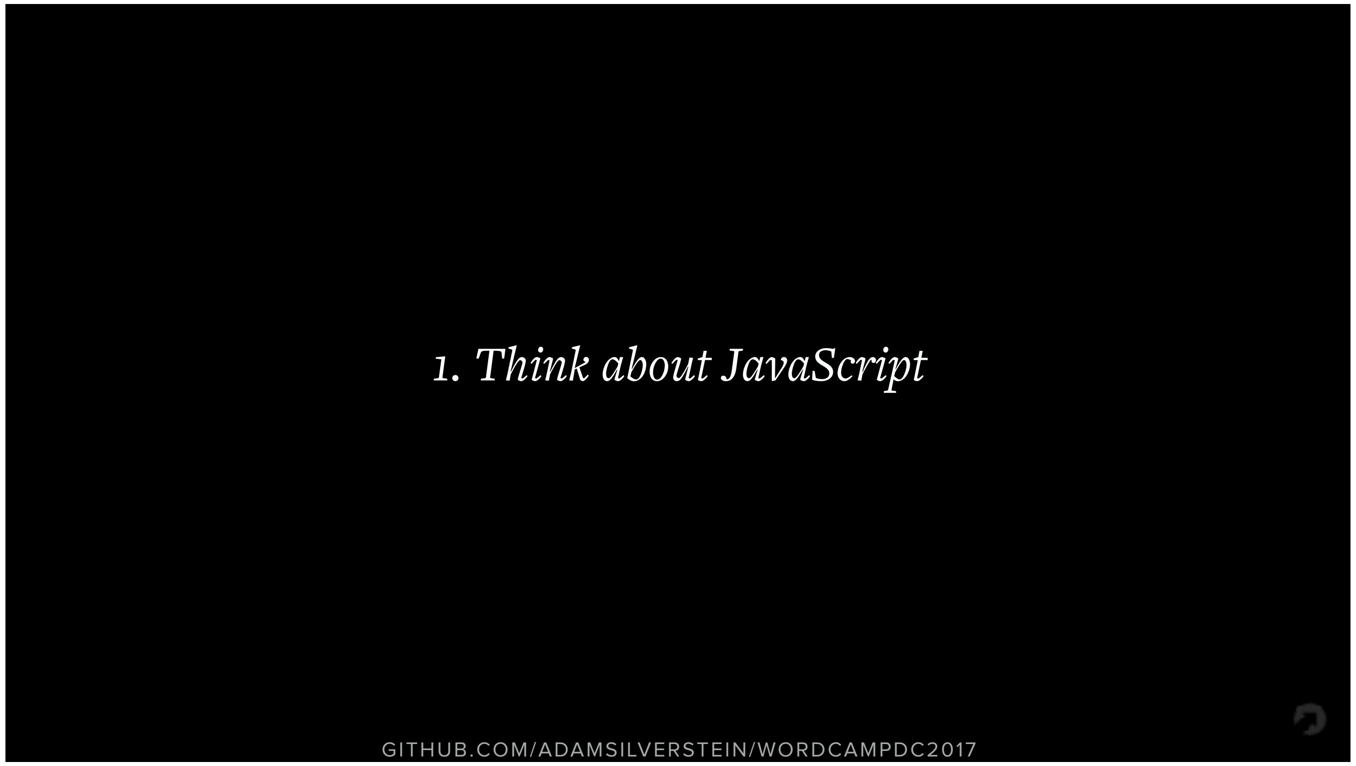
10 Tips for Securing your WordPress JavaScript

Adam Silverstein

@ROUNDEARTH



1. Think about JavaScript

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

Hello everyone!
Lets start off with a bit of background into how I became interested in JavaScript security.
I work for the digital agency 10up and we create websites for enterprise clients. We typically work with large clients where security is paramount.
 My first year at 10up I worked almost exclusively auditing code as we helped clients move to more secure hosting such as WordPress VIP where many of our highest traffic clients host their sites

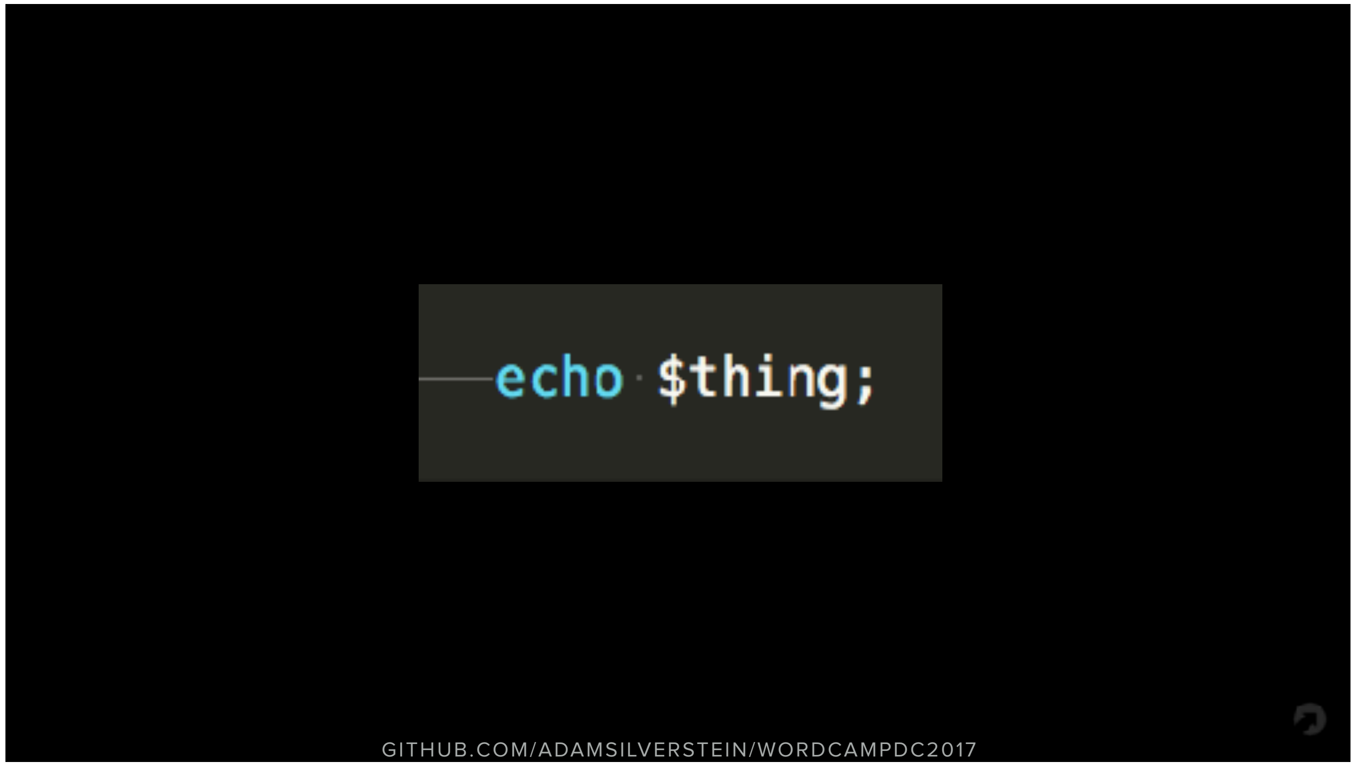
```
		}
	}

	/**
	 * Sets the table prefix for the WordPress tables.
	 *
	 * @since 2.5.0
	 *
	 * @param string $prefix Alphanumeric name for the new prefix.
	 * @param bool $set_table_names Optional. Whether the table names, e.g. wpdb::$posts, should be updated or not.
	 * @return string|WP_Error Old prefix or WP_Error on error
	 */
	public function set_prefix( $prefix, $set_table_names = true ) {

		if ( preg_match( '|[^a-z0-9_]|i', $prefix ) )
			return new WP_Error('invalid_db_prefix', 'Invalid database prefix' );

		$old_prefix = is_multisite() ? '' : $prefix;

		if ( isset( $this->base_prefix ) )
			$old_prefix = $this->base_prefix;

		$this->base_prefix = $prefix;

		if ( $set_table_names ) {
			foreach ( $this->tables( 'global' ) as $table => $prefixed_table )
				$this->$table = $prefixed_table;

			if ( is_multisite() && empty( $this->blogid ) )
				return $old_prefix;

			$this->prefix = $this->get_blog_prefix();

			foreach ( $this->tables( 'blog' ) as $table => $prefixed_table )
				$this->$table = $prefixed_table;

			foreach ( $this->tables( 'old' ) as $table => $prefixed_table )
				$this->$table = $prefixed_table;
		}
		return $old_prefix;
```

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

I spent long days looking very very closely at PHP.

Our audits focused on security and performance, and there were a handful of concerning patterns we learned to watch for and remediate.

We really paid very little attention the the JavaScript part of the code which was often minimal in any case.

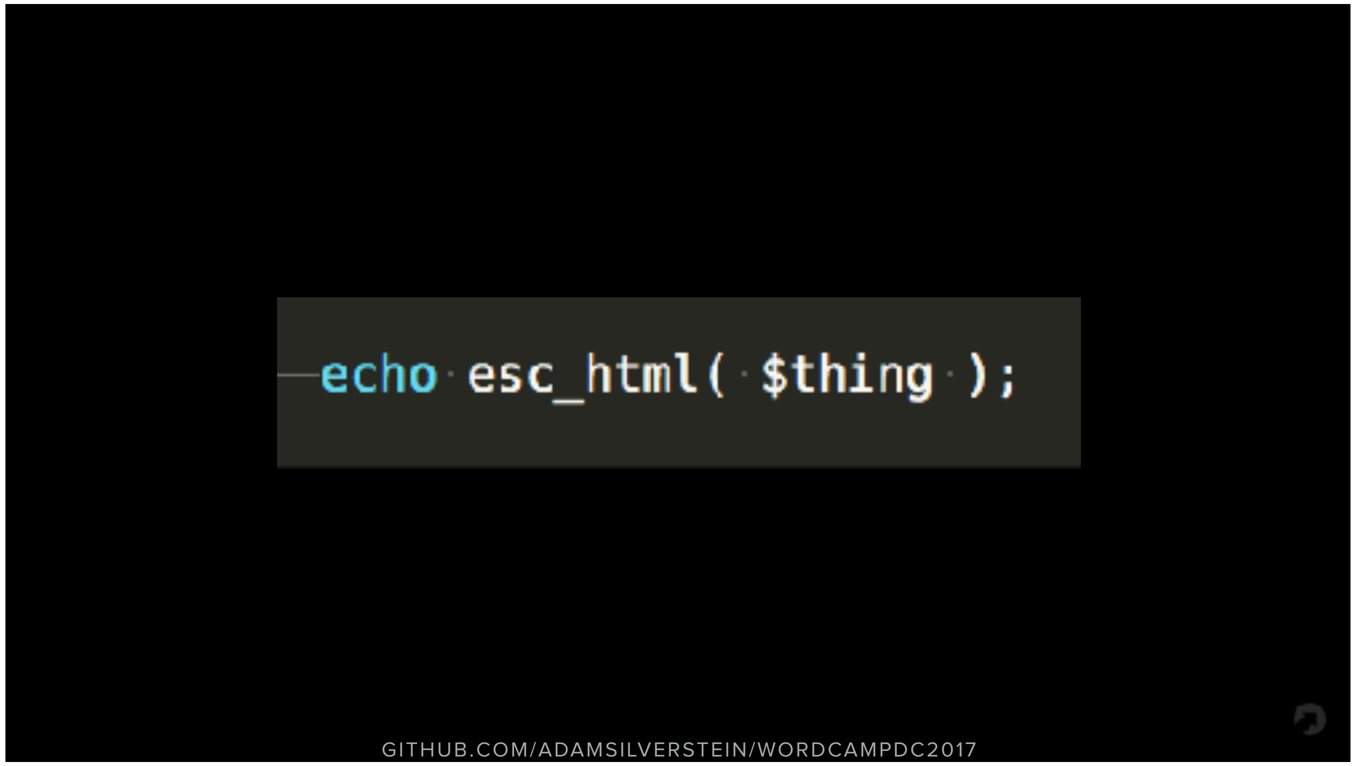


```
echo $thing;
```

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

Well, we did pay attention to JavaScript in one way. One of the major things we looked for was the lack of late escaping.

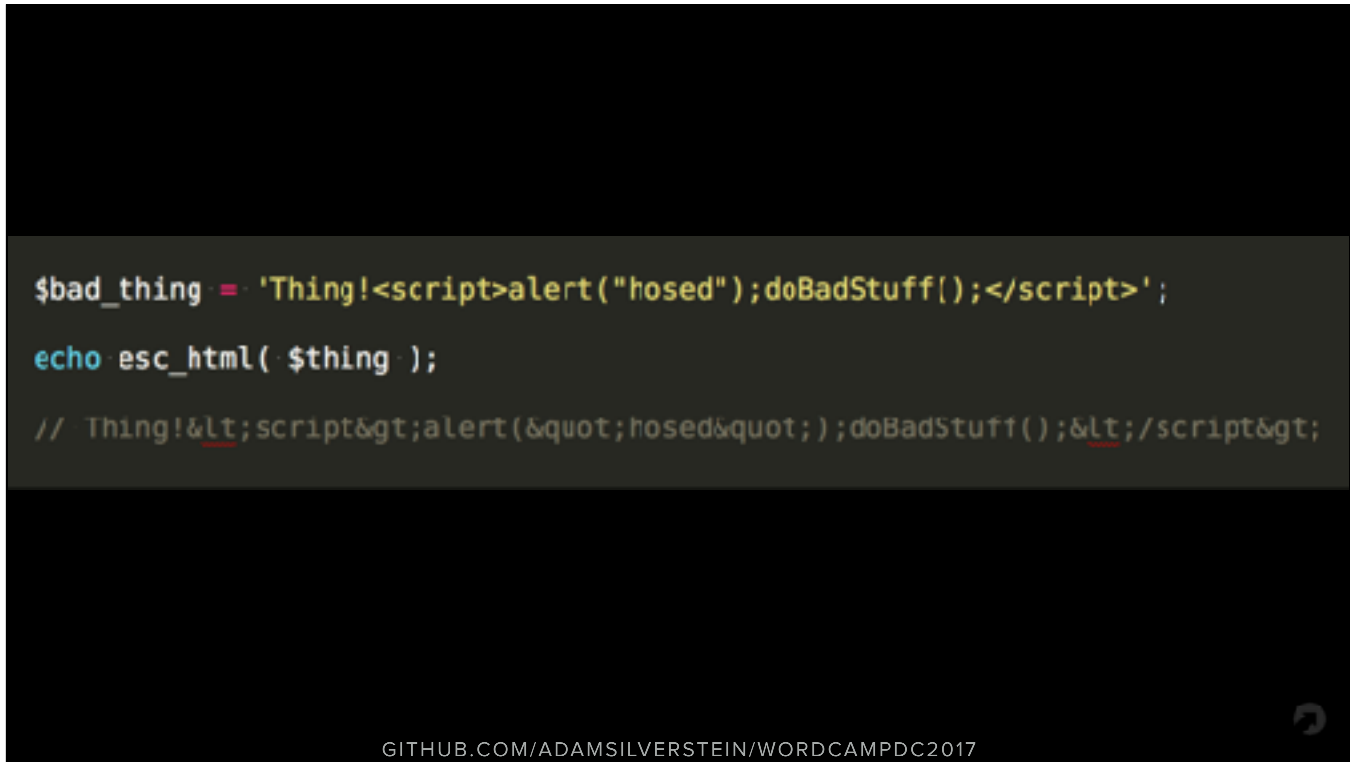
So whenever we saw output some value in a theme or plugin



```
echo esc_html( $thing );
```

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

We expected it to be wrapped in one of WordPress’s escaping functions, depending on the context

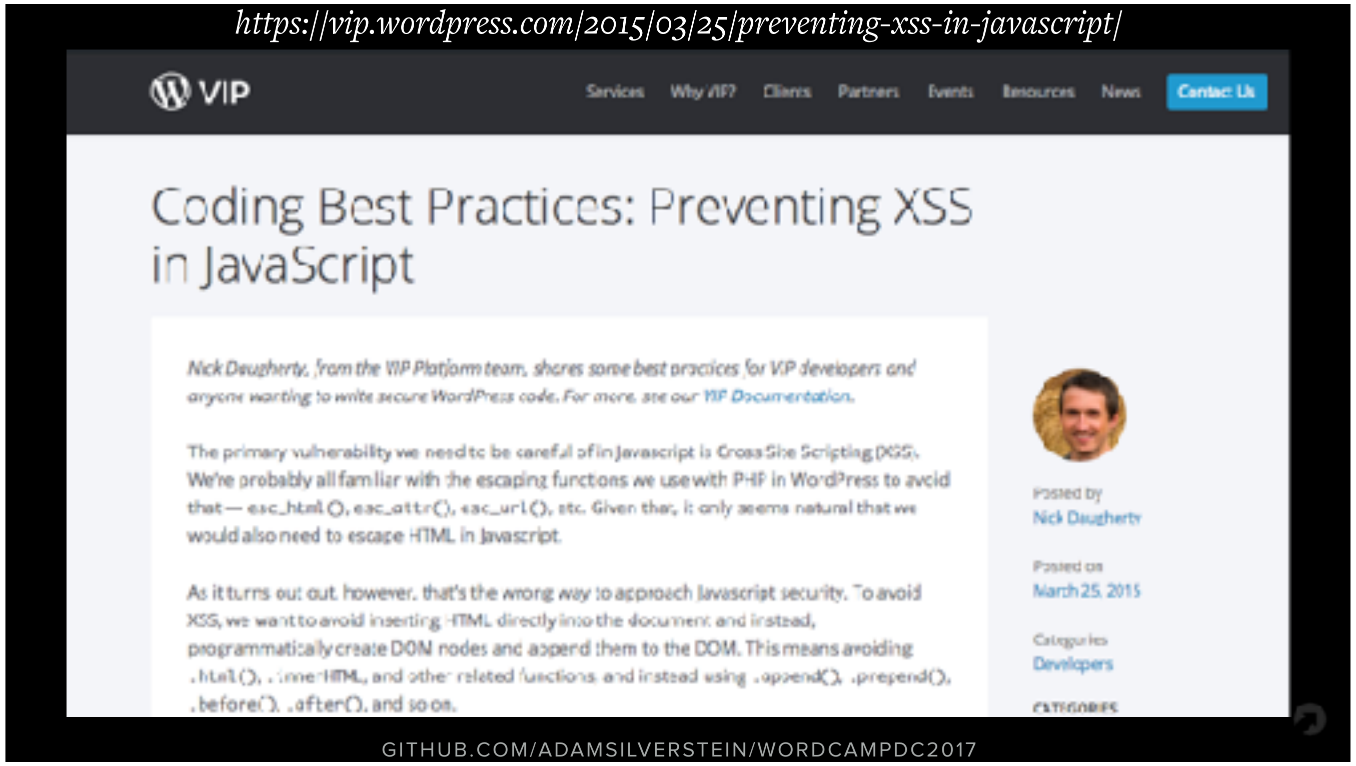


```
$bad_thing = 'Thing!<script>alert("hosed");doBadStuff();</script>';

echo esc_html( $thing );

// Thing!&lt;script&gt;alert(&quot;hosed&quot;);doBadStuff();&lt;/script&gt;
```

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

And what this does most importantly is turn a malicious JavaScript string into a harmless html encoded string that is safe to output. so we were paying a little attention to keeping malicious JavaScript off the page, at least from the PHP side.

https://vip.wordpress.com/2015/03/25/preventing-xss-in-javascript/



Coding Best Practices: Preventing XSS in JavaScript

Nick Daugherty, from the VIP Platform team, shares some best practices for VIP developers and anyone wanting to write secure WordPress code. For more, see our VIP Documentation.

The primary vulnerability we need to be careful of in Javascript is Cross Site Scripting (XSS). We're probably all familiar with the escaping functions we use with PHP in WordPress to avoid that — `esc_html()`, `esc_attr()`, `esc_url()`, etc. Given that, it only seems natural that we would also need to escape HTML in Javascript.

As it turns out out, however, that's the wrong way to approach Javascript security. To avoid XSS, we want to avoid inserting HTML directly into the document and instead, programmatically create DOM nodes and append them to the DOM. This means avoiding `.html()`, `.innerHTML`, and other related functions and instead using `.append()`, `.prepend()`, `.before()`, `.after()`, and so on.

Posted by
Nick Daugherty

Posted on
March 25, 2015

Categories
Developers

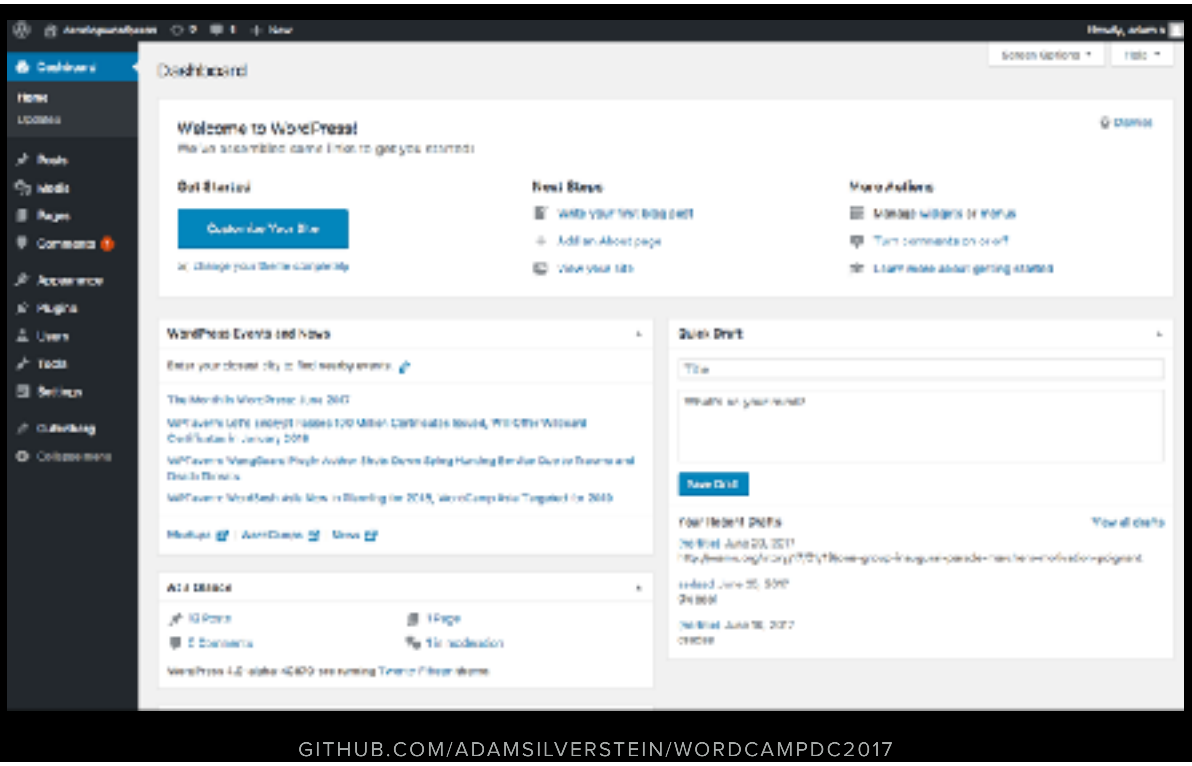
CATEGORIES

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

Then things started changing around 2 years ago. We gradually learned that we had to pay attention to all the JavaScript running on the page - where it loaded from and where it got its data and how it displayed it.

This post from Nick Dougherty on the WordPress VIP blog highlighted some of the biggest issues we had been ignoring, and VIP themselves became far more strict about reviewing JavaScript code than they had been in the past.

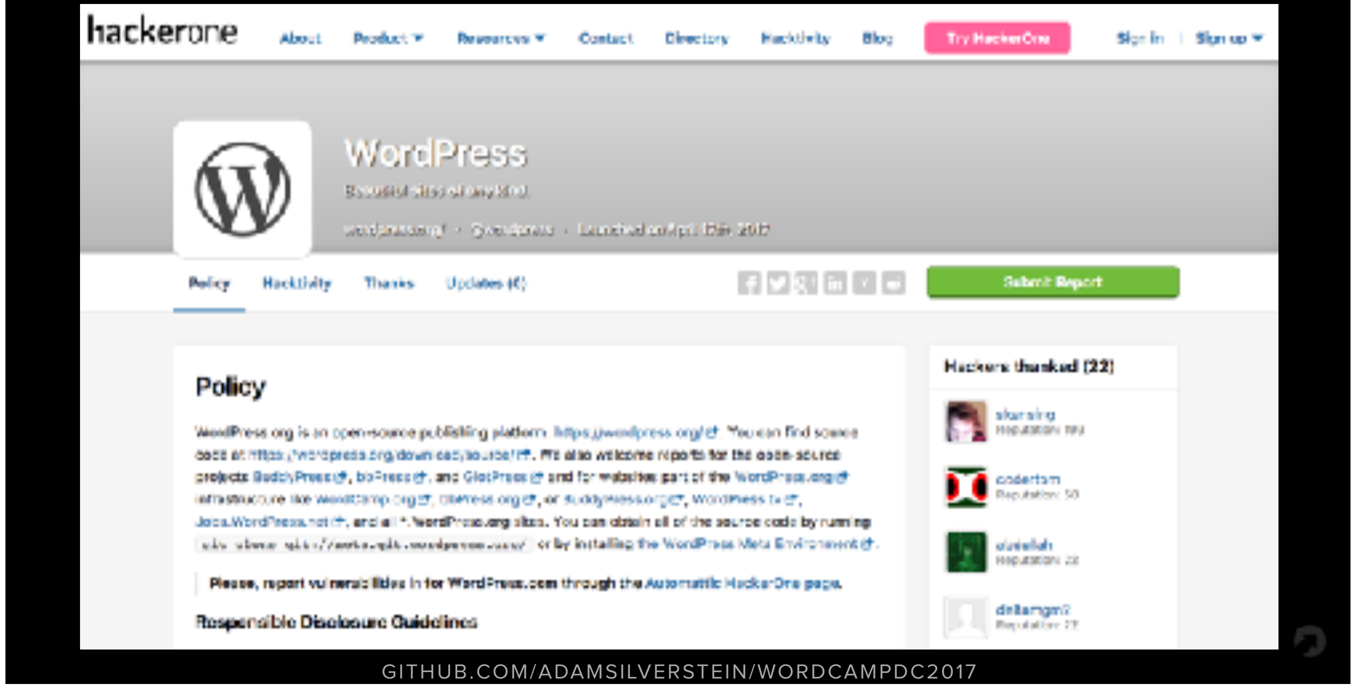
The big takeaway though was start paying attention to JavaScript.



GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

I’ve seen a similar effort taking place in WordPress core where I work as part of the security team.

We have JavaScript enhancing nearly every screen of wp-admin now, and its been developed over the last 14 years - since the very first version. This legacy of old code has the potential to expose security vulnurabilities *https://hackerone.com/wordpress*



Policy

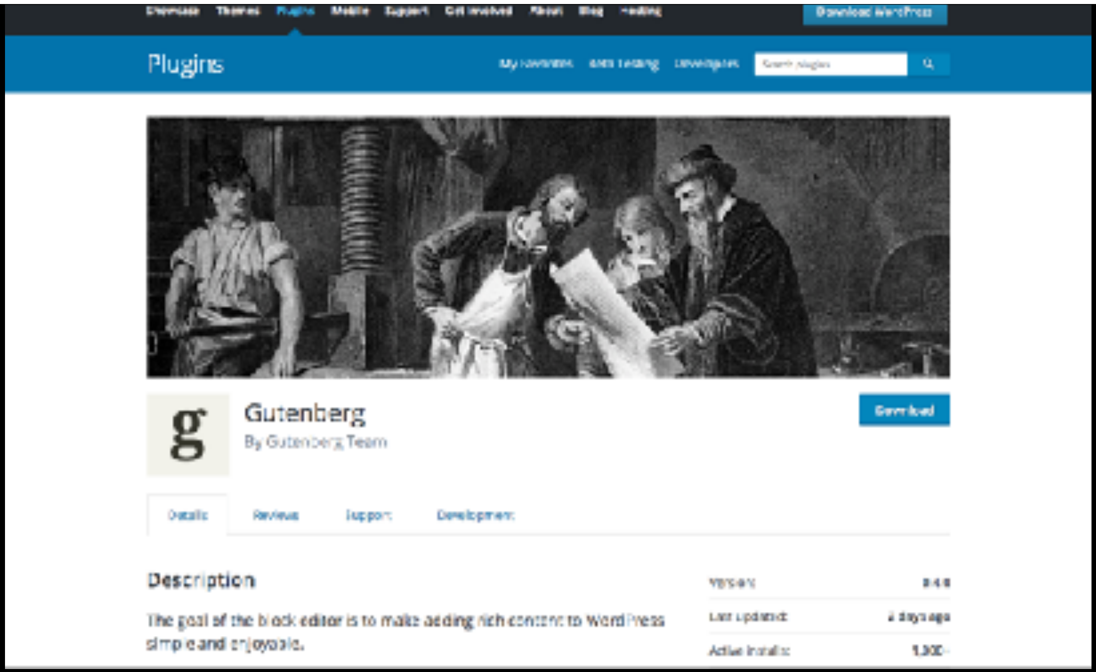
WordPress.org is an open-source publishing platform https://wordpress.org/. You can find source code at https://wordpress.org/download/source/. We also welcome reports for the open-source projects BuddyPress, bbPress, and GlotPress and for websites part of the WordPress.org infrastructure like WordCamp.org, bbPress.org, or BuddyPress.org, WordPress.tv, Jobs.WordPress.net, and all *.WordPress.org sites. You can obtain all of the source code by running [illegible] or by installing the WordPress Meta Environment.

> **Please, report vulnerabilities in for WordPress.com through the Automattic HackerOne page.**

Responsible Disclosure Guidelines

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

We are constantly working to find and fix these issues, and this spring WordPress core opened our security reporting with HackerOne. HackerOne rewards friendly hackers with cash bounties for responsibly disclosing issues. Because JavaScript security hasn’t had much attention over the years, its become an easy attack vector for hackers to probe.



Gutenberg

By Gutenberg Team

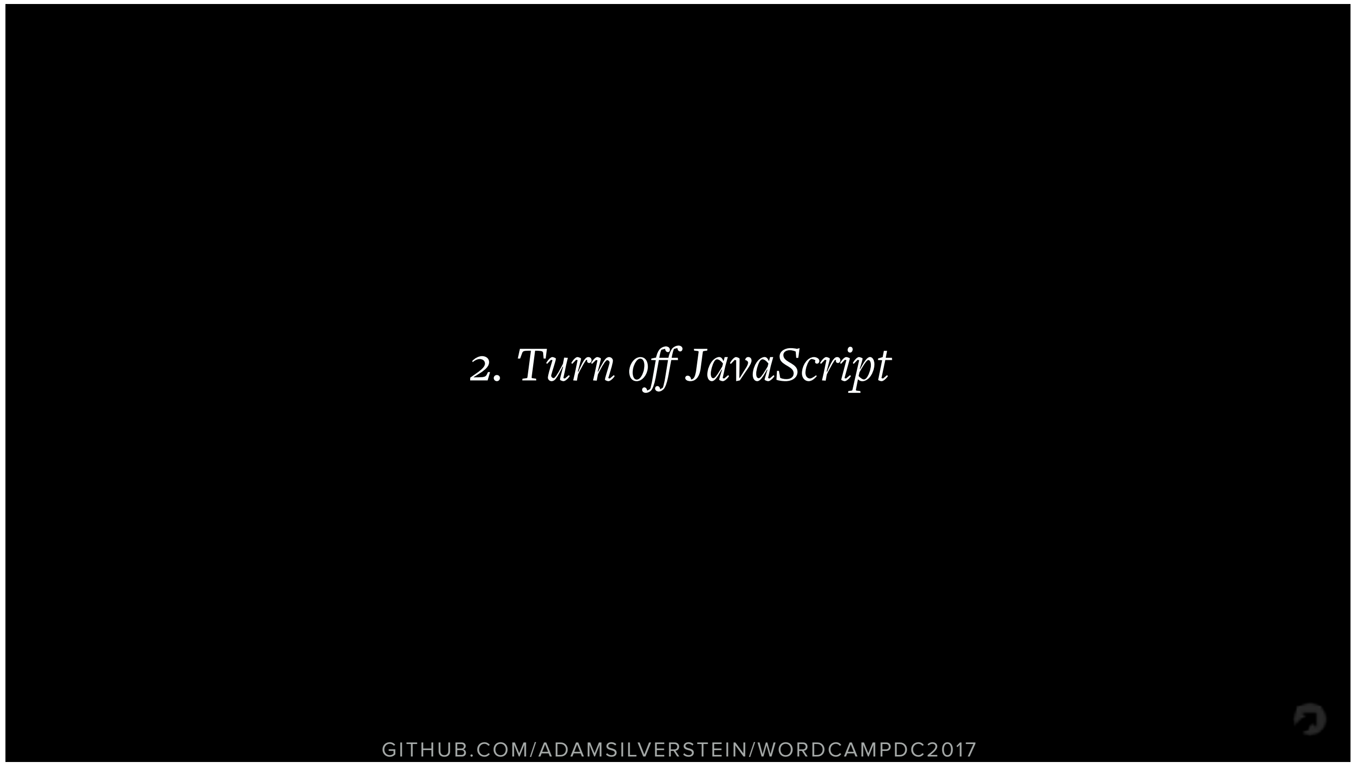
Description

The goal of the block editor is to make adding rich content to WordPress simple and enjoyable.

Version:	[illegible]
Last Updated:	2 days ago
Active Installs:	1,000+

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

If you haven’t see this yet, its new editor being developed for WordPress called Gutenburg which is built in JavaScript. So as we move into an age where JavaScript drives more and more of what we think of as WordPress, we need to pay more attention to JavaScript security issues.



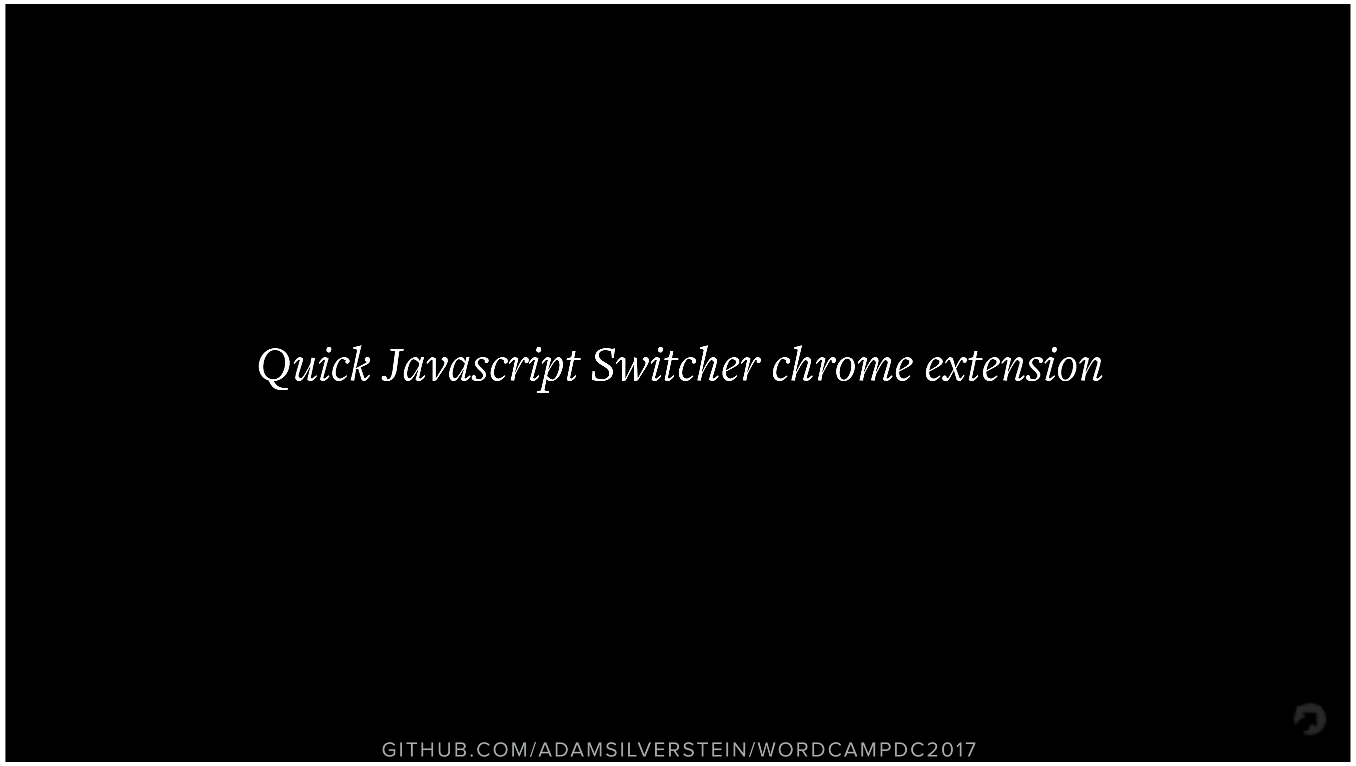
2. Turn off JavaScript

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

While this may seem a little drastic, its at least worth acknowledging that some people will take this route for security reasons.

If a user somehow happen to run malicious JavaScript in their browser, the JavaScript can do anything they would normally do in their browser

That means if they are an admin, the JS can edit posts, add and remove users and content, and even worse edit theme or plugin files directly via the wp-admin—>edit screens - letting them place malicious PHP code directly into your site that will load for every subsequent visitor. So some network administrators may decide that turning off JavaScript for their users is the only truly safe option.

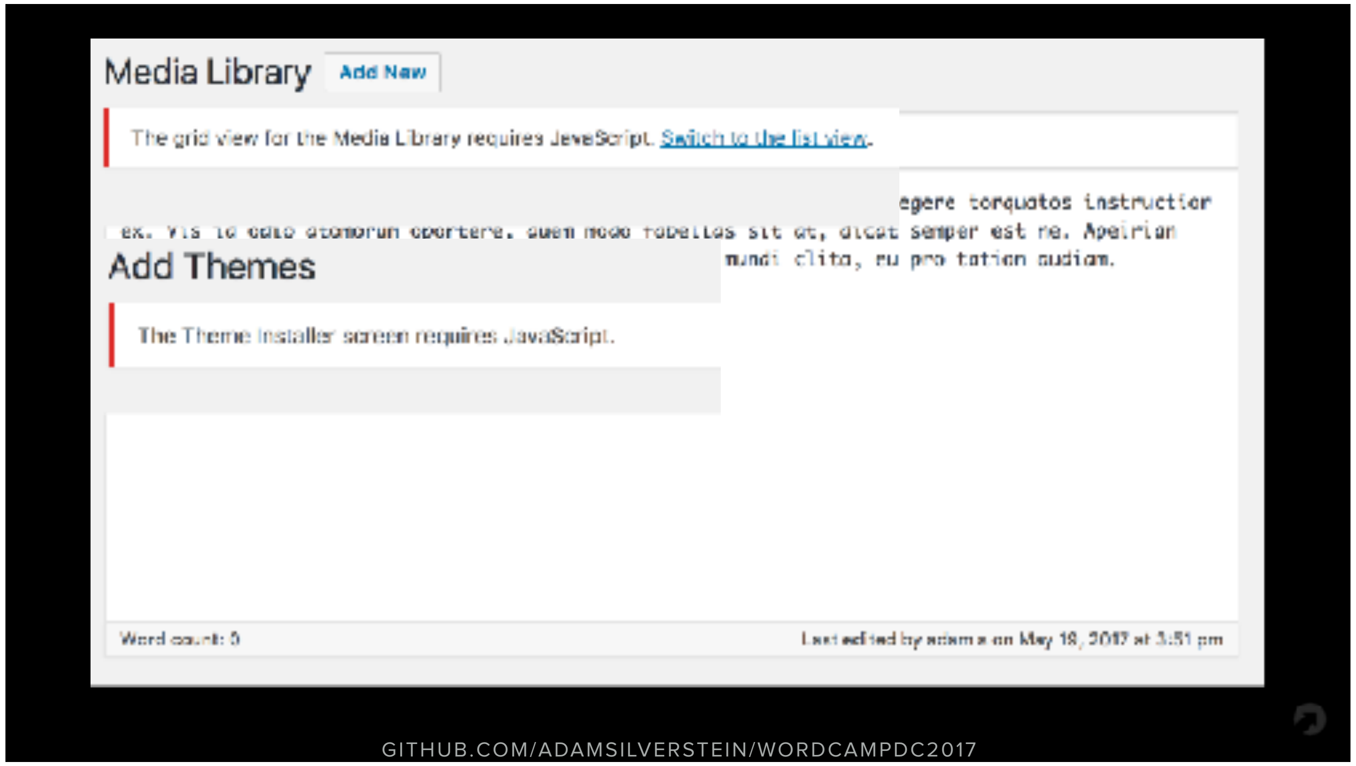


Quick Javascript Switcher chrome extension

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

So try this experiment now or later at home— install the Quick Javascript Switcher chrome extension to find out what a secure browser experience feels like.

you will probably notice a few things right away - first, the internet is wickedly fast without javascript. your browser goes from feeling like a sluggish behemoth to a spry sports car. secondly or maybe first, you will notice that a big swatch of the internet doesn't work at all. try out your 5 or 10 top sites and see what works. Then, try out WordPress.



Media Library Add New

The grid view for the Media Library requires JavaScript. Switch to the list view.

egere torquatos instructior ex. Vis id odio atomorum oportere. Quem modo fabellas sit at, dicat semper est ne. Apeirian mundi clita, eu pro tation audiam.

Add Themes

The Theme Installer screen requires JavaScript.

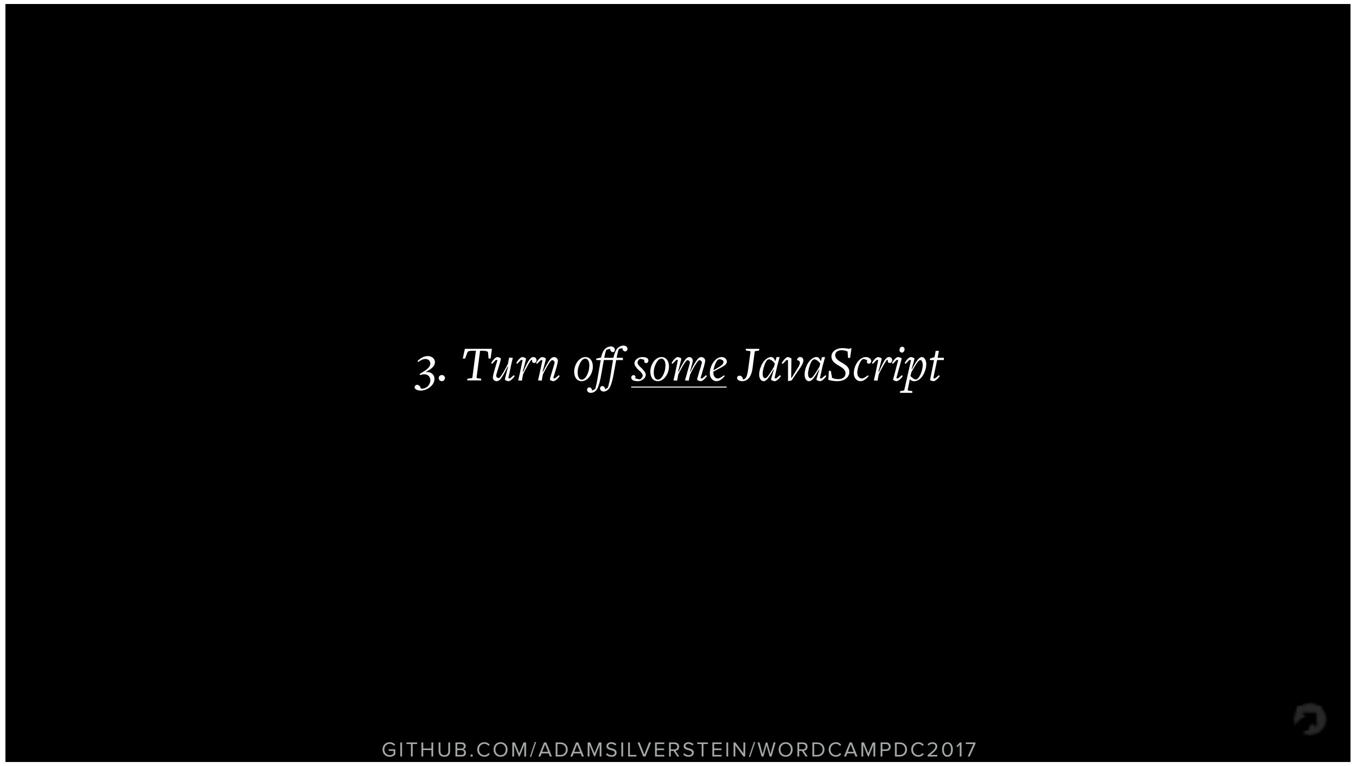
Word count: 5

Last edited by adamsilverstein on May 18, 2017 at 3:51 pm

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

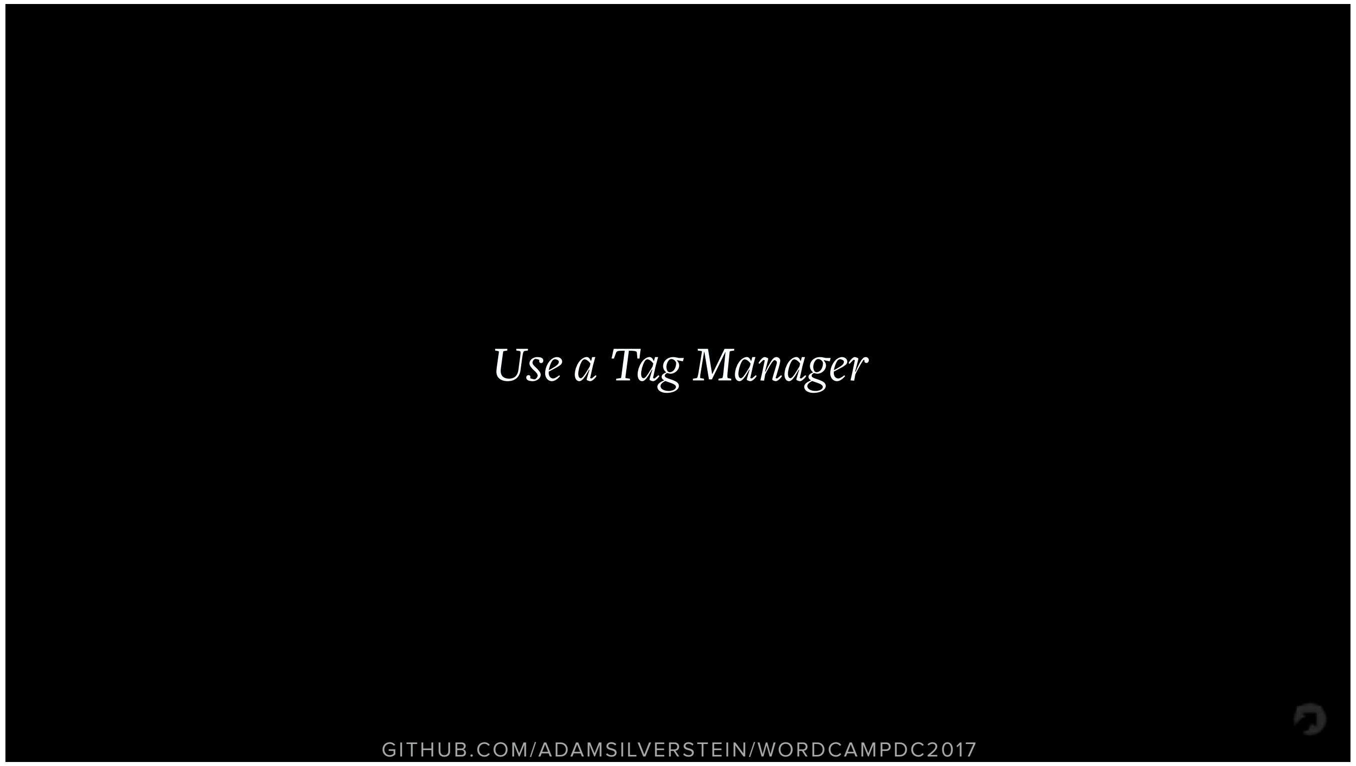
While you can navigate around wp-admin and create a simple post, the experience is pretty minimal - more like the web of 10 years ago - and quite a bit of basic functionality is completely missing including the media grid, the theme browser and the ability to set a featured image.

So, turn JavaScript back on and welcome back to 2017.



3. Turn off some JavaScript

so May turn off some javascript and block the malicious stuff? This applies to users and also web publishers.

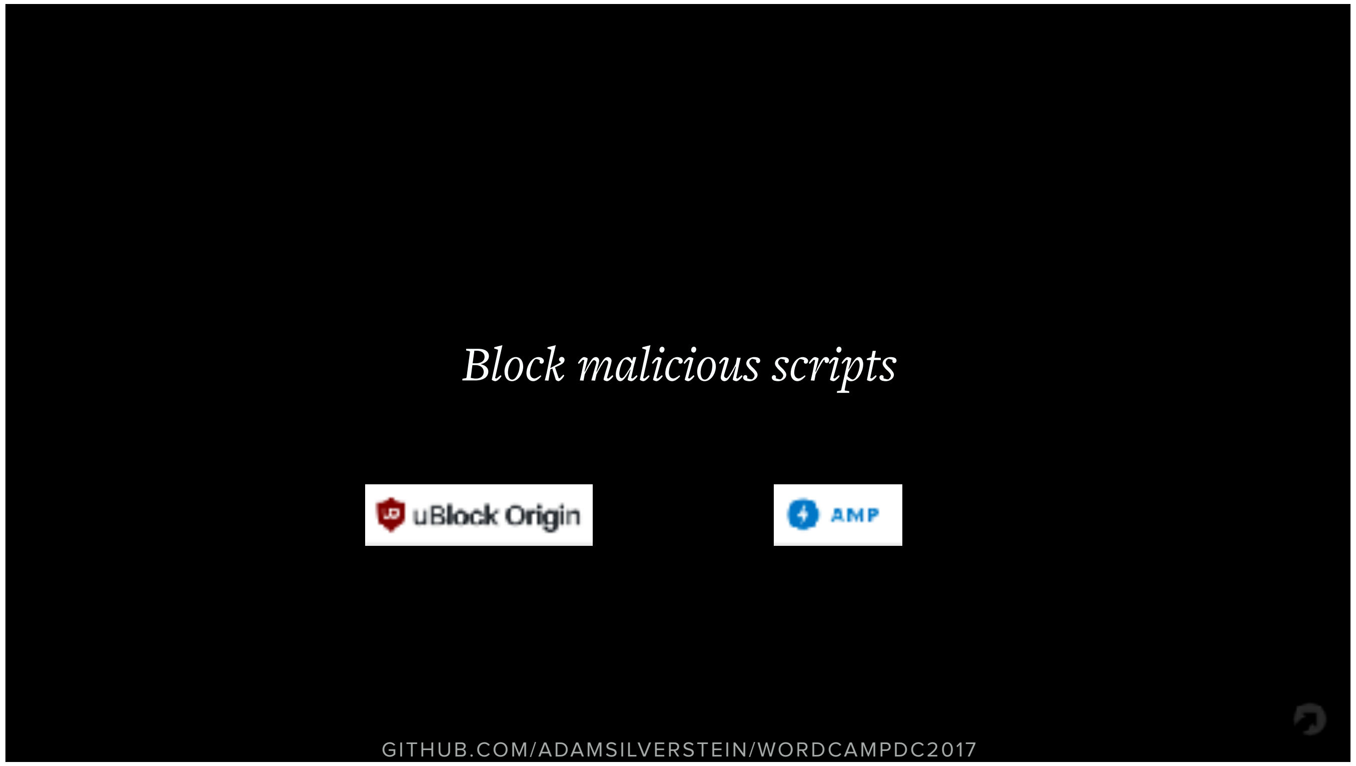


Use a Tag Manager

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

If you are a publisher and you are serving ads on your site using DFP or any third part ad server, you should be aware that ads are scripts that run on your page, not simply images. These scripts are delivered by the ad server, and can include secondary tracking pixels or script wrappers that are delivered alongside the ads and you may not even have full control over what makes it onto the page.

Tag managers lets you control which scripts make it onto your page and when they show up and promise to let you limit or prevent malicious scripts as part of their feature set. There are many offering out there include the free Google tag manager, Tealium, Adobe DTM and OpenX's Mezzobit.

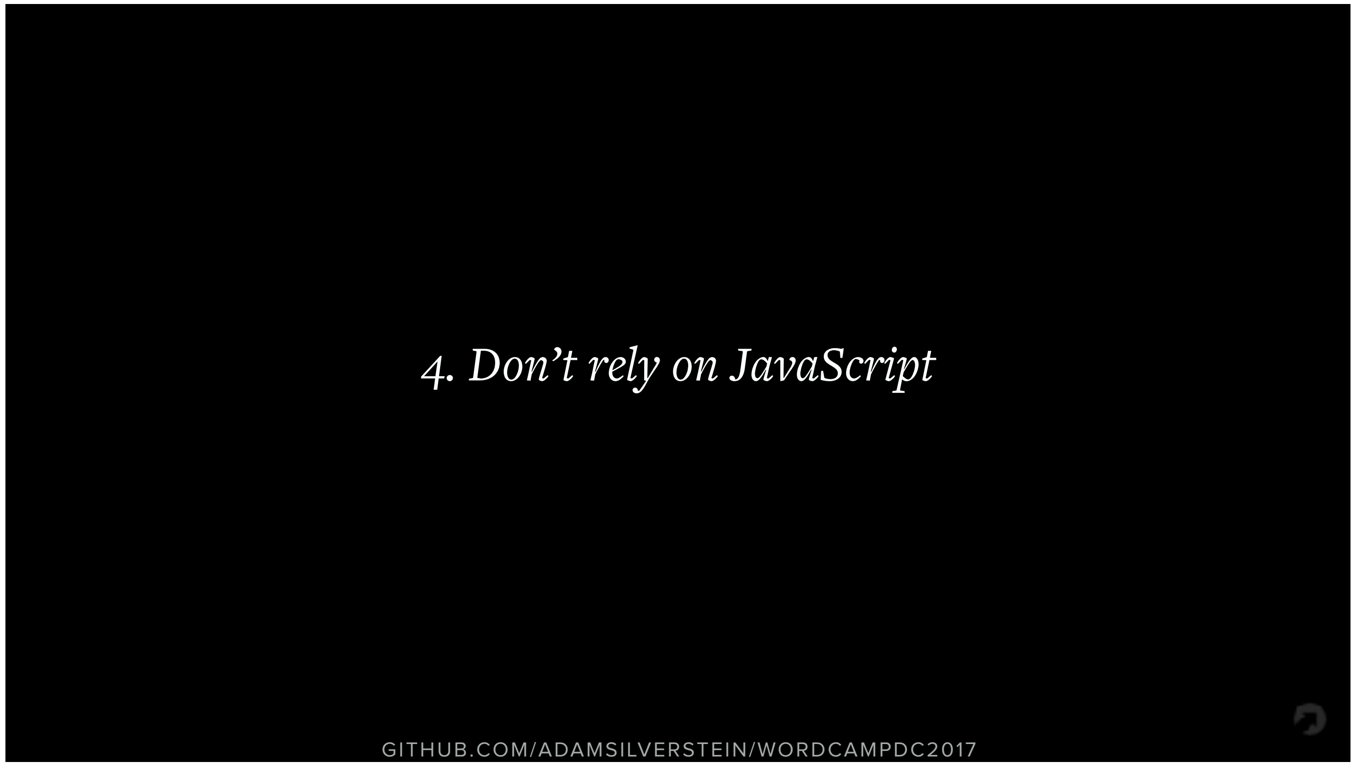


Block malicious scripts

uBlock Origin
AMP

If you are a user - unfortunately, one of the main ways malicious JavaScript makes its way into your browser is via a website's advertising or third party scripts. so you may want to consider an advanced script blocker like uBlock origin. More than an 'ad blocker' this tool blocks all third party scripts and offer controls for novices to advanced users. Even if you don't block scripts with it, its fascinating to see how many scripts are loading on each page you use. Maybe consider avoiding sites with dozens and dozens of script tags on the page - these irresponsible publishers don't deserve your patronage.

Another option is to try the AMP version of a page when you see it available marked by a lightning bolt on google mobile search results — AMP's strict and slim spec results in limited JavaScript kruft.



4. Don’t rely on JavaScript

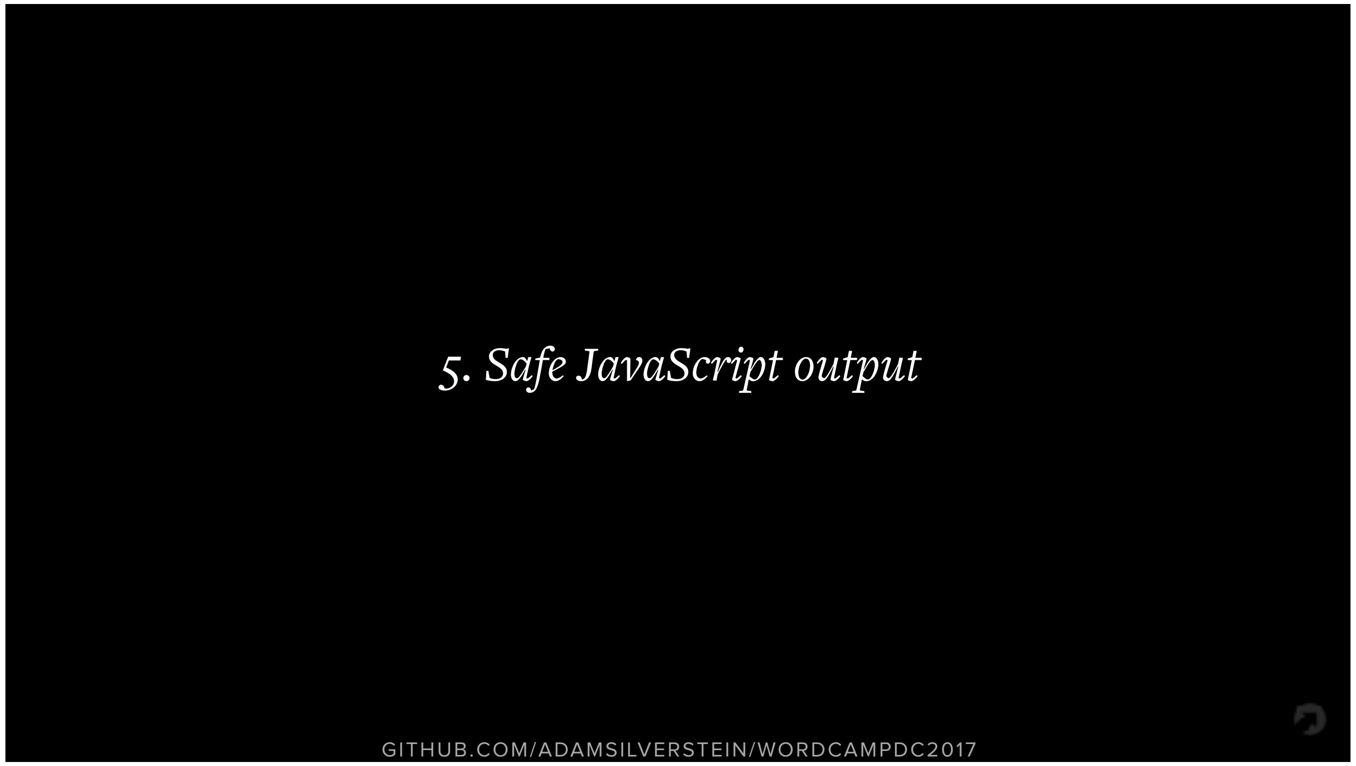
GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

This is part of respecting that fact that some users can’t accept using JavaScript.

If you are building features for a theme or plugin that rely on JavaScript, consider what happens when a user has JavaScript disabled.

Use your handy chrome extension to test out your feature with JavaScript off.

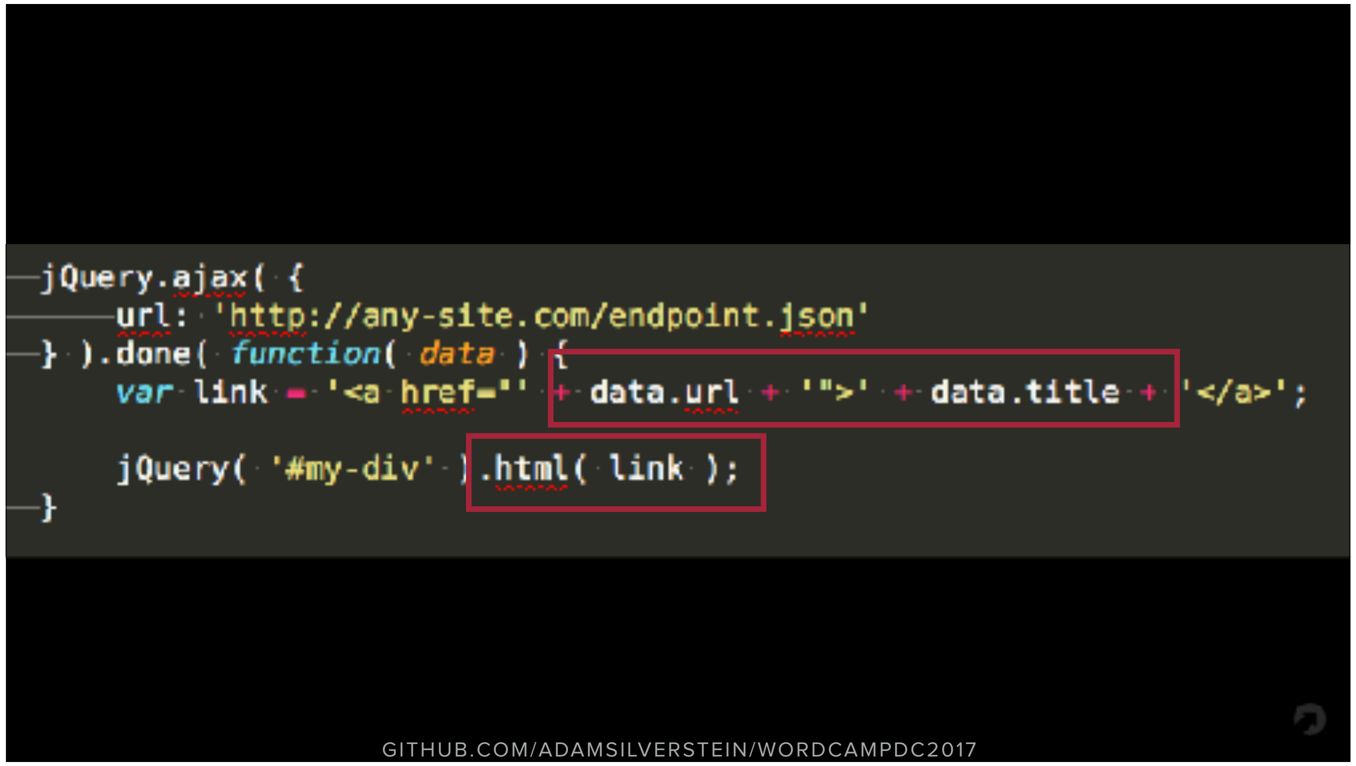
If you can, make the feature work with no JavaScript, or you can at the very least add a notification to tell the user that the feature won’t work correctly without JavaScript.



5. Safe JavaScript output

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

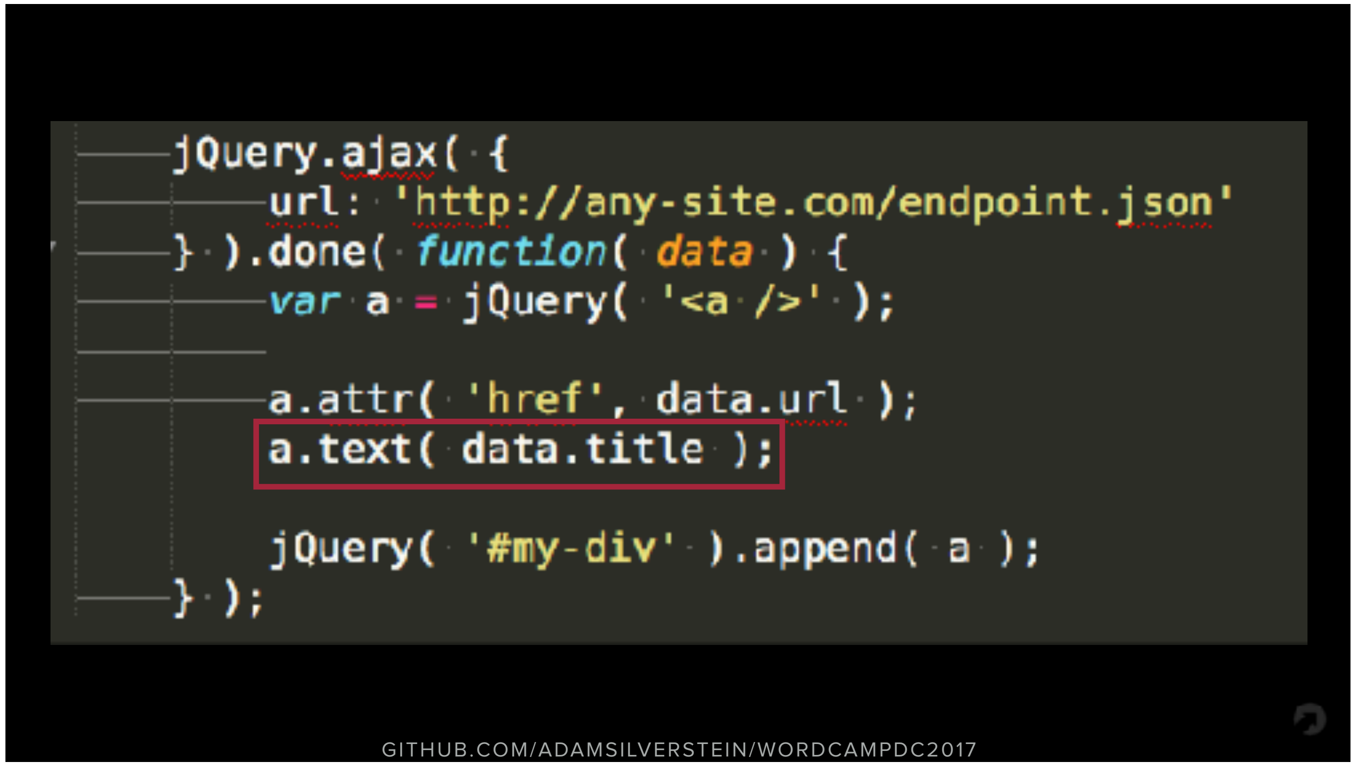
A very important consideration is how you output data in JavaScript. Common scenarios for output include displaying a confirmation or error message sent back from the server from an Ajax action, or lazy loaded content like infinite scroll pages or a comment section.



```
jQuery.ajax( {
        url: 'http://any-site.com/endpoint.json'
} ).done( function( data ) {
        var link = '<a href="' + data.url + '">' + data.title + '</a>';

        jQuery( '#my-div' ).html( link );
}
```

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

The common approach of outputting HTML strings with jQuery's html() or the related insertion methods like append() is dangerous because the underlying innerHTML functionality executes any scripts you pass in. this code is dangerous because we don't really know what we will get back from the ajax request, and if a hacker manages to insert a malicious script in this response variables, that script will execute when this last call runs.



```
jQuery.ajax( {
    url: 'http://any-site.com/endpoint.json'
} ).done( function( data ) {
    var a = jQuery( '<a />' );

    a.attr( 'href', data.url );
    a.text( data.title );

    jQuery( '#my-div' ).append( a );
} );
```

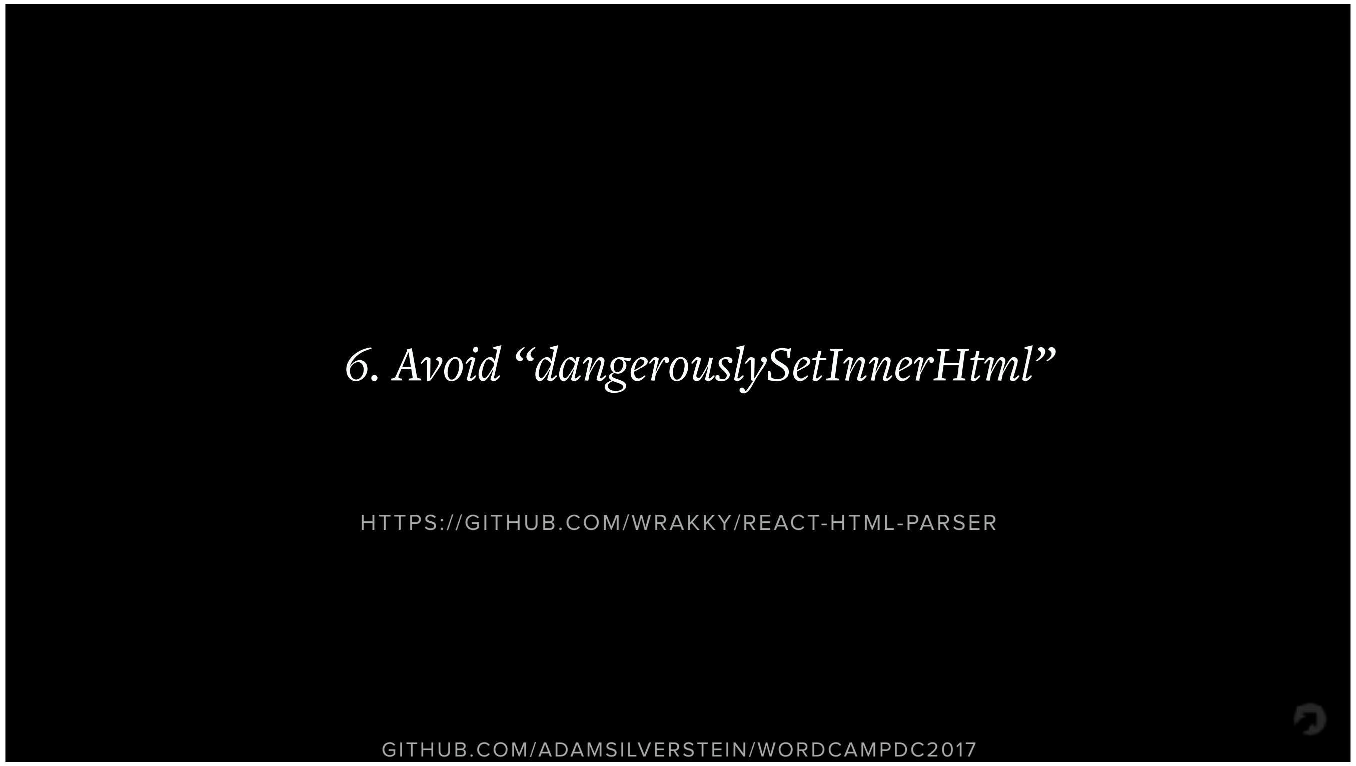
GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

the best solution in most cases is to sanitize or escape the untrusted data before you use it. in this refactoring you can see that i'l using the attr or attribute function to set the link url, and i'm using the text function to set the title inside the a tag. neither of these actions will execute javascript, and the untrusted values can thus be safely used. the critical thing here is that we aren't blindly outputting what we get back to the DOM - instead, we do careful dom construction to build out what we need, using safe functions like text() to handle untrusted data.

wp.sanitize

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

We also have a couple of helper functions in WordPress core - stripTags to strip out HTML tags from content and sanitizeText to Strip HTML tags and convert HTML entities. these are already in core as part of press this and will become easier to use in WordPress 4.9 with the introduction of wp.sanitize



6. Avoid “dangerouslySetInnerHtml”

HTTPS://GITHUB.COM/WRAKKY/REACT-HTML-PARSER

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

This is one of my favorite function names because it so clearly reminds you that you are probably doing something dangerous.

If you have started using React, sooner or later you will come across this function. You might use for something as simple as displaying a post title if you want to support HTML entities like smart or typographic quotes

dangerouslySetInnerHTML is React's replacement for using innerHTML in the browser DOM. Any javascript code in the content you are inserting will get executed, potentially exposing your users to a cross-site scripting (XSS) attack.

Consider an alternative like react-html-parser to add support for entities encoding. And instead of letting users drop scripts into wp-admin, decide which scripts you want to support and add shortcodes (or gutenberg blocks) for editors to insert specific scripts

7. Use JavaScript for good

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

We can help users be more secure with JavaScript.

Log out Inactive users

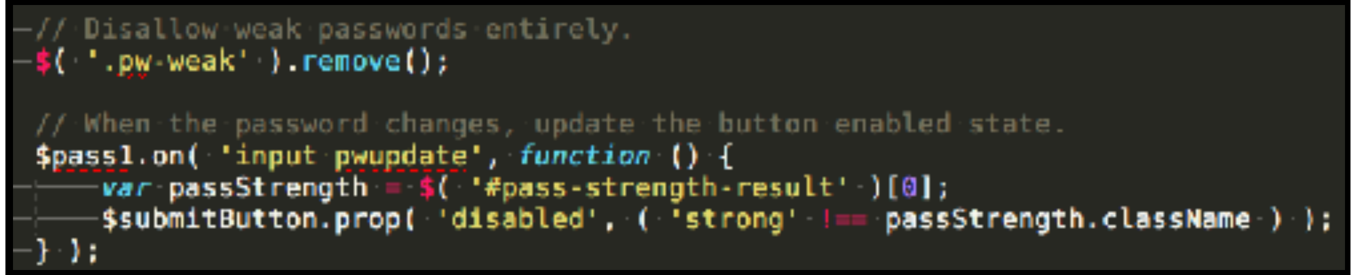
Password

Send User Notification ☑ Send the new user an email about their account.

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

a good example of this is the password strength meter and password auto-generation in WordPress core

this is nice because it helps users out by offering a secure password to start, then if they start typing their own password, it encourages them to come up with a secure password by indicating the strength and forcing them to check a “Use Weak Password” check box if try want to use a easily cracked password. once they get to a medium strength password, the checkbox goes away.

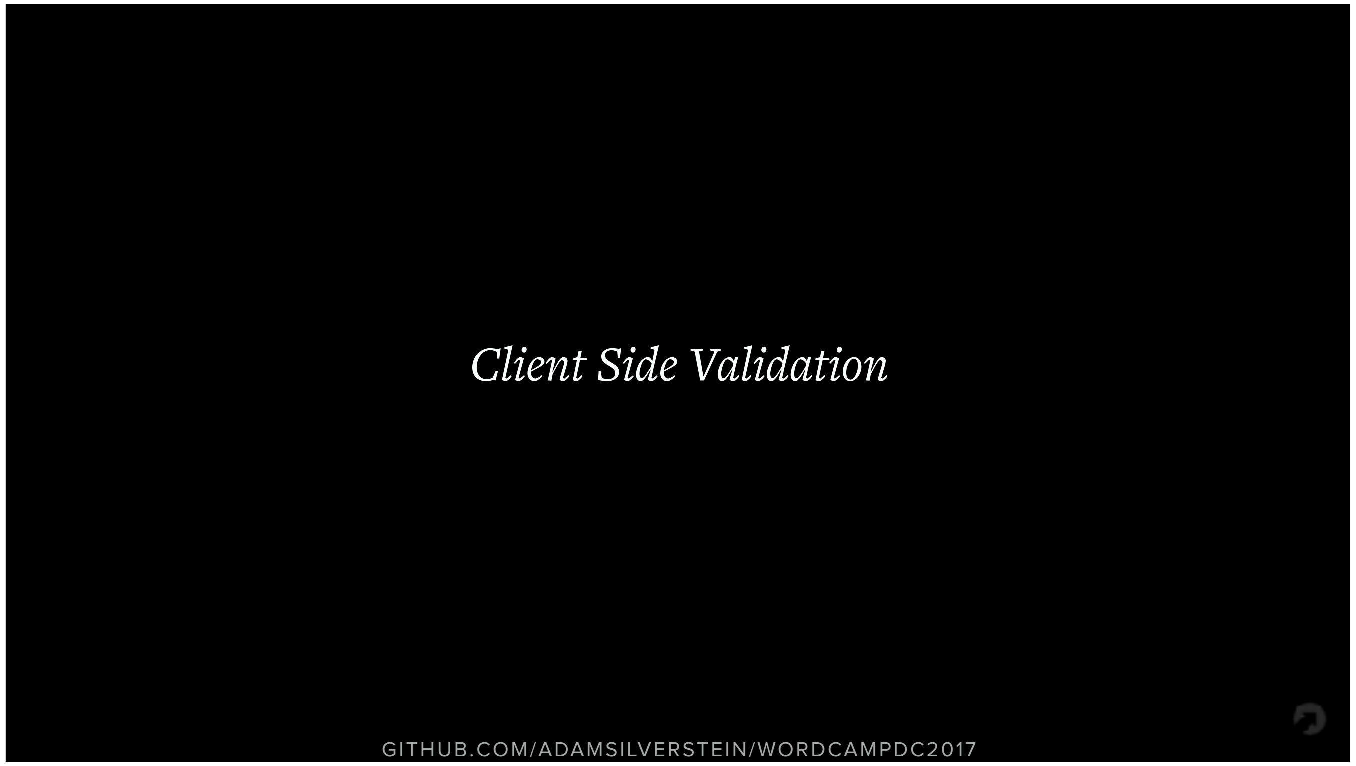


```
// Disallow weak passwords entirely.
$( '.pw-weak' ).remove();

// When the password changes, update the button enabled state.
$pass1.on( 'input pwupdate', function () {
	var passStrength = $( '#pass-strength-result' )[0];
	$submitButton.prop( 'disabled', ( 'strong' !== passStrength.className ) );
} );
```

DISALLOW ALL BUT STRONG PASSWORDS

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

we can further lock down this feature in WordPress by adding a little custom JavaScript that alters the default behavior so that users will only be able to use strong passwords, and also so they can’t bypass the warning by checking a box. I’ve got this script set up as a plugin in the repository for this talk



Client Side Validation

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

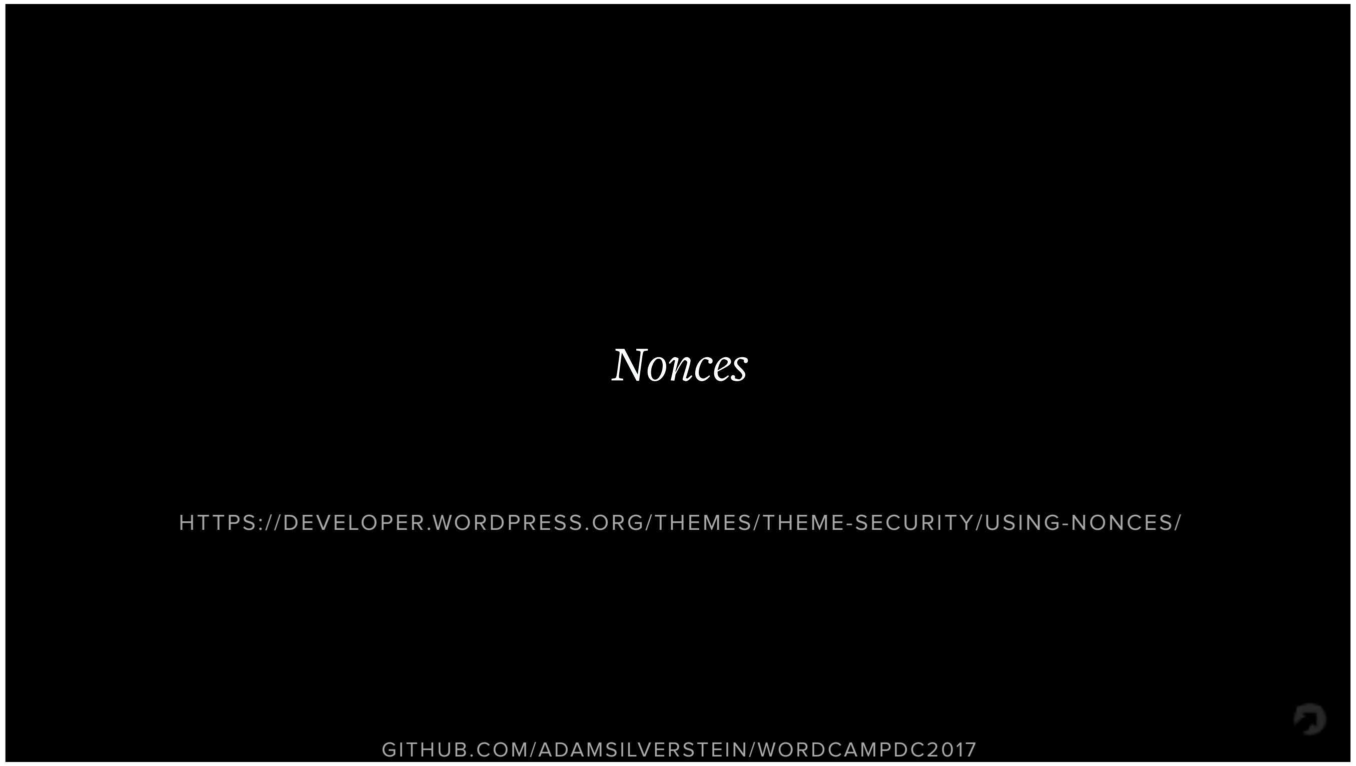
if you are collecting data from users in any form, validate it as much as possible and give them the tools they need to select the right value: color picker, image picker, display an embed so they can verify it
use a shortcode with shortcake and validate the input before accepting it.

Or, if you are building a gutenberg block you might want validate the data before saving

8. Verify intent

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

A common way we use JavaScript to help users be more secure is by confirming they are performing an action they mean to perform

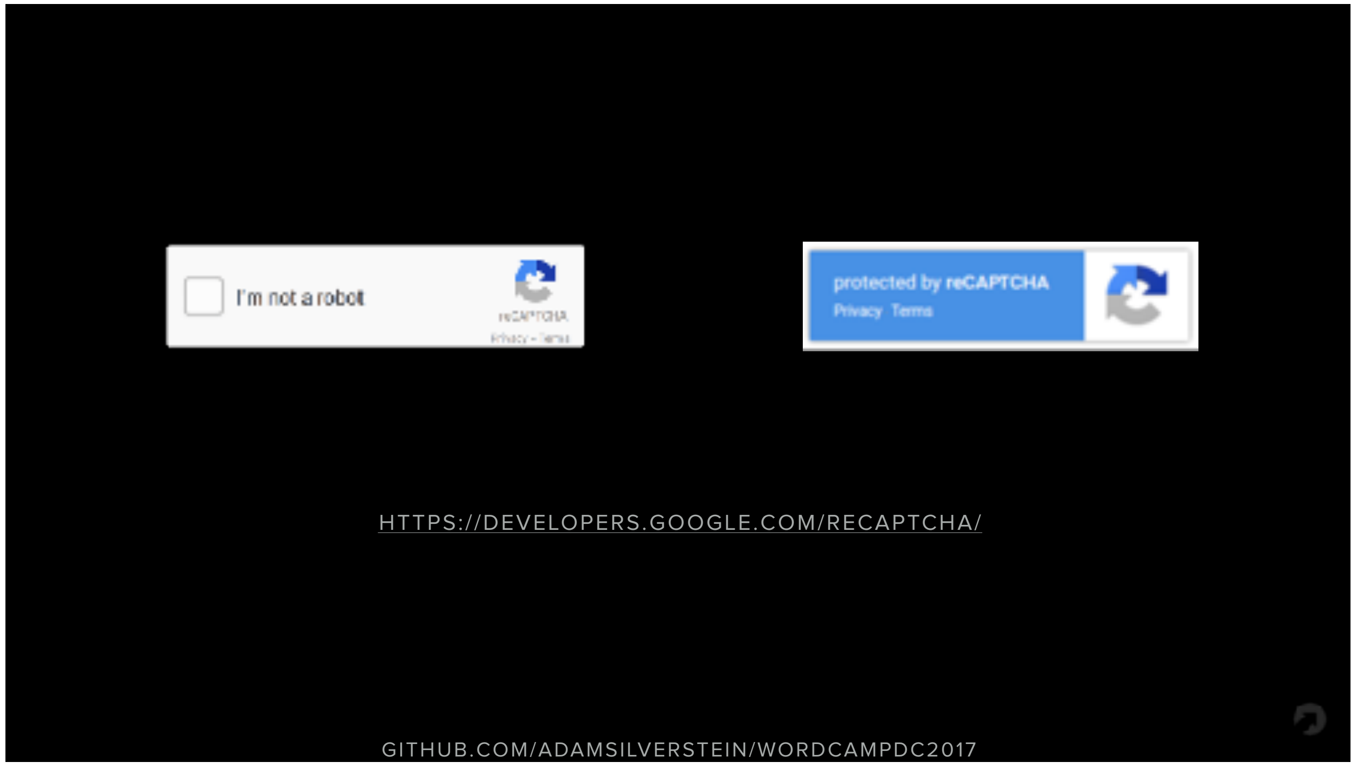


Nonces

HTTPS://DEVELOPER.WORDPRESS.ORG/THEMES/THEME-SECURITY/USING-NONCES/

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

When we connect with WordPress from JavaScript via wp-ajax or with the rest api, we use nonces to verify intent, and you should probably add action specific nonces if you are adding your own wp-ajax or rest-api endpoints. you can read more about nonces in the WordPress theme developers handbook. A quick note that if you use the bundled wp-api JS client, it takes care of adding Nonces for cookie based authentication

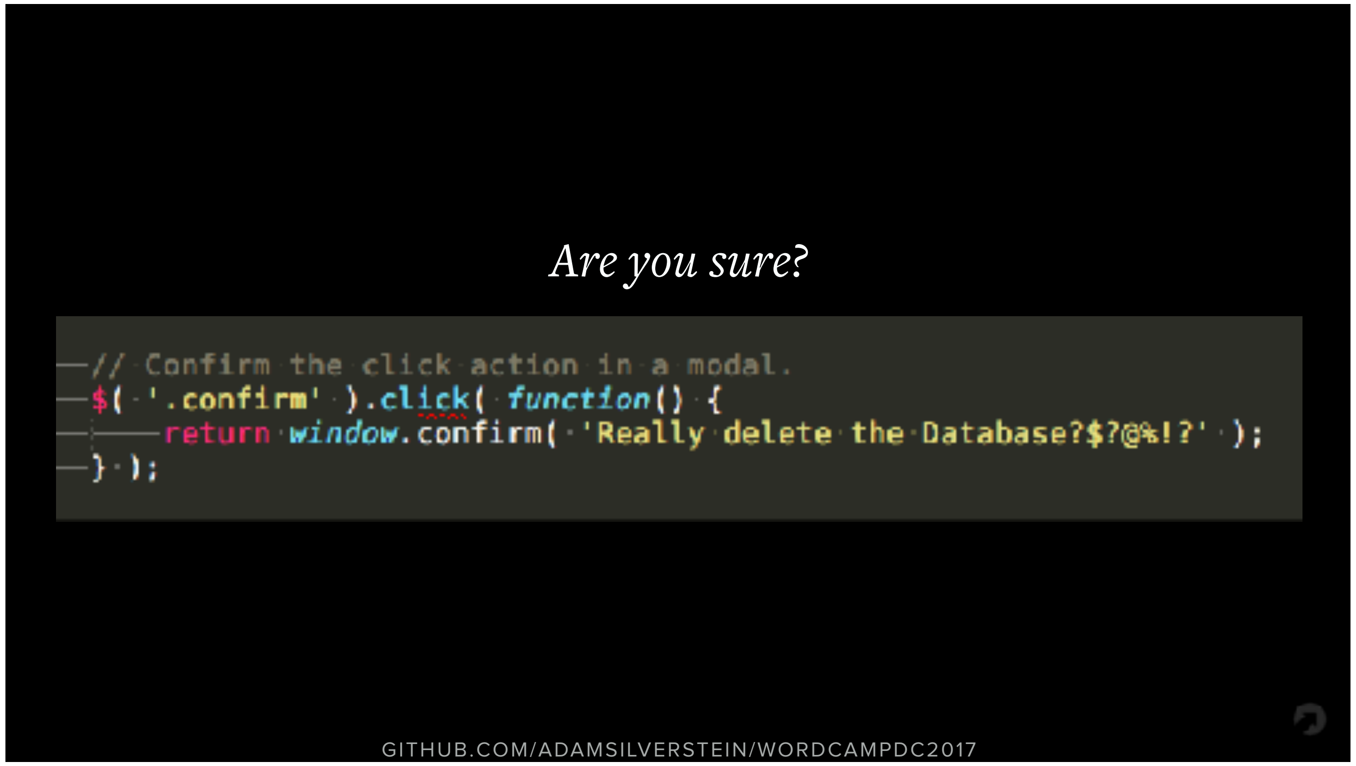


HTTPS://DEVELOPERS.GOOGLE.COM/RECAPTCHA/

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

Captchas are another way to prevent malicious abuse and Google's JavaScript based captchas are pretty cool, and do a great job blocking abuse. They even have an invisible version that you can use in your forms or admin screens when you want to ensure a user with a compromised browser can't be scripted into performing some action they never intended to perform.

Like most things Wordpress, a plugin has you covered for typical uses - enabling invisible re-captchas for gravity forms, contact form 7 or woo commerce is as simple as installing the invisible recaptcha plugin



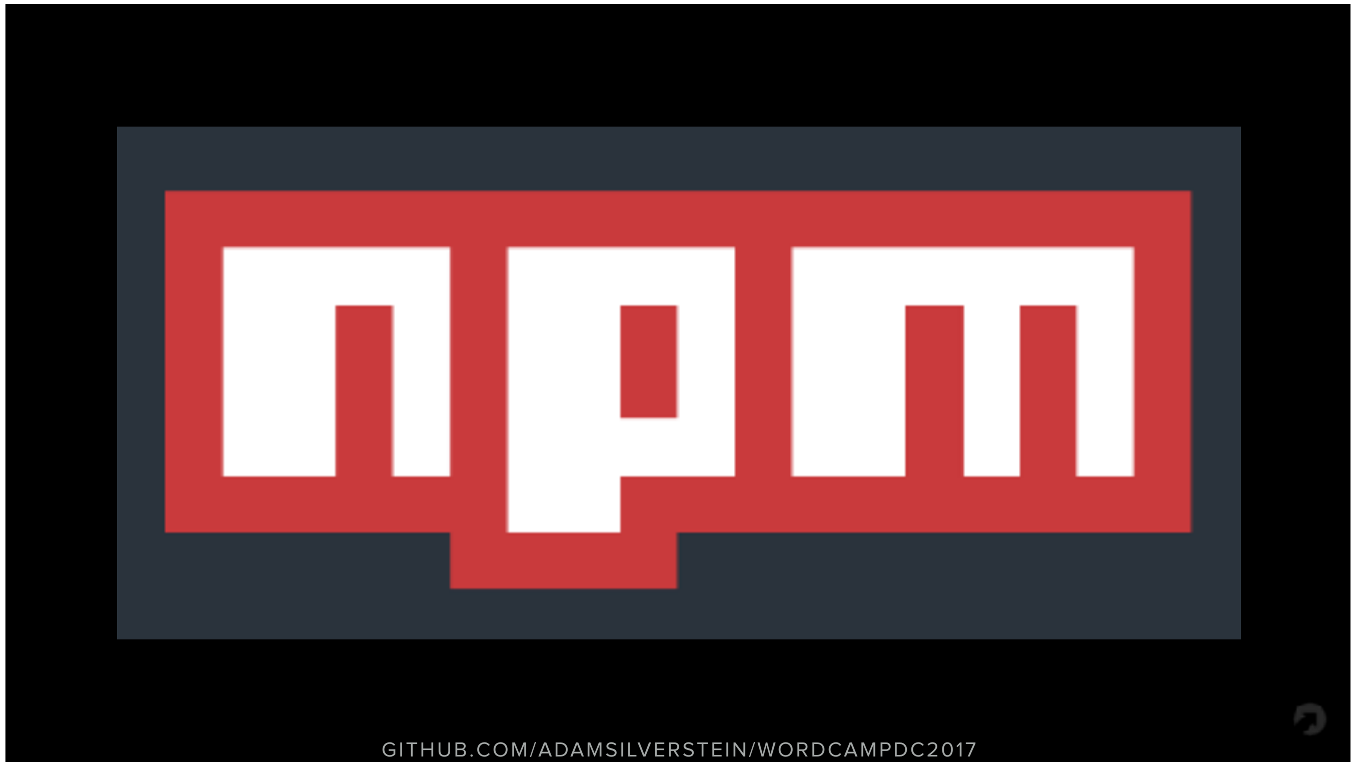
Are you sure?

```
// Confirm the click action in a modal.
$( '.confirm' ).click( function() {
	return window.confirm( 'Really delete the Database?$?@%!?' );
} );
```

Adding a simple ‘are you sure’ modal for critical changes help prevent both mistakes and with a recaptcha will also block automated attacks.

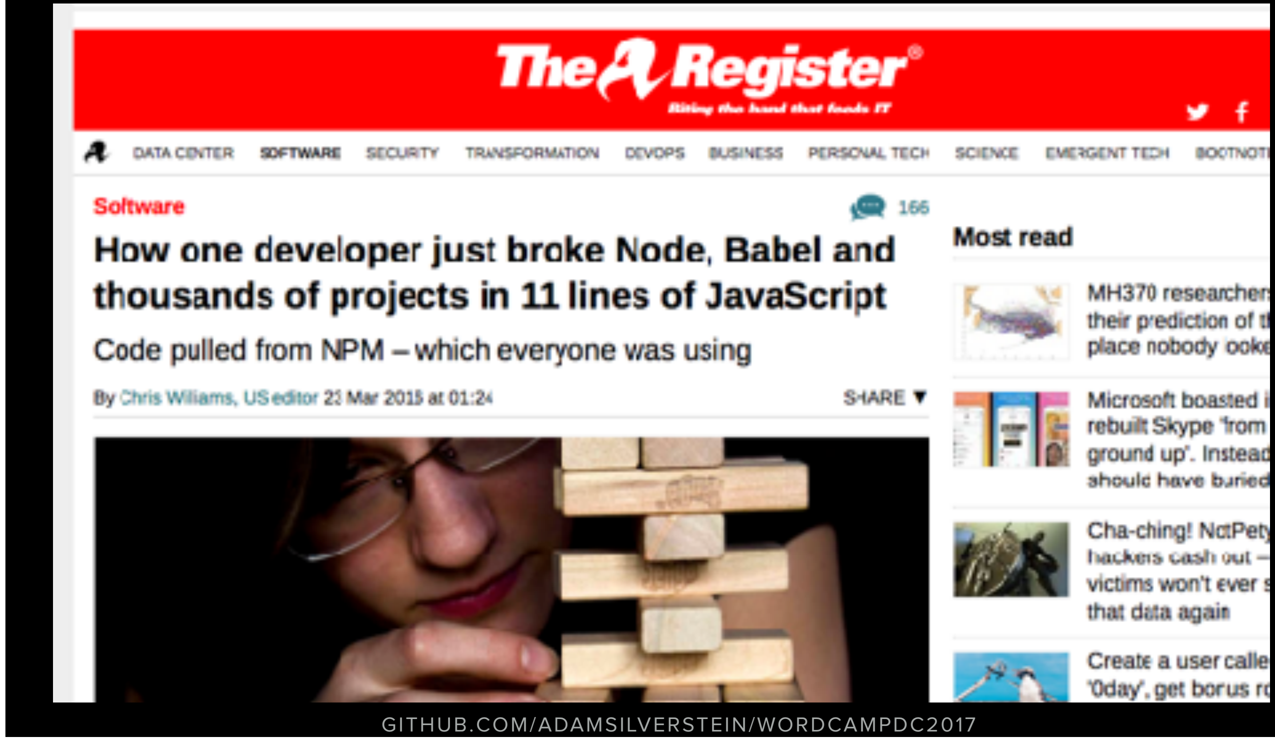
9. Minimize external dependencies

When we are building larger JavaScript apps, especially with npm at our disposal, it is so so easy to add any packages we need.



GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

just do an npm install package and boom, you can use that code in your own project - and there is a package for everything. when you need some functionality, the massive ecosystem of npm modules is just a google search away.



The Register®

Biting the hand that feeds IT

Software

How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript

Code pulled from NPM – which everyone was using

By Chris Williams, US editor 23 Mar 2016 at 01:24

Most read

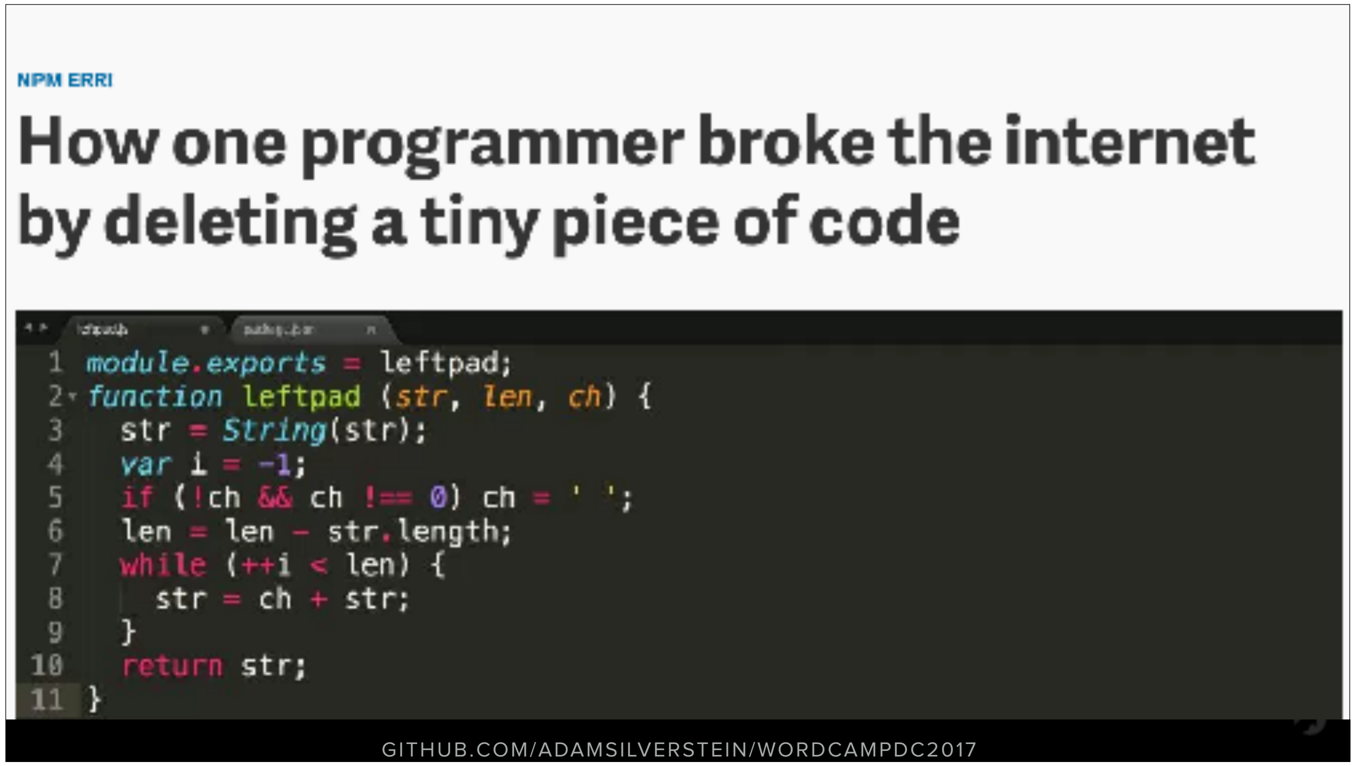
MH370 researcher
their prediction of t
place nobody ooke

Microsoft boasted i
rebuilt Skype 'from
ground up'. Instead
should have buried

Cha-ching! NotPety
hackers cash out –
victims won't ever s
that data again

Create a user calle
'0day', get bonus ro

watch out though. lock down your package versions using the latest version of npm or yarn. remember this fun incident a while back where



NPM ERR!

How one programmer broke the internet by deleting a tiny piece of code

```
module.exports = leftpad;
function leftpad (str, len, ch) {
  str = String(str);
  var i = -1;
  if (!ch && ch !== 0) ch = ' ';
  len = len - str.length;
  while (++i < len) {
    str = ch + str;
  }
  return str;
}
```

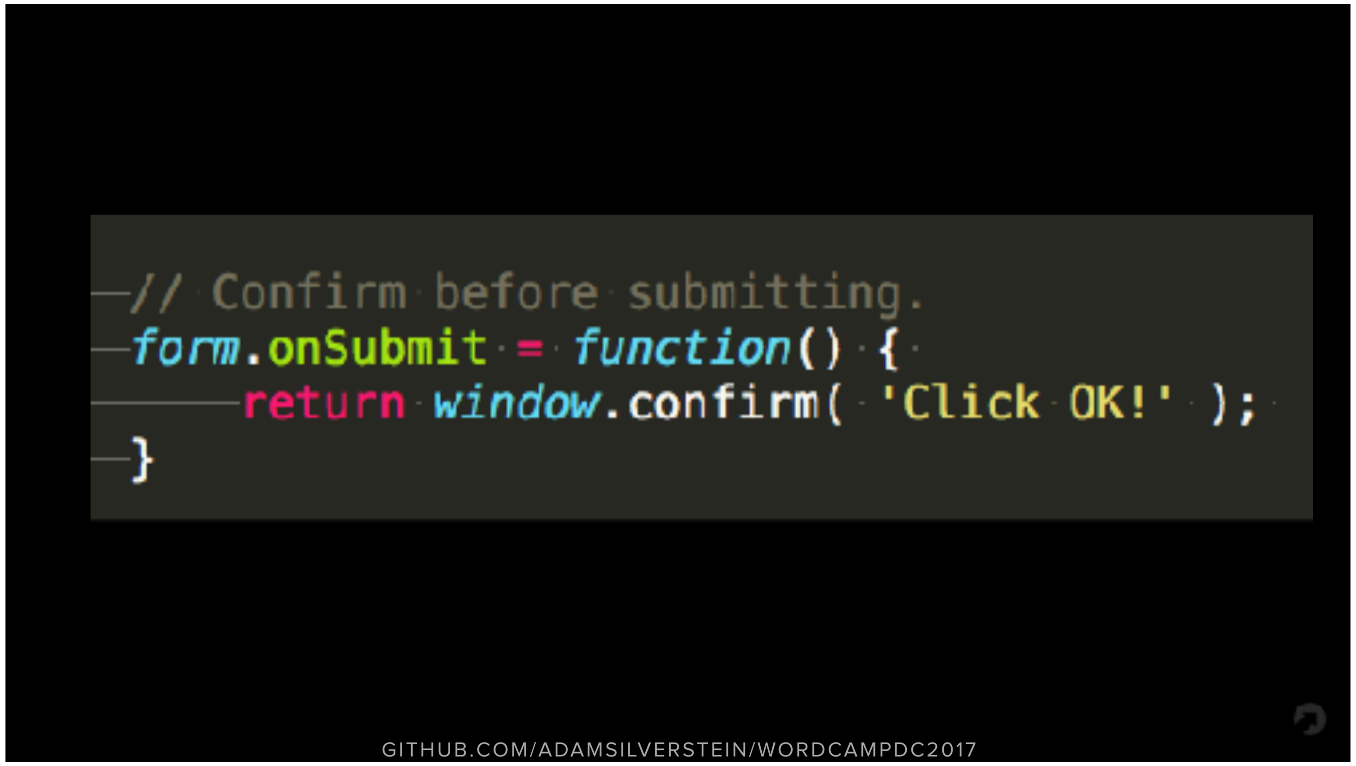
one programmer broke the internet by removing a package from the npm repository?

10. Write defensive JavaScript

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

Lets assume malicious JavaScript is running on your page, what can it do ?

- it can anything a user can do
- it can click on every link
- it can go to any other page and click on any link
- it can phone home
- it can change how core parts of JavaScript work

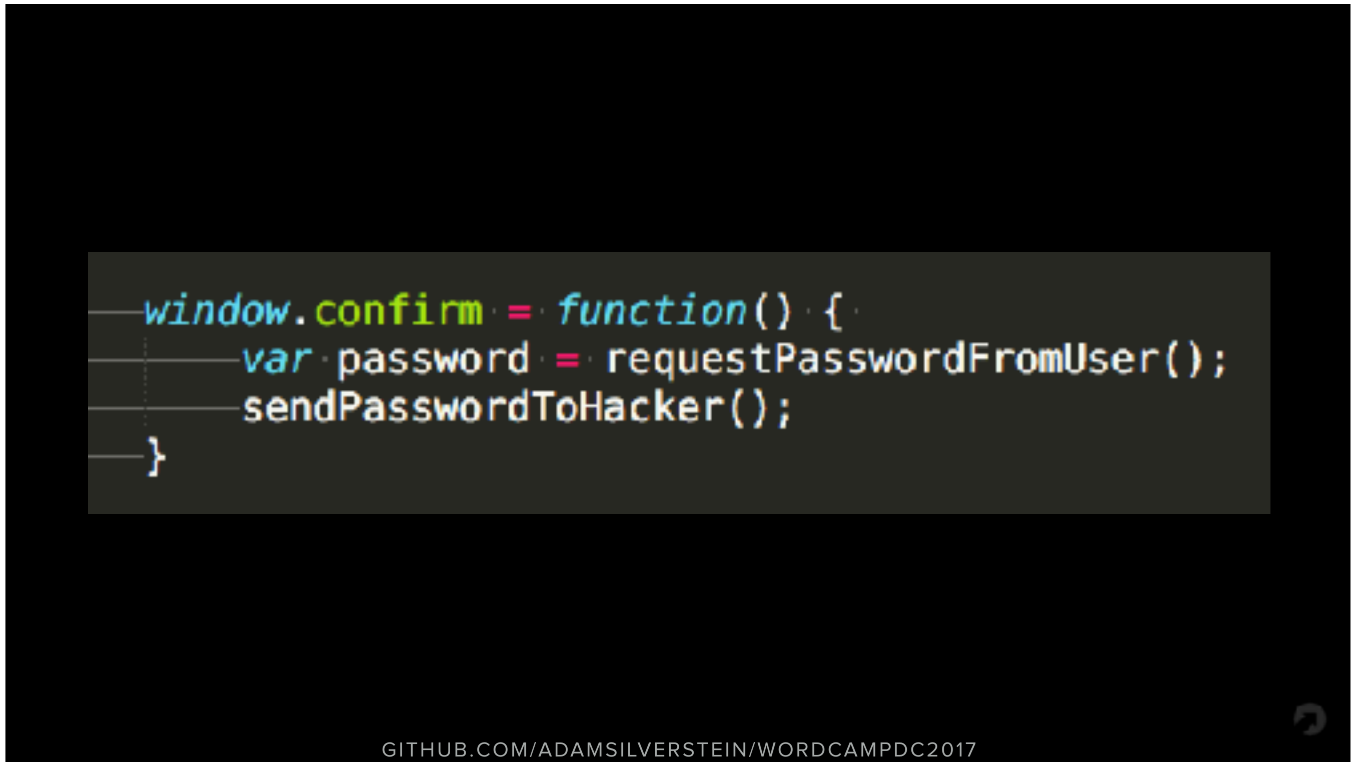


```
// Confirm before submitting.
form.onSubmit = function() {
    return window.confirm( 'Click OK!' );
}
```

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

thats right - it can change how JS itself works because JavaScript is highly mutable. Almost everything in the language can be modified.

think about a simple call like this to open a modal and get a user’s confirmation. it looks safe enough, but if a hacker has managed to get javascript running in your browser, they may have rewritten how ‘window.confirm’ works.



```
    window.confirm = function() {
        var password = requestPasswordFromUser();
        sendPasswordToHacker();
    }
```

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

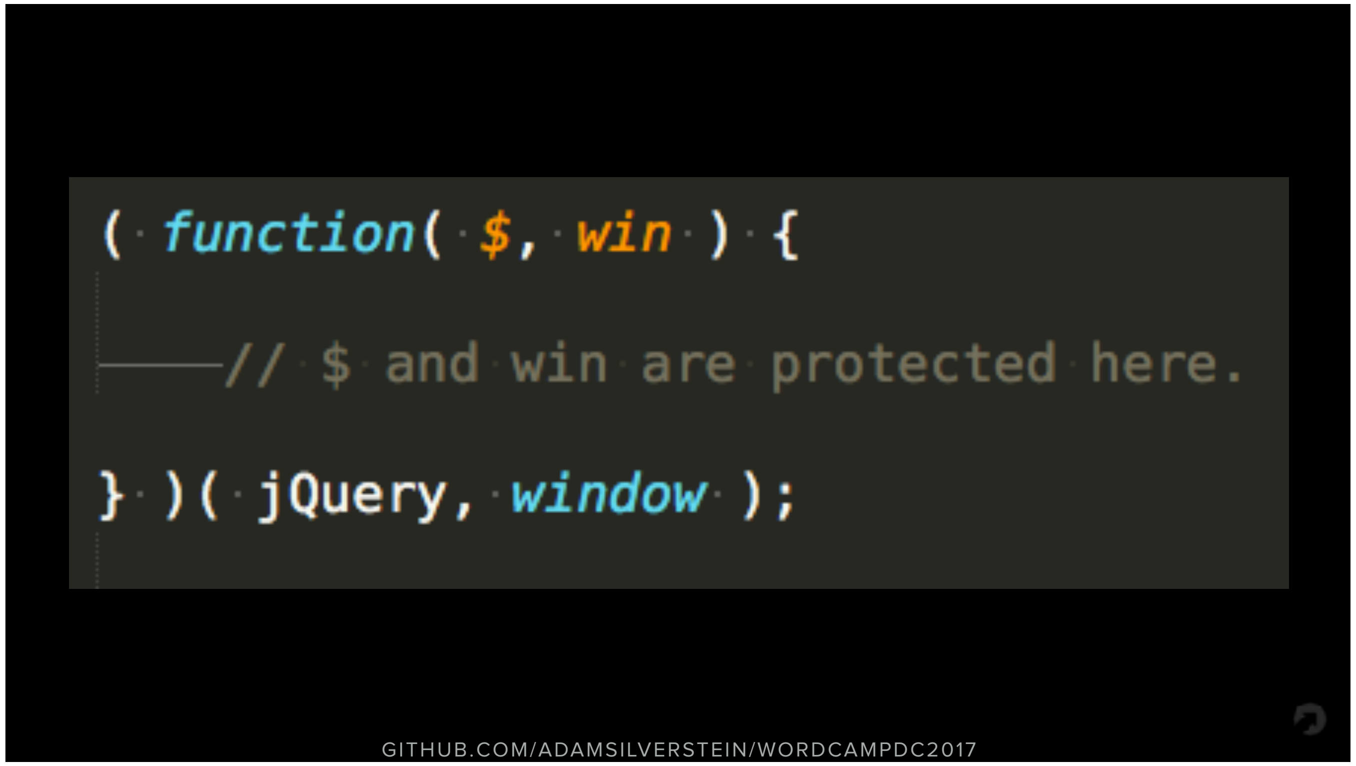
what if hackers have overwrite how window.confrim works so that it ran its own functions, asking users for their password and sending that information to a remote server.

Yes - JavaScript lets you do this kind of crazy stuff so watch out!

https://www.destroyallsoftware.com/talks/wat

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

Welcome to the wonderful world of JavaScript



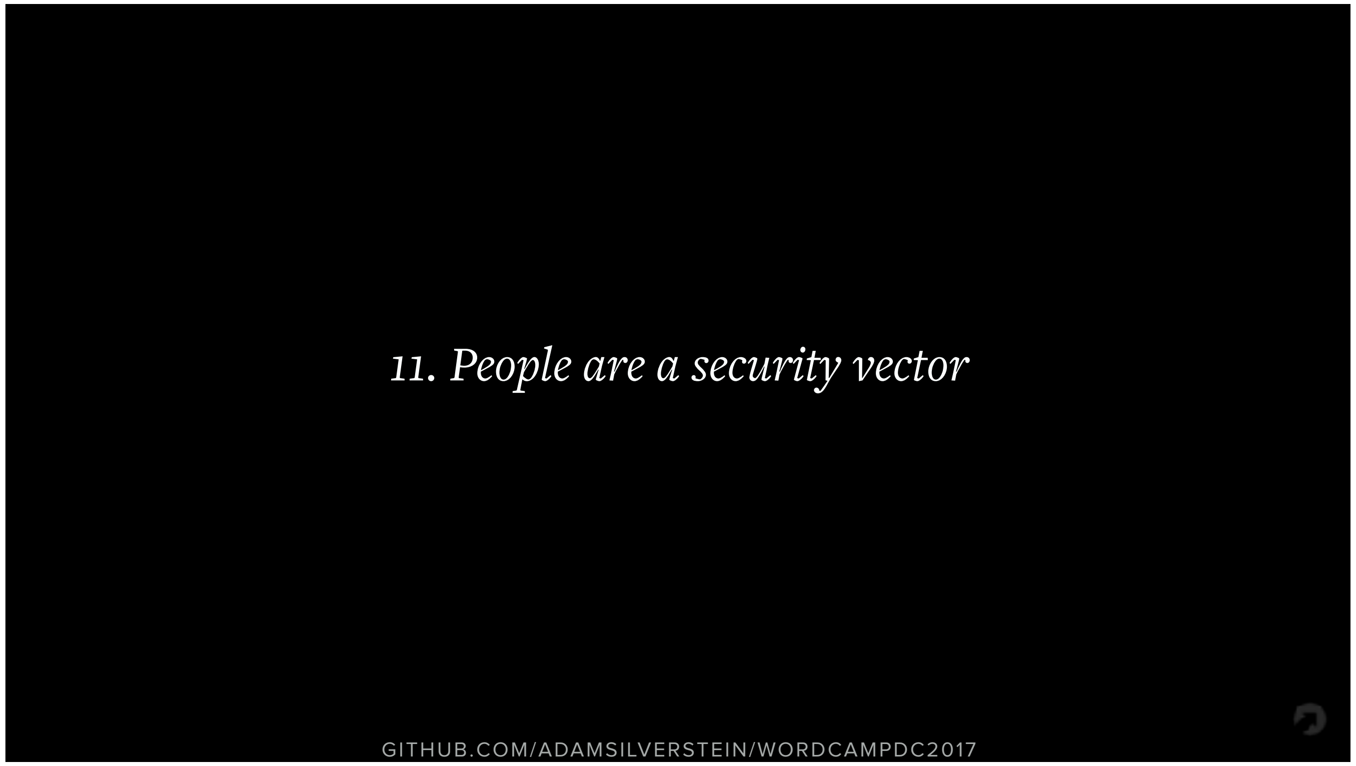
```
( function( $, win ) {
    // $ and win are protected here.
} )( jQuery, window );
```

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

To help defend against this we use a closure pattern. the syntax looks a little funny until you understand it, so lets take a look.

the very outer parenthesis ensures the entire code clock is evaluated immediately.

The inner block is a function that takes two parameters, $ and win. at the bottom you can see we immediacy call the function passing it jQuery and window. Now inside our code we can refer to $ and win and be sure that are unchanged within the context of our closure. this prevents hackers from manipulating core objects you need to use and making them work in malicious ways.



11. People are a security vector

GITHUB.COM/ADAMSILVERSTEIN/WORDCAMPDC2017

To wrap up, I have this bonus point and a story.

No matter what you do, you really can never fully trust your users to do the most secure thing possible. They might stay logged in as admin all day, or choose an insecure password instead of using a password manager - because they are lazy, right? The idea here is not that your users are malicious - rather that they are vulnerable.

A colleague of mine told me a funny story recently that illustrates this point. As a young woman she was living in Seattle and would make regular trips with her friends over to Vancouver to have some fun. Now at the time, all you needed to cross the border was your drivers license, so her habit was to stick her license and credit card into her pocket and head out.

After one of these trips, As she and her friend headed back into the US and prepared to cross the border, she searched her jeans only to realize she had changed and left her license back in Seattle. She prepared herself for a long wait as her friend drove back to Seattle to get her license. Then, as the border guard examined her friends drivers license and was about to ask for hers, something happened in the lane next to them. Suddenly 6 or 7 people were climbing out of the trunk of the car next to them, and pandemonium was breaking out at the border station. The border guard waved my friend thru and she never had to show her license.

Security issues come at you where you least expect them so- please - start paying attention to your JavaScript!

Thank you!

github.com/adamsilverstein/WordCampDC2017

adam@10up.com ● @roundearth
@adamsilverstein on #core slack

Thats it. All my links are here and I’ll be around after the session to answer your questions - find me at the happiness bar.