XSS, CSRF, and WordPress

By: Eli White

CTO & Founding Partner:
musketeers.me

Managing Editor & Conference Chair:
php[architect]

eliw.com - @EliW

php[architect]

About Security

Do we really need to worry about this?

Security? Bah!

Whether big or small.
Someone will try to hack you!

It only takes one person!

Various Attack Vectors

XSS & CSRF oh my!

XSS (Cross Site Scripting)

A user sending data that is executed as script

Many ways this attack can come in, but in all cases:
Everything from a user is suspect (forms, user-agent, headers, etc)
When fixing, escape to the situation (HTML, JS, XML, etc)
FIEO (Filter Input, Escape Output)

XSS - Reflected XSS

Reflected XSS

Directly echoing back content from the user

The Security Hole:

```
<p>Thank you for your submission: <?= $_POST['first_name'] ?></p>
```

The Attack:

First Name: <script>alert('XSS')</script> Submit

XSS - Reflected XSS

Raw PHP Solutions

The Solution (HTML):

```
$name = htmlentities($_POST['first_name'], ENT_QUOTES, 'UTF-8', FALSE);
```

The Solution (JS):

```
$name = str_replace(array("\r\n","\r","\n"),
                    array("\n","\n","\\\n"),addslashes($_POST['first_name']));
```

The Solution (XML):

```
$name = iconv('UTF-8', 'UTF-8//IGNORE',
                preg_replace("#[\\x00-\\x1f]#msi", ' ',
                str_replace('&', '&amp;', $_POST['first_name'])));
```

XSS - Reflected XSS - WordPress

The Solution (HTML):

```
esc_html($text);
```

Returns the text escaped for safe HTML output, equivalent of htmlspecialchars()

```
esc_textarea($text);
```

Encodes text to be safely used inside of a <textarea> element.

```
esc_attr($text);
```

Encodes text to be used safely inside of an HTML tag attribute.

```
esc_url($text);
```

Sanitizes an URL to be output.

The Solution (JS):

```
$name = esc_js($text);
```

The Solution (XML):

```
$name = iconv('UTF-8', 'UTF-8//IGNORE',
                preg_replace("#[\\x00-\\x1f]#msi", ' ',
                str_replace('&', '&amp;', $_POST['first_name'])));
```

XSS - Stored XSS

Stored XSS

You store the data, then later display it

The Security Hole:

```php
<?php
$query = $wpdb->prepare(
    "UPDATE conferenceCFP SET first = %s WHERE id = 42",
    array($_POST['first_name']));
$query->query($query);
?>

[...]

<?php
$result = $wpdb->get_row("SELECT * FROM users WHERE id = 42");
?>
<p>Welcome to <?= $result->first ?>'s CFP Dashboard</p>
```

XSS - Stored XSS - WordPress

The Solution (HTML):

```
esc_html($result->first);
```

Returns the text escaped for safe HTML output, equivalent of htmlspecialchars()

```
esc_textarea($result->first);
```

Encodes text to be safely used inside of a <textarea> element.

```
esc_attr($result->first);
```

Encodes text to be used safely inside of an HTML tag attribute.

```
esc_url($result->first);
```

Sanitizes an URL to be output.

The Solution (JS):

```
$name = esc_js($result->first);
```

The Solution (XML):

```
$name = iconv('UTF-8', 'UTF-8//IGNORE',
                preg_replace("#[\\x00-\\x1f]#msi", ' ',
                str_replace('&', '&amp;', $result->first)));
```

The Same!

XSS - DOM XSS

DOM XSS

What happens in JavaScript, stays in JavaScript

The Security Hole:

```
<script>
$('#verify').submit(function() {
    var first = $(this).find("input[name=first]").val();
    $(body).append("<p>Thanks for the submission: " + first + "</p>");
    return false;
});
</script>
```

XSS - DOM XSS

DOM XSS

What happens in JavaScript, stays in JavaScript

The Solution (Simple):

```
<script>
function escapeHTML(str) {
     str = str + ""; var out = "";
     for (var i=0; i<str.length; i++) {
         if (str[i] === '<') { out += '&lt;'; }
         else if (str[i] === '>') { out += '&gt;'; }
         else if (str[i] === "'") { out += '&#39;'; }
         else if (str[i] === '"') { out += '&quot;'; }
         else { out += str[i]; }
     }
     return out;
}
</script>
```

Or just never directly echo in JS, always roundtrip to the server.

XSS - DOM XSS - jQuery Encoder

Since you need to escape output differently in JavaScript based upon whether it's being used as a tag name, CSS, attribute, class, etc.

One library that can help you is jQuery Encoder:

https://github.com/chrisisbeef/jquery-encoder/

Provided Methods:

```
encodeForCSS( String input, char[] immune )
encodeForHTML( String input )
encodeForHTMLAttribute( String input, char[] immune )
encodeForJavascript( String input, char[] immune )
encodeForURL( String input, char[] immune )
```

CSRF (Cross Site Request Forgery)

A user having the ability to forge or force a request on behalf of another user.

Simplistically via IMG tag or POST forms

Complicated via JavaScript

CSRF (Cross Site Request Forgery)

A user having the ability to forge or force a request on behalf of another user.

The Attack:

```
<img width="1" height="1"
     src="http://quackr.example.com/quackit?msg=CSRF+Attacks+Rock!" />
```

or

```
<script>
$.post({
    url: 'http://quackr.example.com/quackit',
    data: { msg: 'CSRF Attacks Rock!'}
});
</script>
```

CSRF (Cross Site Request Forgery)

Protect via CSRF token

The Solution (on form):

```
<form method="POST" action="">
    <input name="msg" value="" />
    <input type="hidden" name="token" value="<?= wp_create_nonce($action); ?>" />
    <input type="submit" />
</form>
```

The Solution (on submission):

```
<?php
if (wp_verify_nonce( $POST[‘token’], $action ) {
    // SUCCESS - Process the form
} else {
    // FAILURE - Block this:
    header('HTTP/1.0 403 Forbidden');
    die;
}
?>
```

CSRF (Cross Site Request Forgery)

You can also call the following to have it create the form field for you:

```
wp_nonce_field( $action, $name );
```

In AJAX context, there's a different function for verifying the nonce:

```
if (check_ajax_referer( $action, $query_arg, $die )) { /* ... */ };
```

The $die above is optional, but defaults to TRUE, which means it will completely kill the request if the token verification fails.

For this presentation & more:
http://eliw.com/

Twitter: @EliW

php[architect]: http://phparch.com/
musketeers: http://musketeers.me/

php[architect]