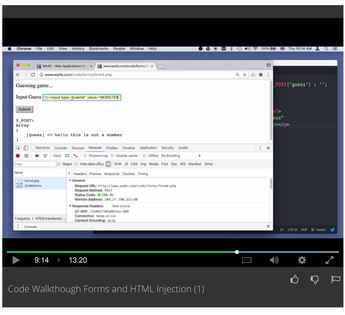
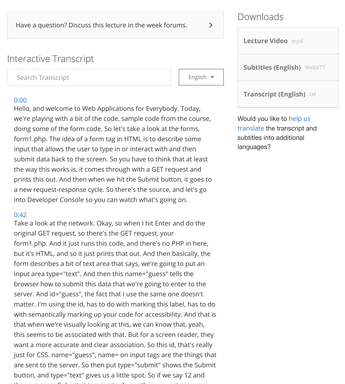
Lecture Materials





Code Walkthrough Forms and HTML Injection (1)

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript English

0:00
Hello, and welcome to Web Applications for Everybody. Today, we're playing with a bit of the code, sample code from the course, doing some of the form code. So let's take a look at the forms, form1.php. The idea of a form tag in HTML is to describe some input that allows the user to type in or interact with and then submit data back to the screen. So you have to think that at least the way this works is, it comes through with a GET request and prints this out. And then when we hit the Submit button, it goes to a new request-response cycle. So there's the source, and let's go into Developer Console so you can watch what's going on.

0:42
Take a look at the network. Okay, so when I hit Enter and do the original GET request, so there's the GET request, your form1.php. And it just runs this code, and there's no PHP in here, but it's HTML, and so it just prints that out. And then basically, the form describes a bit of text area that says, we're going to put an input area type="text". And then this name="guess" tells the browser how to submit this data that we're going to enter to the server. And id="guess", the fact that I use the same one doesn't matter. I'm using the id, has to do with making this label, has to do with semantically marking up your code for accessibility. And that is that when we're visually looking at this, we can know that, yeah, this seems to be associated with that. But for a screen reader, they want a more accurate and clear association. So this id, that's really just for CSS. name="guess", name= on input tags are the things that are sent to the server. So then put type="submit" shows the Submit button, and type="text" gives us a little spot. So if we say 12 and then we press Submit, it is going to do another request-response cycle, right, and is going to run this code again. Except that this time, it's going to add this parameter, ?guess=12, and so that's how that works. And now, inside of our PHP code, we are going to want to be able to take a look at that. And PHP has some magical variables that are super global called $_GET. And what $_GET does, let's take a look at form2.php.

2:21
Well, I'll just leave the guess=12 on. What $_GET does is $_GET takes, and PHP parses these key value pairs and the query string after the question mark and then gives us things in here. And if I put &x=42, then it would give me {guess} =>12 and [x] => 42. And so then this allows us inside of our PHP code to sort of detect the time when it's coming in with just no parameters versus parameters are being set. And so there's no parameters, and we're printing out an empty array.

2:57
And we as the developer are giving our user a easy way to send us data, and it just concatenated on. And so that's form2.php. form3.php says there are two ways to send this data. One way is as parameters. But this gets kind of ugly, and there are reasons not to do it, and you're not supposed to modify data. Now, guessing in a guessing game is probably okay to use as a GET. But all we have to do to switch to sending the POST instead of GET is to say method="post". So the form is exactly the same, constructed in exactly the same way. And we have GET parameters, and $_POST is a new thing. I'll take off this guess=12 and switch to form3.php. And so I did, that's a GET request, so if I take a look here, that was a GET request. So if I come here and I hit the Enter key, that forces a GET request to that URL. Hitting Refresh, as we'll soon see, doesn't always force a refresh. If the last operation was a POST, it redoes the POST, if the last operation was a GET. But in this particular bit of code here, View Page Source. In this particular bit of code, we have told it that we want to send the data that you entered by the user, like our 123, is sent by via POST. And you'll see that if we look at what's going on here, it's sent a POST to this URL. And if you cruise down, the data was actually sent as part of what's called Form Data in a format called URL encoded. And we could look at more detail, but the first thing you'll notice is it's not on the URL as a question mark, as a query parameter. And that's because it's actually set as part of the connection. The connection connects to www.wa4e.com on port 80, sends POST to that URL, and then it sends these request headers. These are actually part of what is sent, and then a blank line, and then Form Data is sent. The response headers are what come back to our application after the request-response cycle.

5:04
And basically, just like with GET, $_POST is created for us before the first line of our program. And it just handles this stuff, and it's all magic. You make a form, and then when the POST happens, $_POST has key value pairs that are properly set for us, okay? So let's take a look at form4.php. The one thing that you'll notice here is, my previous guess was 123, but it didn't show up here.

5:32
It's so often that forms put their old data there, 125, Submit. Where'd the 125 go? We kind of expect that the 125 is going to be there, but it's not. And it turns out that we have to just explicitly put it there. And so it's so common that developers grab this POST data as it comes in and echo it back right there so you can actually modify it. So if your guess was something long and you wanted to change it, right? It's not there, I wish it were changed. And so in form4, we show how to solve that problem. So let's go to form4 and view the source of form4, which of course is also over here.

6:16
And so what we're going to do is, we know this code is going to run once to do the GET. And that just is drawing the form, that's when I hit this Enter key, and it draws it, and it sends a GET, right? When I hit the button, because method="post", it's going to send us POST request. And the key is, is I just have this input tag, and I add value= and then this little bit of stuff. And this is a contraction for PHP to print out this variable. And the variable is, if there is data in $_POST['guess'], I'm going to take that data, otherwise it's a blank. And so if I do a View Source here, in this value here, there is no data. Because that was a GET request, the POST is not set. But when I send a POST like hello this is not a number and hit Submit, well, it comes back. And so if I now do a View Source on this page, you will see that my PHP code, particularly, put this in right here. Using this little bit right here to say, generate the HTML, but replace this little variable oldguess. And whatever was in the oldguess, which came in, in $_POST['guess'], okay? Now, the nice thing is now I can fix this. I go, I typed that wrong, and now it works. So the idea of here is a POST still, but the idea that this stuff is persistent is not something the browser does for you automatically. Sometimes, the browser does auto fill-in, which is tacky of certain fields. It does auto fill-in of those fields, in my browser, they're yellow, which is different than the server restoring the data, okay? So that's cool, that's easy. But in doing this, we just solved the first and most fundamental rule of PHP application security.

8:17
And the first and most fundamental rule of PHP security, and you can go and Google,

8:26
Cross site scripting and learn more about this. Cross site scripting, blah blah blah, it's XSS injecting, blah blah blah. Go ahead and read all that, okay? And it's nasty, and how did we just, [LAUGH] allow the user into our back yard? Well, the problem is, is this $_POST['guess'] right here is data that came from the user. Whatever I typed here shows up here. And so if I am clever, I can actually make oldguess contain sort of a snippet of valid HTML. So what I'm going to do is I'm going to make my oldguess start with a double quote.

9:08
And I'm going to say '/ and that's going to end the input tag. And then I'm going to start another input tag, type="submit" value=MONSTER. But notice that I only have the opening quote and not the close quote. Now let's put a br tag in here, or no, an hr tag.

9:25
So there's an hr tag as well.

9:27
And so when I submit this, you're going to see what happens.

9:32
So now I have this MONSTER button. And the MONSTER button is something that I created, and I could do something evil with this little MONSTER button. Now, how did that happen? Well, let's take a look at the View Source of this.

9:48
And you can see with syntax coloring, now, this part here came out of my PHP, and this part here came out of my PHP. But all this stuff, all that stuff there came from me typing. But the browser, when it sees this, it's like, no, that's not good. And so then the browser sees that as an input tag, it sees that as an hr tag. Even though this came from me, it sees this as an input tag. And this is the input tag, even though this last part of the input tag came from the PHP. It doesn't know the difference. It just can't tell, okay? And so that's really bad. Because this could do something like take your bank account cookies and send them to my evil server, if I was super clever and did this. So this is called cross site scripting, which means that I have typed something into your application. Hopefully I get someone else to stumble into my typeable-in stuff, so you're viewing somebody else's information. And then in that information, it is stealing something from you. So that's really bad, so this is terrible, terrible PHP.

10:53
Good news, not hard to fix. Let's take a look at form5.php. There's this function called htmlentities. And as long as you print out htmlentities of that value instead of the value directly, then you're generally safe. And that has to do with this. So let's go to form5.php, and let's type that same thing here, got it here.

11:20
Okay, we're going to type this same thing, and we're going to submit it, and it's going to be okay. And now if I do a View Page Source, it'll be far more clear as to what is going on. So this part came from my PHP, this part came my PHP, this part came from the user. But what htmlentities did is it saw dangerous characters like double quote, greater than less than, greater than less than, single quote, that's double quote. And it turned them into these things called HTML entities. And HTML entities are a sort of ampersand form of those characters. But now, even though the browser knows that's a quote, and it shows it as a quote right here. It does not then allow this user to escape and start entering HTML into my background document.

12:12
You can actually note that this kind of messed up, because my debug output messed up, right? Because my pre tag is still printing that value out without filtering it. So be careful, because I fixed this error, I mean, this error right here, but I didn't fix that error right there. Okay, so just rule number one is, if this data came externally or at any point came from the user, just wrap it in htmlentities. You can wrap things in htmlentities that didn't come from the user, it doesn't harm it, it's harmless, but always, for user data, wrap it. I tend not to wrap everything in htmlentities so that I can kind of mentally know where I was getting user data. So that is a zip through of some of the forms, sample code for Web Applications for Everybody.

13:01
We'll talk about some of these others in an upcoming video. So again, I think, hope this was helpful. [MUSIC]

Downloads

Lecture Video mp4
Subtitles (English) WebVTT
Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?