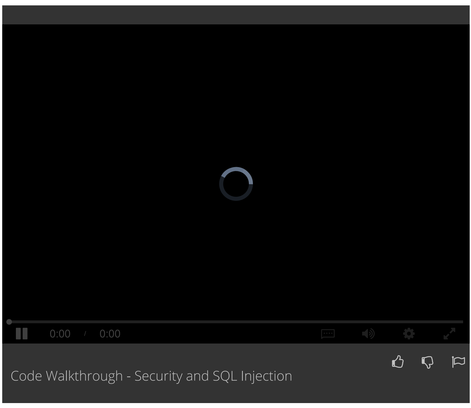
Lecture Content



Code Walkthrough - Security and SQL Injection

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

Interactive Transcript

English

0:00
Hello and welcome to Web Applications for Everybody. We're working through some of the code that we have, the PDO code, and we're in the middle of it. And the code that I want to show you right now is a code called login.php. Okay, and so here's login.php, so let's walk you through what it's going to do. Of course let's start with the data model, so here's our people. We have passwords like Chuck, 123. We should of course be hashing these passwords, but for now, we're just doing plain text passwords. And we're going to log people in. And so if it works, I can type in csev@umich.edu and password 123, perfectly fine password. csev@umich.edu password of 123 and it says, yay you're logged in. And if I say csev@umich.edu and I say password of not 123, It says no, you're not logged in. Isn't this cool? So let's see how we do this. Login1.php, so require1.p, this is the post code. And we'll just leave this for the moment. We'll come back to that in a second. And then I put out this form that is just to take the.

1:16
It simply gets the email and password and posts it. So let's take a look at what we are doing here. We're pulling in the post data. We're pulling email and password out of the post data. And look how fun we've done this. We've done this with a double quote, and we're using the variable substitution that you can, and put email equals single quote, e single quote for the email, and password. And we create the SQL. We run the query. We grab to see if there's one row. If there is one row, row is non false. If there's no row, row is false. And basically you're not logged in if the row is false. And you are logged in if you got one row and your login success. So that's how it works. All right, csev@umich.edu and 123 runs this select statement, right? And I can go run that select statement right here.

2:14
Run the select statement.

2:17
And I found a row, I've got one row. So that means that I'm logged in and that's exactly what we see. We see that we've got the record set name maps to the string Chuck. So we select name from users where email. And so login success, right? And if I do csev@umich.edu and I put in the wrong password, then it gets no row because there is no record that matches these two things, okay? So that seems like a good idea.

2:47
But it's not a good idea at all.

2:52
And that's because there is this thing called SQL injection. And so this is the classic explanation of SQL injection. It's a XKCD comic.

3:06
And mom is saying, hearing, hi, this is your son's school. We're having some computer trouble.

3:14
And mom says, dear did he break something. And the school says in a way. And then the school says did you really name your son Robert Single Quote Semicolon Drop Table Student Semicolon Dash Dash Question Mark? And mom says of course. We named him Little Bobby Tables as his nickname. And then the school says well, we've lost this year's student records, so I hope you're happy. And then mom says, and I hope you've learned to sanitize your database inputs. And the problem is just like with HTML, if we're taking data raw from the user and putting it into strings and then sending that to the database, a clever user can type an evil thing and take over. So let's try to become evil and break in to our thing.

4:01
Now what I need to do is I need to type into the password something that has a single quote in it, because then I can convince it that it's a single quote. So here's one that's pretty good. So I'll say csev@umich.edu. And I'm going to say that the password is single quote, or P single quote or single quote 1, single quote equals single quote 1. Now, again, it's just like when I was doing HTML entities, I am going to put this string there and then SQL is going to interpret the single quotes that I just sent in, okay? And so this is not the right password, but then when I submit it, look at the select statement. The SELECT statement is this is what I typed. This is the unescaped data that came from the user. So I meant for it to be that this was the password, but that's not how SQL is going to read it. It's going to say this, or that's SQL, 1=1, which is always true. So there is a record that's going to come back, and we used in the code the existence of one record as login successful and so away we go. So I just broke into this system by figuring out a cleverly constructed password to get in, okay? That's bad, that's bad. Now I wish I could show you a little a drop table, students but they made it so that this query right here won't do semicolons unless you force it to do semicolons. So the more fun thing to do is add a semi-colon in there and do a second SQL command. But they took away all the fun, so I can't even demonstrate that. So I can just demonstrate cheating and logging in without actually knowing the password. So how do we solve this? Well it's so easy. The way we solve this is we use login2.php instead. And we use PDO the way it was meant to be used. And if I put in the P' or '1' 'p' or 1=1, it's not going to work. And that's because the SQL that's actually sent has these placeholders in it. It doesn't have the actual text. We actually pass in the text to this execute. And this execute carefully escapes as much as necessary all of these user entered values. It assumes anything that's being mapped in here might be dangerous. So it's like, create a little place, don't let this data escape. Create a little place, don't let this data escape. If they put funky characters in there, do whatever. It's kind of like HTML entities and before we had PDO and prepared statements, there was a series of MySQL escape things that made the code look really nasty. Now we just use PDO and many languages have these things where they parametrize all their SQL. And so the answer to this is just use PDO prepared statements. Don't use string concatenation, especially when user entered data is present. Never ever use string concatenation to make SQL when, end user data is part of the query. And end user data is inevitably going to be part of the query.. So that's why PDO Is my best friend, and I love PDO, and I can't say enough good things about PDO. And so that is how we do a void SQL injection, okay? So, I hope this little code walk through was helpful. Cheers. [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?