Escaping Data

- Escaping Functions
- Custom Escaping Example
- Always escape late
- … Except when you can’t
- Escaping with Localization
- Examples
  - Escaping any numeric variable used anywhere
  - Escaping arbitrary variable within HTML attribute
  - Escaping arbitrary URL within HTML attribute, but also in other contexts
  - Passing an arbitrary variable to JavaScript via wp_localize_script()
  - Escaping arbitrary variable within JavaScript block
  - Escaping arbitrary variable within inline JavaScript
  - Escaping arbitrary variable within HTML attribute for use by JavaScript
  - Escaping arbitrary string within HTML textarea
  - Escaping arbitrary string within HTML tags
  - Escaping arbitrary string within XML or XSL context

Escaping output is the process of securing output data by stripping out unwanted data, like malformed HTML or script tags. This process helps secure your data prior to rendering it for the end user.

Escaping Functions

WordPress has many helper functions you can use for most common scenarios.

Pay close attention to what each function does, as some will remove HTML while others will permit it. You must use the most appropriate function to the content and context of what you’re echoing. You always want to escape when you echo, not before.

- `esc_html()` – Use anytime an HTML element encloses a section of data being displayed. This will remove HTML.

```
<h4><?php echo esc_html( $title ); ?></h4>
```

- `esc_js()` – Use for inline Javascript.

```
<div onclick='<?php echo esc_js( $value ); ?>' />
```

- `esc_url()` – Use on all URLs, including those in the src and href attributes of an HTML element.

```
<img alt="" src="<?php echo esc_url( $media_url ); ?>" />
```

- `esc_url_raw()` – Use when storing a URL in the database or in other cases where non-encoded URLs are needed.
- `esc_xml()` – Use to escape XML block.
- `esc_attr()` – Use on everything else that's printed into an HTML element's attribute.

```
<ul class="<?php echo esc_attr( $stored_class ); ?>">
```

- `esc_textarea()` – Use this to encode text for use inside a textarea element.
- `wp_kses()` – Use to safely escape for all non-trusted HTML (post text, comment text, etc.). This preserves HTML.
- `wp_kses_post()` – Alternative version of `wp_kses()` that automatically allows all HTML that is permitted in post content.
- `wp_kses_data()` – Alternative version of `wp_kses()` that allows only the HTML permitted in post comments.

Custom Escaping Example

In the case that you need to escape your output in a specific way, the function wp_kses() (pronounced "kisses") will come in handy.

This function makes sure that only the specified HTML elements, attributes, and attribute values will occur in your output, and normalizes HTML entities.

```
<?php
echo wp_kses_post( $partial_html );
echo wp_kses(
    $another_partial_html,
```

```
    array(
        'a'      => array(
            'href'  => array(),
            'title' => array(),
        ),
        'br'     => array(),
        'em'     => array(),
        'strong' => array(),
    )
); ?>
```

In this example, all tags other than `<a>`, `<br>`, `<em>`, and `<strong>` will be stripped out. Additionally, if an `<a>` tag is passed, the escaping ensures that only the `href` and the `title` are returned.

Always escape late

It is best to do the output escaping as late as possible, ideally as data is being outputted.

It is better to escape late for a few reasons:

- Code reviews and deploys can happen faster because it can be deemed safe for output at a glance, rather than hunting through many lines of code to see where and if it was already escaped.
- Something could inadvertently change the variable between when it was firstly cast and when it is outputted, introducing a potential vulnerability.
- Late escaping makes it easier to do automatic code scanning, saving time and cutting down on review and deploy times.
- Late escaping whenever possible makes the code more robust and future proof.
- Escaping/casting on output removes any ambiguity and adds clarity (always develop for the maintainer).

```
// Okay, but not great.
$url = esc_url( $url );
$text = esc_html( $text );
echo '<a href="'. $url . '">' . $text . '</a>';

// Much better!
echo '<a href="'. esc_url( $url ) . '">' . esc_html( $text ) . '</a>';
```

… Except when you can’t

It is sometimes not practical to escape late. In a few rare circumstances output cannot be passed to `wp_kses()`, since by definition it would strip the scripts that are being generated.

In situations like this, always escape while creating the string and store the value in a variable that is a postfixed with `_escaped`, `_safe` or `_clean` (e.g., `$variable` becomes `$variable_escaped` or `$variable_safe`).

If a function cannot output internally and escape late, then it must always return “safe” HTML. This allows `echo my_custom_script_code();` to be done without needing the script tag to be passed through a version of `wp_kses()` that would allow such tags.

Escaping with Localization

Rather than using `echo` to output data, it’s common to use the WordPress localization functions, such as `_e()` or `__()`.

These functions simply wrap a localization function inside an escaping function:

```
esc_html_e( 'Hello World', 'text_domain' );
// Same as
echo esc_html( __( 'Hello World', 'text_domain' ) );
```

These helper functions combine localization and escaping:

- esc_html__()
- esc_html_e()
- esc_html_x()
- esc_attr__()
- esc_attr_e()
- esc_attr_x()

Examples

Escaping any numeric variable used anywhere

echo $int;

Depending on whether it is an integer or a float, `(int)`, `absint()`, `(float)` are all correct and acceptable.
At times, `number_format()` or `number_format_i18n()` might be more appropriate.

`intval()`, `floatval()` are acceptable, but are outdated (PHP4) functions.

Escaping arbitrary variable within HTML attribute

echo ' ';

This should be escaped with one call to `esc_attr()`.
When a variable is used as part of an attribute or url, it is always better to escape the whole string as that way a potential escape character just before the variable will be correctly escaped.

Correct:

```
echo '<div id="', esc_attr( $prefix . '-box' . $id ), '">';
```

Incorrect:

echo ' ';

Note: nonces created using `wp_create_nonce()` should also be escaped like this if used in an HTML attribute.

Escaping arbitrary URL within HTML attribute, but also in other contexts

echo '<a href="', $url, '">';

This should be escaped with `esc_url()`.

Correct:

```
echo '<a href="', esc_url( $url ), '">';
```

Incorrect:

echo '<a href="', esc_attr($url), '">';
echo '<a href="', esc_attr(esc_url($url)), '">';

Passing an arbitrary variable to JavaScript via `wp_localize_script()`

```
wp_localize_script( 'handle', 'name',
    array(
        'prefix_nonce' => wp_create_nonce( 'plugin-name' ),
        'ajaxurl'      => admin_url( 'admin-ajax.php' ),
        'errorMsg'     => __( 'An error occurred', 'plugin-name' ),
    )
);
```

No escaping needed, WordPress will escape this.

Escaping arbitrary variable within JavaScript block

<script type="text/javascript">
   var myVar = <?php echo $my_var; ?>
</script>

`$my_var` should be escaped with `esc_js()`.

Correct:

```
<script type="text/javascript">
    var myVar = <?php echo esc_js( $my_var ); ?>
</script>
```

Escaping arbitrary variable within inline JavaScript

<a href="#" onclick="do_something(<?php echo $var; ?>); return false;">

`$var` should be escaped with `esc_js()`.

Correct:

```
<a href="#" onclick="do_something(<?php echo esc_js( $var ); ?>); return false;">
```

Escaping arbitrary variable within HTML attribute for use by JavaScript

<a href="#" data-json="<?php echo $var; ?>">

`$var` should be escaped with `esc_js()`, `json_encode()` or `wp_json_encode()`.

Correct:

```
<a href="#" data-json="<?php echo esc_js( $var ); ?>">
```

Escaping arbitrary string within HTML textarea

echo '<textarea>', $data, '</textarea>';

`$data` should be escaped with `esc_textarea()`.

Correct:

```
echo '<textarea>', esc_textarea( $data ), '</textarea>';
```

Escaping arbitrary string within HTML tags

echo ' ', $phrase, ' ';

This depends on whether `$phrase` is expected to contain HTML or not.

- If not, use `esc_html()` or any of its variants.
- If HTML is expected, use `wp_kses_post()`, `wp_kses_allowed_html()` or `wp_kses()` with a set of HTML tags you want to allow.

Escaping arbitrary string within XML or XSL context

echo '<loc>', $var, '</loc>';

Escape with `esc_xml()` or `ent2ncr()`.

Correct:

```
echo '<loc>', ent2ncr( $var ), '</loc>';
```

□□□ #2
□ □□□ □□□ 2026-10-31 18:07:55 UTC
□ □□□ □□□ 2026-10-31 18:07:56 UTC