Introduction

This paper is on the topic of vulnerability CVE-2013-7226. The problem lies in the fact that PHP incorrectly handled certain values when using the imagecrop() function in ext/GD/gd.c PHP 5.5.x (versions prior to 5.5.9) which allows remote attackers to cause a denial of service or even an unspecified other impact through an imagecrop() function call with a large x dimension value, which would lead to a heap-based buffer overflow. A buffer overflow is an anomaly where a program, while writing data to a buffer, overruns the buffer's boundary and overwrites adjacent memory. This is a special case of violation of memory safety.

The vulnerability

The imagecrop() function can be used to crop an image. The implementation of imagecrop() in ext/gd/gd.c performs very little checking of the supplied dimensions. That means if the dimension's array contains “x”, it's value is treated like it is integer, even if it's not. This can be used as an information leak vulnerability, because strings and array contain pointers which can be used for subsequent exploits: allows access to unauthorized disclosure of information, allows unauthorized modification, allows disruption of the service.

GdImageCreateTrueColor () and gdImageCreate () functions are smart enough to block all attempts to cause overflow using the width and height parameters, returns a NULL pointer when it happens. However, this code does not check the return value of these functions, using the first variable's memory writes unconditionally. This means it can cause a NULL pointer (or a close-to-NULL pointer) write, which is probably just crash the process, but since the gd image structure is very large, it could even touch the isolated memory.

The function then performs some bounds checks:

```c
if (src->sx < (crop->x + crop->width -1)) {
    crop->width = src->sx - crop->x + 1;
}
if (src->sy < (crop->y + crop->height -1)) {
    crop->height = src->sy - crop->y + 1;
}
```

These are using signed integer arithmetic's and can be overflowed and tricked into incorrect calculations. The crop->x and crop->y are completely user-supplied values and we can supply negative values. This way we can force the copying code to read outside of the source image pixel data, causing a crash or an information leak.

We must however keep the crop->width and crop->height positive, and reasonable, because they are used at the beginning of the function to create a destination bitmap. We can however trick the bounds checking code:

```c
if (src->sx < (crop->x + crop->width -1)) {
    crop->width = src->sx - crop->x + 1;
}
```

When supplying a very large crop->x value, we can make the condition pass, assigning a value to crop->width which is larger than the real destination's pixel buffer width. The memcpy will then copy more data than the heap-based buffers can hold, causing a heap-based buffer overflow.

The Solution

The vulnerability's impact is only partial because the modification of some system files or information is possible, but the attacker does not have control over what can be modified, or the scope of what the attacker can affect is limited.

The patch used:
@@ -4987,6 +4987,10 @@ PHP_FUNCTION(imagecrop)
 } else {
     rect.x = Z_LVAL_PP(tmp);
     if (rect.x < 0) {
+            php_error_doref(NULL TSRMLS_CC, E_WARNING, "Negative x position");  
+            RETURN_FALSE;
+    }
    } else {
        php_error_doref(NULL TSRMLS_CC, E_WARNING, "Missing x position");
        RETURN_FALSE;
@@ -5002,6 +5006,10 @@ PHP_FUNCTION(imagecrop)
 } else {
     rect.y = Z_LVAL_PP(tmp);
     if (rect.y < 0) {
+            php_error_doref(NULL TSRMLS_CC, E_WARNING, "Negative y position");  
+            RETURN_FALSE;
+    }
    } else {
        php_error_doref(NULL TSRMLS_CC, E_WARNING, "Missing y position");
        RETURN_FALSE;
@@ -5017,6 +5025,10 @@ PHP_FUNCTION(imagecrop)
 } else {
     rect.width = Z_LVAL_PP(tmp);
     if (rect.width < 0) {
+            php_error_doref(NULL TSRMLS_CC, E_WARNING, "Negative width");  
+            RETURN_FALSE;
+    }
    } else {
        php_error_doref(NULL TSRMLS_CC, E_WARNING, "Missing width");
        RETURN_FALSE;
@@ -5032,11 +5044,20 @@ PHP_FUNCTION(imagecrop)
 } else {
     rect.height = Z_LVAL_PP(tmp);
     if (rect.height < 0) {
+            php_error_doref(NULL TSRMLS_CC, E_WARNING, "Negative height");  
+            RETURN_FALSE;
+    }
    } else {
        php_error_doref(NULL TSRMLS_CC, E_WARNING, "Missing height");
        RETURN_FALSE;

    if (rect.x >= gdImageSX(im) || rect.y >= gdImageSY(im)) {
        php_error_doref(NULL TSRMLS_CC, E_WARNING, "Position exceeded");
        RETURN_FALSE;
    } else {
        im_crop = gdImageCrop(im, &rect);
        if (im_crop == NULL) {
            diff --git a/ext/gd/gd.c b/ext/gd/gd.c
            index 49970c1..0791809 100644
--- a/ext/gd/gd.c
+++ b/ext/gd/gd.c
@@ -46,9 +46,15 @@ gdImagePtr gdImageCrop(gdImagePtr src, const gdRectPtr crop)
if (src->trueColor) {
    dst = gdImageCreateTrueColor(crop->width, crop->height);
    if (dst == NULL) {
        return NULL;
    }
    gdImageSaveAlpha(dst, 1);
} else {
    dst = gdImageCreate(crop->width, crop->height);
    if (dst == NULL) {
        return NULL;
    }
    gdImagePaletteCopy(dst, src);
}
dst->transparent = src->transparent;

Sources:

https://bugs.php.net/bug.php?id=66356
https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-7226
http://git.php.net/?p=php-src.git;a=commit;h=8f4a5373bb71590352fd934028d6dde5bc18530b