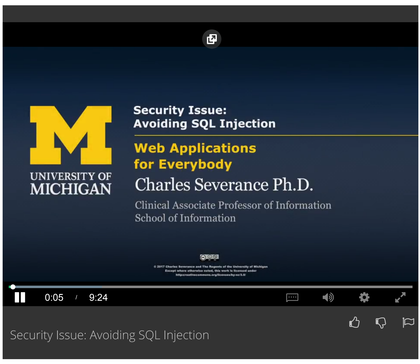
## Lecture Content



## Assignments

## Bonus Materials




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

## Interactive Transcript

English

0:08
So now I want to talk a little bit about concerns about database compromising using what's called SQL injection. And so, let's go back to what we talked about a couple lectures back, about how ill-intentioned people figure out that your code is not protecting itself and they construct something to put into a field. Okay. And in that, if you don't sanitize your html, then you can sort of escape out from under things. And if you recall, we fix that with a thing called HTML entities right? Because, there's this quote, because we didn't sanitize this user input that quote finished that and so we escaped into the background document which then rendered all of that stuff. And again, HTML entities is the solution to that. So that's called HTML injection, by using the user input and typing something really nasty. SQL injection is the same for SQL and it has to do with the user typing something into a field that you just simply take and put into a query. And if the user figures out that you've done that, then they just start doing things like putting in single quotes and or statements and and statements et cetera, et cetera. et cetera. And so the danger here is that, if you don't sanitize your user input when you use it in SQL queries you are leaving yourself wide open. And so, let's say that you were going to make your life really simple and you just wanted to use the query rather than the prepare and the execute. So, this is another way to say, check a password, right. So, we're going to say, if we got a post date, we make in a log in form, ID, password, log-in right? So, we pressed it. And so we grabbed the e-mail out and we grabbed the password out and we just make some SQL. Now, remember double quotes in PHP to dollar sign replacement, so the email goes in here and the password goes in there. And look how simple that is. And now we don't have to do prepare execute, we just do a query. And we'll just take whatever the e-mail is, whatever the password is, we put it in the query and then just run the query. This is terrible. This is very, very terrible because this allows SQL injection, because we didn't process the user data, we just concatenated the user's typed-in data right into an SQL statement. And in a way SQL injection is worse than HTML injection. And so, here we go. So let's just show how this works. So here's some accounts that we've created. So I'm going to type in, and behave this time, and I'm going to type in the wrong password, and so if you look, it puts in the email into here, in the where clause and the password in there and runs this select. And if you run the select against this data, you're going to get back no row and so you're going to correctly detect that log in is not correct. Okay. So, that's when you have a user who is nice. And there is an XKCD comic. I'll let you read this comic for a second. So, the comic starts by a mom picking the phone up and the school says; Hi, This is your son's school, we're having some computers trouble and then she goes; Oh dear, did he break something? And the school says, in a way. And the school says; Did you really name your son, Robert single quote, parenthesis, semi-colon, Drop table students semi-colon, dash, dash, question mark? And mom says; of course Little Bobby tables is what we call him. And the school says; well, we've lost this year student records and I hope you're happy. And mom says; and I hope you've learned to sanitize your database inputs. And of course, the thing is, this is just like HTML injection in that mum carefully named her son to have a single quote. And so, there's going to be some single quote and then that single quote ends it and then semi-colon used to mean, well, it still does mean, go to the next command and then drop table students, that's a legit SQL command, drop-table students, and another semi-colon and this is a comment. Okay? Now, it turns out that using PTO unless you force it to, it doesn't respect the semi-colon, so, that's good, because this is so shameful, right? So we can't quite do this. The example I gave you that is doing this concatenation is the best I can do, because PTO doesn't allow the semi-colon. I so would like it to be, I'll have this but this is so bad that they change the underlying libraries, not to allow semi-colons step through, but, we can still break in, if you do it wrong. Okay? Now, this is kind of funny and we think well, someone really named that? Here, at the University of Michigan, we have a thing called unique name, right? And my unique name is csev and I met a student who's unique name was Null. So, null turns out to be an SQL keyword. And so, it's one of those things where you're looking in a language, null is not a string when it's a keyword. And so, he would find all kinds of things where he would be registered for classes that he didn't mean. So, he actually renamed himself Null, just out of a sick sense of humor knowing that was going to cost him all kinds of misery, but, he was helping University learn to sanitize their database input. So this is not as outrageous as it seems in this cartoon. Okay? So, remember the code, right. The code basically takes the raw, e-mail and the raw typed in password, and puts it in. And you'll notice that I've got a single quote. So now, what I do, is I type in P or one equals one, with all these quotes carefully constructed, right. And so, when this goes in, it just concatenates this stuff, this is bad, concatenates the stuff. Select name from users where e-mails equals csev. This is the user entered stuff right here, that's simple enough, and password equals. Now, this is the SQL, this is the SQL, but this is the stuff I typed. But once, while SQL is executing, it doesn't know what came from the user and what came from you as the developer. And so, it doesn't know how this happened to be correct constructed. It does like; some of that's from the user, no, no, no. That's the database command that you typed and you sent to the database. And so, what it says is, e-mail is csev@umich.edu and password equals P or one equals one. Well, turns out this is always true. So this is always true, so this is always true. So, that whole thing is always true. And so now you're logged in as csev@umich.edu. Log-in success. Because that gives you back a row, that's SQL injection. Because this is lazy, lazy. I'll concatenate it all together, that'll be great. PHP, especially PHP4 and 3, this was like the biggest problem that PHP had and it really contributed a lot to PHP's being a disrespected language. And because, people who didn't know much how to program would use this and they'd use this pattern and then they would leave themselves so wide open. Erasmus Leardorff, you've probably seen a video that I put up for the class. He came and he said, the first thing he does at any campus, he walks in and starts typing these little single quote things and tries to break into every campus server. It's like, why don't you guys clean this up? So, the bad news is, before we had PTO, and this is one of the things I love about PTO, is before we had PTO there was this thing where you had to do the MySQL Scape Strings and then they had a version of PHP that was terrible, I don't even want to talk about it. It was so sad. PTO, is so beautiful, that's why I love PTO. It turns out, as long as you're using prepared statements SQL injection doesn't happen. So, any user data has to be put in through these little placeholders. And the placeholders are careful when the execute runs, the place all execute is careful to do all necessary escaping. Okay, and so you don't put quotes in, you don't put anything in, you just say, put this thing that came from the user and protect it. So, it's like having HTML entities for HTML except, it's automatic. It's all automatic. And so, this is why, if you're partaking data from the user, you want to do a prepare execute step rather than a query step, as I showed you in that last thing. It's automatically escaped, automatically handled, automatically makes it so SQL injection. So, as long as you're not concatenating it, but using prepared statements and placeholders, life is good. And that's why I like it. Okay. So up next, we're going to talk about, how exceptions get thrown and what kind of things go wrong, and what are and what are not errors in database connections for PHP.

## Downloads

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

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