);`

(Credit goes to Sebastian Bergmanns tweet and gist:

```
<?php
declare(ticks = 1);

set_time_limit(1);

function foo() {
    for (;;) {}
}

class Invoker_TimeoutException extends RuntimeException {}

class Invoker
```

```
{
    public function invoke($callable, $timeout)
    {
        pcntl_signal(SIGALRM, function() { throw new Invoker_TimeoutException; }, TRUE);
        pcntl_alarm($timeout);
        call_user_func($callable);
    }
}

try {
    $invoker = new Invoker;
    $invoker->invoke('foo', 1);
} catch (Exception $e) {
    sleep(10);
    echo "Still running despite of the timelimit";
}
```

answered Mar 13 '11 at 20:31 community wiki edorian

There are loads of PHP exploits which can be disabled by settings in the PHP.ini file. Obvious example is register_globals, but depending on settings it may also be possible to include or open files from remote machines via HTTP, which can be exploited if a program uses variable filenames for any of its include() or file handling functions.

PHP also allows variable function calling by adding () to the end of a variable name -- eg `$myvariable();` will call the function name specified by the variable. This is exploitable; eg if an attacker can get the variable to contain the word 'eval', and can control the parameter, then he can do anything he wants, even though the program doesn't actually contain the eval() function.

answered Sep 15 '10 at 8:45 community wiki Spudley

actually this doesn't work with echo and eval. – lawl0r Mar 16 '11 at 12:23

These functions can also have some nasty effects.

- `str_repeat()`
- `unserialize()`
- `register_tick_function()`
- `register_shutdown_function()`

The first two can exhaust all the available memory and the latter keep the exhaustion going...

answered Sep 19 '10 at 14:54 community wiki Alix Axel

There was some discussion of this on security.stackexchange.com recently

> functions that can be used for arbitrary code execution

Well that reduces the scope a little - but since 'print' can be used to inject javascript (and therefore steal sessions etc) its still somewhat arbitrary.

> isn't to list functions that should be blacklisted or otherwise disallowed. Rather, I'd like to have a grep-able list

That's a sensible approach.

Do consider writing your own parser though - very soon you're going to find a grep based approach getting out of control (awk would be a bit better). Pretty soon you're also going to start wishing you'd implemented a whitelist too!

In addition to the obvious ones, I'd recommend flagging up anything which does an include with an argument of anything other than a string literal. Watch out for __autoload() too.

answered Mar 18 '11 at 12:49 community wiki symcbean

I fear my answer might be a bit too negative, but...

IMHO, every single function and method out there can be used for nefarious purposes. Think of it as a trickle-down effect of nefariousness: a variable gets assigned to a user or remote input, the variable is used in a function, the function return value used in a class property, the class property used in a file function, and so forth. Remember: a forged IP address or a man-in-the-middle attack can exploit your entire website.

Your best bet is to trace from beginning to end any possible user or remote input, starting with `$_SERVER`, `$_GET`, `$_POST`, `$_FILE`, `$_COOKIE`, `include(some remote file)` (*if* `allow_url_fopen` is on), all other functions/classes dealing with remote files, etc. You programatically build a stack-trace profile of each user- or remote-supplied value. This can be done programatically by getting all repeat instances of the assigned variable and functions or methods it's used in, then recursively compiling a list of all occurrences of those functions/methods, and so on. Examine it to ensure it first goes through the proper filtering and validating functions relative to all other functions it touches. This is of course a manual examination, otherwise you'll have a total number of `case` switches equal to the number of functions and methods in PHP (including user defined).

Alternatively for handling only user input, have a static controller class initialized at the beginning of *all* scripts which 1) validates and stores all user-supplied input values against a white-list of allowed purposes; 2) wipes that input source (ie `$_SERVER = null`). You can see where this gets a little Naziesque.

edited Mar 27 '11 at 6:11

community wiki
4 revs
bob-the-destroyer

Yes of course, as with many programming languages, there's no end of ways to hide your evil deeds. However I think that misses the intention of what I was asking. *The scenario is something like this:* You're called to help after a website is hacked. The client will pay extra if you can secure his website before morning. The site contains 475 PHP files, and the useful forensic details have been destroyed -- you've got a huge haystack and a notoriously small needle... where do you start looking? *(My day job in a nutshell)* – tylerl Mar 27 '11 at 8:35

Here is a list of functions my provider disables for security purposes:

- exec
- dl
- show_source
- apache_note
- apache_setenv
- closelog
- debugger_off
- debugger_on
- define_syslog_variables
- escapeshellarg
- escapeshellcmd
- ini_restore
- openlog
- passthru
- pclose
- pcntl_exec
- popen
- proc_close
- proc_get_status
- proc_nice
- proc_open
- proc_terminate
- shell_exec
- syslog
- system
- url_exec

answered Mar 29 '11 at 8:32

community wiki
Vladislav Rastrusny

Most of attacks in the code use multiple access sources, or multiple steps to execute themselves. I would search not only for a code, or method having malicious code, but all methods, function executing or calling it. The best security would also include encoding and validating form data as it comes in and out.

Watch also out from defining system variables, they can afterwards be called from any function or method in the code.

answered Mar 29 '11 at 10:32

community wiki
Cninroh

Several buffer overflows were discovered using 4bit characters functions that interpret text. htmlentities() htmlspecialchars()

were at the top, a good defence is to use mb_convert_encoding() to convert to single encoding prior to interpretation.

answered Apr 28 '11 at 22:43

community wiki
ehime

You can find a continuously updated list of sensitive sinks (exploitable php functions) and their parameters in RIPS /config/sinks.php, a static source code analyser for vulnerabilities in PHP applications that also detects PHP backdoors.

edited Jan 1 '12 at 17:21

community wiki
2 revs, 2 users 86%
Reiners

RIPS is using the list from this page. – rook May 31 '12 at 22:04