```
wp_verify_nonce( string $nonce,
string|int $action = -1 ): int|false
```

In this article

Verifies that a correct security nonce was used with time limit.

Description

A nonce is valid for between 12 and 24 hours (by default).

Parameters

`$nonce` `string` required

Nonce value that was used for verification, usually via a form field.

`$action` `string|int` optional

Should give context to what is taking place and be the same when nonce was created.

Default: `-1`

Return

`int|false` 1 if the nonce is valid and generated between 0-12 hours ago, 2 if the nonce is valid and generated between 12-24 hours ago.
False if the nonce is invalid.

More Information

The function is used to verify the nonce sent in the current request usually accessed by the `$_REQUEST` PHP variable.

Nonces should never be relied on for authentication or authorization, access control. Protect your functions using `current_user_can()`, always assume Nonces can be compromised.

Source

```
wp-includes/pluggable.php
```

```php
function wp_verify_nonce( $nonce, $action = -1 ) {
    $nonce = (string) $nonce;
    $user  = wp_get_current_user();
    $uid   = (int) $user->ID;
    if ( ! $uid ) {
        /**
         * Filters whether the user who generated the nonce is logg
         *
         * @since 3.5.0
         *
         * @param int        $uid    ID of the nonce-owning user.
         * @param string|int $action The nonce action, or -1 if not
         */
        $uid = apply_filters( 'nonce_user_logged_out', $uid, $actio
    }

    if ( empty( $nonce ) ) {
        return false;
    }

    $token = wp_get_session_token();
    $i     = wp_nonce_tick( $action );

    // Nonce generated 0-12 hours ago.
    $expected = substr( wp_hash( $i . '|' . $action . '|' . $uid .
    if ( hash_equals( $expected, $nonce ) ) {
        return 1;
    }

    // Nonce generated 12-24 hours ago.
    $expected = substr( wp_hash( ( $i - 1 ) . '|' . $action . '|'
    if ( hash_equals( $expected, $nonce ) ) {
        return 2;
    }

```

```
2404      /**
2405       * Fires when nonce verification fails.
2406       *
2407       * @since 4.4.0
2408       *
2409       * @param string     $nonce  The invalid nonce.
2410       * @param string|int $action The nonce action.
2411       * @param WP_User    $user   The current user object.
2412       * @param string     $token  The user's session token.
2413       */
2414      do_action( 'wp_verify_nonce_failed', $nonce, $action, $user, $
2415
2416      // Invalid nonce.
2417      return false;
2418  }
```

View all references · View on Trac · View on GitHub

Hooks

apply_filters(‘nonce_user_logged_out’, int $uid, string|int $action)

Filters whether the user who generated the nonce is logged out.

do_action(‘wp_verify_nonce_failed’, string $nonce, string|int $action, WP_User $user, string $token)

Fires when nonce verification fails.

Related

Uses	Description
wp_get_session_token() wp-includes/user.php	Retrieves the current session token from the logged_in cookie.

Uses	Description
wp_nonce_tick() wp-includes/pluggable.php	Returns the time-dependent variable for nonce creation.
wp_hash() wp-includes/pluggable.php	Gets the hash of the given string.
wp_get_current_user() wp-includes/pluggable.php	Retrieves the current user object.

Show 2 more

Used by	Description
WP_Recovery_Mode::handle_exit_recovery_mode() wp-includes/class-wp-recovery-mode.php	Handles a request to exit Recovery Mode.
wp_edit_theme_plugin_file() wp-admin/includes/file.php	Attempts to edit a file for a theme or plugin.
rest_cookie_check_errors() wp-includes/rest-api.php	Checks for errors when using cookie-based authentication.
wp_handle_comment_submission() wp-includes/comment.php	Handles the submission of a comment, usually posted to wp-comments-post.php via a comment form.
wp_ajax_destroy_sessions() wp-admin/includes/ajax-actions.php	Handles destroying multiple open sessions for a user via AJAX.

Show 10 more

Changelog

Version	Description
2.0.3	Introduced.

User Contributed Notes

Codex 11 years ago ▲ 4 ▼

Example

Verify an nonce created with `wp_create_nonce()` :

```
<?php
// Step A: Create an nonce, and add it as a query var in a link
$nonce = wp_create_nonce( 'my-nonce' );
echo "<a href='myplugin.php?_wpnonce={$nonce}'>" . __( 'Save So
?>
```

```
// Step B: In our file that handles the request, verify the non
$nonce = $_REQUEST['_wpnonce'];
if ( ! wp_verify_nonce( $nonce, 'my-nonce' ) ) {
    die( __( 'Security check', 'textdomain' ) );
} else {
    // Do stuff here.
}
```

You may also decide to take different actions based on the age of the nonce:

```
$nonce = wp_verify_nonce( $nonce, 'my-nonce' );
switch ( $nonce ) {
    case 1:
        echo 'Nonce is less than 12 hours old';
        break;
    case 2:
        echo 'Nonce is between 12 and 24 hours old';
        break;
    default:
        exit( 'Nonce is invalid' );
}
```

Log in to add feedback

WordProof 5 years ago 3

Per the WordPress Coding Standards, this is the way as to verify a nonce:

```
if ( isset( $_REQUEST['_wpnonce'] ) && wp_verify_nonce( $_REQUE

  //do you action

} else {

  die( __( 'Security check', 'textdomain' ) );

}
```

Show feedback (1) · Log in to add feedback

Roy Orbitson 5 years ago -1

This system is not very accurate and the comment about the return values is incorrect. The value `1` means < 12 hours old, but `2` means anywhere from 1 second to < 24 hours old.

Observe what happens to the nonce tick value over a day:

local time ~~~ seconds since epoch ~~~ tick
2021-05-20T00:00:00+03:00 ~~~ 1621458000 ~~~ 37534
2021-05-20T01:00:00+03:00 ~~~ 1621461600 ~~~ 37534
2021-05-20T02:00:00+03:00 ~~~ 1621465200 ~~~ 37534
2021-05-20T03:00:00+03:00 ~~~ 1621468800 ~~~ 37534
2021-05-20T04:00:00+03:00 ~~~ 1621472400 ~~~ 37535
2021-05-20T05:00:00+03:00 ~~~ 1621476000 ~~~ 37535
2021-05-20T06:00:00+03:00 ~~~ 1621479600 ~~~ 37535
2021-05-20T07:00:00+03:00 ~~~ 1621483200 ~~~ 37535
2021-05-20T08:00:00+03:00 ~~~ 1621486800 ~~~ 37535
2021-05-20T09:00:00+03:00 ~~~ 1621490400 ~~~ 37535
2021-05-20T10:00:00+03:00 ~~~ 1621494000 ~~~ 37535
2021-05-20T11:00:00+03:00 ~~~ 1621497600 ~~~ 37535
2021-05-20T12:00:00+03:00 ~~~ 1621501200 ~~~ 37535
2021-05-20T13:00:00+03:00 ~~~ 1621504800 ~~~ 37535
2021-05-20T14:00:00+03:00 ~~~ 1621508400 ~~~ 37535
2021-05-20T15:00:00+03:00 ~~~ 1621512000 ~~~ 37535
2021-05-20T16:00:00+03:00 ~~~ 1621515600 ~~~ 37536
2021-05-20T17:00:00+03:00 ~~~ 1621519200 ~~~ 37536
2021-05-20T18:00:00+03:00 ~~~ 1621522800 ~~~ 37536
2021-05-20T19:00:00+03:00 ~~~ 1621526400 ~~~ 37536
2021-05-20T20:00:00+03:00 ~~~ 1621530000 ~~~ 37536
2021-05-20T21:00:00+03:00 ~~~ 1621533600 ~~~ 37536
2021-05-20T22:00:00+03:00 ~~~ 1621537200 ~~~ 37536
2021-05-20T23:00:00+03:00 ~~~ 1621540800 ~~~ 37536

...and over the boundary of a tick:

local time ~~~ seconds since epoch ~~~ tick
2021-05-20T14:59:58+03:00 ~~~ 1621511998 ~~~ 37535
2021-05-20T14:59:59+03:00 ~~~ 1621511999 ~~~ 37535
2021-05-20T15:00:00+03:00 ~~~ 1621512000 ~~~ 37535
2021-05-20T15:00:01+03:00 ~~~ 1621512001 ~~~ 37536
2021-05-20T15:00:02+03:00 ~~~ 1621512002 ~~~ 37536

In this example, you can see that a nonce generated at 3pm and verified one second later will return `2` because of the tick change. The ticks do not align with timezones due to their basis in universal time, so nonces will always appear “old” to your code at certain times of the day.

Show feedback (1) · Log in to add feedback