CYBSEC S.A.
www.cybsec.com

Advisory Name: PHPMailer Infinite Loop Denial of Service

Vulnerability Class: Denial of Service

Release Date: 05.27.2005

Affected Applications:
- PHPMailer <= 1.72

Affected Platforms:
- Platform-Independent: Tested on Apache 2.0.52 / PHP 4.3.11 & PHP 5.0.4

Local / Remote: Remote

Severity: High

Author: Mariano Nuñez Di Croce

Vendor Status: Notified. No patch available.

Reference to Vulnerability Disclosure Policy:
http://www.cybsec.com/vulnerability_policy.pdf

Overview:

PHPMailer is a PHP class that supports the creation of HTML-based e-mails, attachments, multiple TOs, CCs, BCCs and REPLY-TOs, SMTP authentication, etc.

According to the developer, this class has been implemented in the following Projects: eGroupWare, Mambo Open Source, PostNuke, MyPHPNuke, Mantis, Moodle, XOOPS, Sourdough, Open Source Suite CRM, Xaraya, Ciao EmailList Manage, Owl Intranet Knowledgebase, pLiMa (php List Manager), phplist, Octeth Email Manager Pro, phpwebtools, sendcard and has more than 100,000 downloads.

A vulnerability has been discovered in PHPMailer that allows an attacker to raise CPU and memory use to 100% in approximately 20 seconds.

Vulnerability Description:

The vulnerability specifically exists in the handling of long mail headers. A header field >= 998 characters (without blanks) will make the Data() function enter an infinite loop in which new memory keeps being requested.

The problem exists within the SMTP-Class Data() function defined in *class.smtp.php*. Below is the respective code fragment:

```
      ...
      ...
      function Data($msg_data) {
            ...
            ...

            $field = substr($lines[0],0,strpos($lines[0],":"));
            $in_headers = false;
            if(!empty($field) && !strstr($field," ")) {
    (1)         $in_headers = true;
            }

    (2)    $max_line_length = 998; # used below; set here for ease in change

            while(list(,$line) = @each($lines)) {
                $lines_out = null;
                if($line == "" && $in_headers) {
                    $in_headers = false;
                }
                # ok we need to break this line up into several
                # smaller lines
                while(strlen($line) > $max_line_length) {
                    $pos = strrpos(substr($line,0,$max_line_length)," ");
    (3)             $lines_out[] = substr($line,0,$pos);
                    $line = substr($line,$pos + 1);
                    # if we are processing headers we need to
                    # add a LWSP-char to the front of the new line
                    # rfc 822 on long msg headers
                    if($in_headers) {
    (4)                 $line = "\t" . $line;
                    }
                }
                $lines_out[] = $line;

                # now send the lines to the server
           ...
           ...
```

When processing the headers, *$mail_headers* is set to "true" in (1). Next, line length is limited to 998 characters long in (2). Then, a substring of *$line*, from 0 to the value set in *$pos*, is added to the *$lines_out* array in (3). Finally, in (4), if it is processing a header field, an '\t' is added to the front of *$line* string.

Now suppose a header field like this is submitted:
From: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA... x 998

In the first loop, it will take out the "From:" substring. In the next iteration, *$pos* will be set to "", because there are no blanks left in the string. Therefore, a null string will be added to the *$lines_out* array. In the following instruction, *$line* will be shortened one character, because it is assigned a substring of itself from the first character. The problem is that in (4), one new character is added, so *$line* will keep being of the same length (> *$max_line_length*) and the loop will become endless.

The processing of a malformed request like this one will make the server rapidly starve memory and processor resources, turning it truly unstable (probably denying access to other services as well) and a reboot may be needed to reestablish normal functioning.

Solutions:

Probably, the quickest workaround is to limit the length of the strings received before appending them to the class variables.

Anyway, a possible workaround is presented below:

```
...
...
function Data($msg_data) {
        ...
        ...

        $field = substr($lines[0],0,strpos($lines[0],":"));
        $in_headers = false;
        if(!empty($field) && !strstr($field," ")) {
            $in_headers = true;
        }

       $max_line_length = 998; # used below; set here for ease in change

        while(list(,$line) = @each($lines)) {
            $lines_out = null;
            if($line == "" && $in_headers) {
                $in_headers = false;
            }
            # ok we need to break this line up into several
            # smaller lines
            while(strlen($line) > $max_line_length) {
                $pos = strrpos(substr($line,0,$max_line_length)," ");

                #-------------- fix --------------------------
                     if (!$pos) {
                           $pos = $max_line_length - 1;
                     }
                #----------  end of fix ---------------------

                $lines_out[] = substr($line,0,$pos);
                $line = substr($line,$pos + 1);
                # if we are processing headers we need to
                # add a LWSP-char to the front of the new line
                # rfc 822 on long msg headers
                if($in_headers) {
                 $line = "\t" . $line;
```

```
                    }
                }
                $lines_out[] = $line;

                # now send the lines to the server
        ...
```

Vendor Response:

Vendor confirmed the vulnerability 2 days after the first contact and stated that a fix will be available in the next version. He didn't reply to any further contact or provide information about the project state.

04.19.2005 – Vendor Notified.
04.21.2005 – Vendor Confirmed Vulnerability.
05.09.2005 – Vendor Contacted – No Reply.
05.18.2005 – Vendor Contacted – No Reply.
05.27.2005 – Vulnerability Public Disclosure

Special Thanks: Leonardo Cammareri, Leandro Meiners, Juan Pablo Perez Etchegoyen.

Contact Information:

For more information regarding the vulnerability feel free to contact the author at mnunez {at} cybsec.com.

For more information regarding CYBSEC: www.cybsec.com