ALL THINGS IN MODERATION

PHP STRING COMPARISON VULNERABILITIES

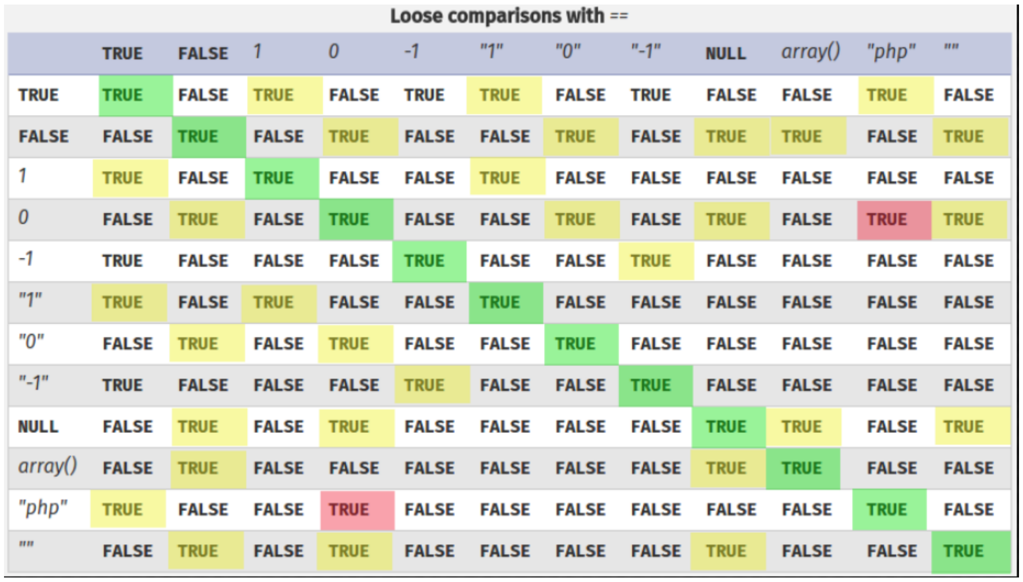
By groot | May 5, 2017 | Network Security | 1 Comment

1. BYPASS PHP '==' AND '!=' COMPARISON OPERATORS

'==' and '!=' is the default comparison in other languages. But in PHP has two main comparison modes, lets call them loose ('==' and '!=')
and strict ('===' and '!==').

Consider 2 following table to see the strict in '===' and the loose in '==':

PHP Comparisons: Loose



Loose comparisons with ==

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

FIND US ON FACEBOOK

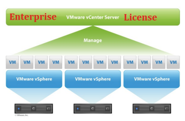
Hydrasky
585 likes

Like Page

Be the first of your friends to like

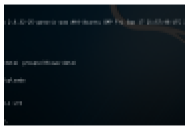
Search the site

POPULAR | RECENT



Activatio key for vCenter, vSphere 6.5

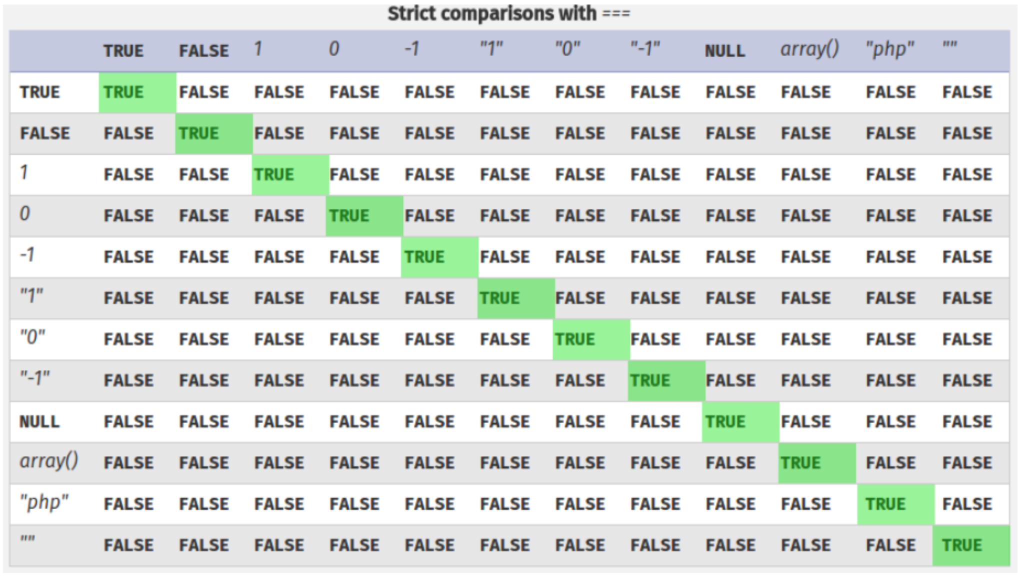
February 23, 2018



create a web backdoo payload with metasplo

January 18, 2017

PHP Comparisons: Strict


Strict comparisons with ===

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
1	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
-1	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"1"	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
array()	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
"php"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

So PHP will do type juggling before compare using '==' operator (http://php.net/manual/en/language.operators.comparison.php)

Type juggling means If PHP decides that both operands look like numbers, even if they are actually strings, it will convert them both and perform a numeric comparison:

- TRUE: "0e12345" == "0e54321"
- TRUE: "0e12345" <= "1"
- TRUE: "0e12345" == "0"
- TRUE: "0xF" == "15"

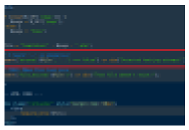
Or if we comparing a string to a number, PHP will attempt to convert the string to a number then perform a numeric comparison:

- TRUE: "0000" == int(0)
- TRUE: "0e12" == int(0)
- TRUE: "1abc" == int(1)
- TRUE: "0abc" == int(0)
- TRUE: "abc" == int(0)

FOR EXAMPLE BYPASS AUTHENTICATION USING '==' AND '!=' COMPARISON OPERATORS

Consider following PHP code handling to check for valid user's token.

```
<php
        $token = "0e124656823434657657655654324342";
```

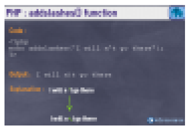


PHP assert() Vulnerat to Local File Inclusior

July 1, 2017

sqlite injection

Septembe 5, 2016



SQL injection bypass PHP addslash function

August 23, 2016

Next »

ARCHIVES

- October 2018
- September 2018
- July 2018
- June 2018
- May 2018
- April 2018
- March 2018
- February 2018
- January 2018
- December 2017
- November 2017
- October 2017
- September 2017
- August 2017
- July 2017
- June 2017
- May 2017
- April 2017

```
        if(isset($_COOKIE['token']) && $_COOKIE['token'] ==
                // access to privilege area
        }
        else {
                // login require
        }
?>
```

or

```
<php
        $token = "0e124656823434657657655654324342";
        if(isset($_COOKIE['token']) && $_COOKIE['token'] !=
                 // login require
        }
        else {
                // access to privilege area
        }
?>
```

So If we input to application $COOKIE['token']='0'. What happen next?

$COOKIE['token'] == $token ('0e124656823434657657655654324342' == '0') will return TRUE
$COOKIE['token'] != $token ('0e124656823434657657655654324342' != '0') will return FALSE

=> authentication passed.

RECOMMENDATIONS

Use === as your default comparison. Only reach for == if you really need it

2. BYPASS PHP STRCMP() FUNCTION

strcmp is a function created to compare strings.

int strcmp(string $str1, string $str2);

- Return <0 if $str1 < $str2
- Return 0 if $str1 === $str2
- Return &Gt;0 if $str1 > $str2

For example: Consider following PHP code handling to check for valid user's token.

- March 2017
- February 2017
- January 2017
- December 2016
- November 2016
- October 2016
- September 2016
- August 2016
- July 2016
- June 2016
- May 2016
- April 2016

BLOG AUTHORS

Stephen Stinson
Network Security and Malware Analysis

groot
Network Security and Enterprise Network Design

Cloudi
Network Security and Mobile Malware Analysis

```
<php
        $token = "0e37bd1f669d8bb5eae47ef80013e4d3d8287c11'
        if(isset($_COOKIE['token']) && strcmp($_COOKIE['tol
                // access to privilege area
        }
        else {
                // login require
        }
?>
```

If we request with cookie token is an array to pass an array instead of a string to strcmp(), it will gives a warning ('WARNING strcmp() expects parameter 2 to be string, array given on line number …!') but the compare result return 0.

This request look like:

```
GET / HTTP 1.1
Host: example.com
Cookie: token[]=''
.....
```

=> $_COOKIE['token'] = array(0 => "");

strcmp(array(0 => ""), "0a37bd1f669d8bb5eae47ef80013e4d3d8287c11") will return 0.

=> authentication passed.

RECOMMENDATIONS

Don't use this function to compare 2 variables which you don't know types.

Perform type conversions to string using the cast (string) before put into strcmp().

REFERENCES

https://www.owasp.org/images/6/6b/PHPMagicTricks-TypeJuggling.pdf

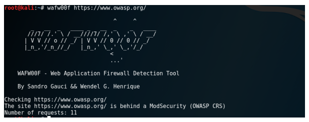
Tags: bug, php security, string comparison

RELATED POSTS

Andreas Griffin
IoT Security and Reverse Engineering

Luke Walker

Win Stark
Network Security and Enterpise Network Design



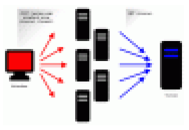
```
root@kali:~# wafw00f https://www.owasp.org/

                                 ^     ^
        _   __  _   ____ _   __  _    _   ____
       ///7/ /.' \ / __////7/ /,' \ ,' \ / __/
      | V V // o // _/ | V V // 0 // 0 // _/
      |_n_,'/_n_//_/   |_n_,' \_,' \_,'/_/
                                <
                                 ...'

    WAFW00F - Web Application Firewall Detection Tool

    By Sandro Gauci && Wendel G. Henrique

Checking https://www.owasp.org/
The site https://www.owasp.org/ is behind a ModSecurity (OWASP CRS)
Number of requests: 11
```



XML-RPC DDOS

By Cloudi

API security checklist

By Win Stark

ABOUT THE AUTHOR

groot

Network Security and Enterpise Network Design

ONE RESPONSE

Frederic AUGUST 22, 2017

I agree: don't use loose comparison.
But, your are wrong in your "vulnerability" sample.
Here an 3v4l:
https://3v4l.org/VqDnL

LEAVE A REPLY

Comment Text*

Name*

Email*

Website

I'm not a robot

reCAPTCHA
Privacy - Terms

POST COMMENT