CS 240: Secure Password Storage Transcript

[00:00:00] So let's talk about secure password storage in a little more detail.

Start visual description. The professor demonstrates the concept of secure password storage, explaining the importance of protecting user passwords in applications. End visual description.

[00:00:04] So many applications on the web especially, but even not on the web, many applications have some notion of a user account so that you have to log in with a username and a password.

[00:00:16] And what that means is that that application has a database somewhere that, that stores usernames and their passwords.

[00:00:24] So that when you try to log in, they can tell if, if you typed in the right password.

Start visual description. The professor shows a database table storing usernames and passwords in plain text, highlighting the security risks associated with this method. End visual description.

[00:00:30] Now, the simplest way, the most naive way that you could, could, could store usernames and passwords is just in a database table.

[00:00:36] You could, you could just have a user table for example. And for each user, you could create a row and, in that row, you would store uh the username, you store their password, you'd store any other information about the user.

[00:00:47] That that is interesting to you.

[00:00:51] And the, the perhaps the most obvious way to do that is just to store the username and password in plain text form.

[00:00:58] And when I say plain text, that means that anybody who got access to that, that user table could just read the passwords because they're not encrypted.

[00:01:05] They're not obfuscated in any way, they're just in plain text.

[00:01:08] So, in security realms, that's what they call unencrypted data, they call it plain text and it's really not secure to store passwords, at least in plain text format in a database because that means any hacker that, that breaks in and gains access to that database can then look at the user table and see everybody's password, which we, we don't want. So, we want to protect people's passwords.

[00:01:33] And so we don't want to store them in plain text.

[00:01:35] So what we do instead of storing the passwords in plain text is we store a cryptographic hash of the passwords instead.

Start visual description. The professor introduces cryptographic hashing, demonstrating how passwords are hashed before being stored in the database to enhance security. End visual description.

[00:01:42] So when the user creates their account, they're going to provide their password.

[00:01:47] And rather than storing the password in the database, we're going to run that password through a cryptographic hash function and then store the hash of the password in the database.

[00:01:57] And that way if somebody breaks in all they'll get is the, the hash values or the message digests of the, the passwords rather than the actual passwords themselves.

[00:02:07] Now, if you do this, when a user logs in, of course, they have to type in their username and password.

[00:02:12] And so what the server is going to have to do or what the application will have to do in this case is uh take the, the password that the user logged in with and, and run it through the hash function and then compare the digest of the, the typed in password to the, the password hash in the database.

[00:02:27] And if they're equal, then, then the user did type in the, the correct password.

[00:02:32] So it complicates things a little bit, but it's far more secure to store hash hashes than to store the plain text passwords.

[00:02:43] Now, um, even hashing the passwords has some, some limitations that are, are problematic.

[00:02:52] For example, if we are storing the, the password hashes.

[00:02:56] So you can see on the right here, there's a, there's a user table and uh it's got the usernames in, in plain text, which is fine, but it has password hashes.

Start visual description. The professor explains the issue of identical password hashes for different users, using an example of two users with the same password resulting in the same hash. End visual description.

[00:03:07] And the problem with doing this is that although we don't know what the user's passwords are, what we can tell is that if two users have the same password, um, that they'll have the same hashes.

[00:03:21] So in this case, Steven at the top, he's got a password that hashes to 39 E 717, et cetera.

[00:03:27] And so does Sarah. So, Sarah has a password with the same hash.

[00:03:31] So with very, very high probability, I know that Steven and Sarah have the same password.

[00:03:36] Now, that's a little bit of a security breach because if I'm trying to, to crack and, and figure out what Steven and Sarah's passwords are knowing that they're the same might, might give me some information that will help me figure out what, what their passwords are.

[00:03:50] For example, maybe I know that Stephen and Sarah are married and so if they have the same password, well, the password probably relates to something they share in common. So, so maybe it's part of their, their home address or maybe it's uh, it relates to their children's names or birthdays or something like that.

[00:04:08] So this is actually not a good thing that I can tell when people have the same password, even if I don't know what their passwords are.

[00:04:16] Um Another reason that that storing password hash hash is, is not all that secure is that hackers can still figure out what the, the passwords are pretty easily if they have a precomputed dictionary of password hashes.

[00:04:34] So what, what um hackers will often do is they'll, they have precomputed dictionaries of common passwords and their, their hash values.

[00:04:46] So just imagine somebody took a whole bunch, you know, millions of, of passwords that people commonly use and then they run them all through the, the cryptographic hash function.

[00:04:56] And so now they have this big file that maps passwords to their, their digest.

[00:05:01] So if I have a precomputed uh dictionary, that, that looks like that if I can get access to the user table in a database, I can still pretty quickly figure out what people's passwords are just by using this pre computed dictionary.

[00:05:16] Now, in this case, I'd have to take the, the hash values of the passwords and look them up in my dictionary.

[00:05:20] But, but I can easily do that, and I can map those hash values back to the passwords.

[00:05:25] And so it's really not as secure as we would like to hash the passwords. So, we need to do something a little bit stronger than just hashing the passwords.

[00:05:35] So to make this, this system more secure, we're going to employ a technique called salt.

Start visual description. The professor discusses the concept of "salt," showing how adding a random value to passwords before hashing can prevent precomputed dictionary attacks. End visual description.

[00:05:42] So what we're going to do is we're not going to hash the passwords alone.

[00:05:46] We're going to add what's called salt to each password. And so, before we, we hash a password, we're going to append to it a random value that that's called salt.

[00:05:57] And then I'll take the, the password and the salt appended together and I'll pass that through the hash function and I'm going to store the hash function of, of or the hash value of the password and the salt together and that turns out to be more secure. So, let's, let's uh look in a little more detail, um how this works.

[00:06:16] So if you're going to salt passwords, which you should do, and the first thing you want to do is for each user account, you want to generate a random value called the salt.

[00:06:26] So if you, you look in the, the user table over here at the bottom, you can see that it's got an extra column called random salt.

Start visual description. The professor demonstrates the process of generating a random salt for each user account and appending it to the password before hashing. End visual description.

[00:06:34] And so when we create a user account, we generate a random value that is that user's salt.

[00:06:43] And so that value is fixed. So, the user's salt value is, is always the same.

[00:06:48] And then we still have the username column and, and the password hash column.

[00:06:53] Ok. So now that every user has a salt value, what we'll do is when um they create their user account, we'll take the password they provided, and we'll append their password to the salt value for that user.

[00:07:07] And every user has a different salt value.

[00:07:09] So we take the salt for that user and append it to their password.

[00:07:11] And then we take the combination of those two things and run that through the, the hash function and then we store the hash of the password, and the salt combined in the database table.

[00:07:25] Now, what good does that do? Well, it does a lot of good actually because um, it solves both the problems we discussed on the previous slide.

[00:07:35] If we use salt, then if two users have the same password, we won't be able to tell that because even though Steven and Sarah have the same password, they have different hash or they have different salt values.

[00:07:47] And because they have different salt values, which means the, the hash value that we store in the password hash column is they're going to be different even though the passwords are the same.

[00:07:57] So it solves that problem.

[00:07:58] It also solves the problem we had with hackers using precomputed password dictionaries to, to hack into our passwords.

[00:08:06] And the reason the salt solves that is because it's really quite impossible to precompute a dictionary of passwords and their hashes because we're not really hashing just passwords. We're, we're hashing passwords plus their salt together.

[00:08:25] And because every user has a different salt value, um, it's, it's really not feasible to pre compute a dictionary of, of password hashes because you'd have to, to do that, you'd have to pre compute a dictionary for every possible hash or every pass possible salt value.

[00:08:43] And so it just makes it computationally infeasible to do these brute force dictionary-based um attacks.

[00:08:49] And so salt salting your passwords is, is essential and uh very commonly done in the industry.

[00:09:00] Now, one thing that's interesting that comes up here is, is what hash function should we use to hash our passwords? Well, you might think let's just use the hash functions that we've already learned about like Shaw 256 or Shaw 512 or, or something like that.

Start visual description. The professor talks about different hash functions suitable for password hashing, emphasizing the need for slower, more resource-intensive algorithms like Bcrypt. End visual description.

[00:09:16] Now, the challenge with these, these hash functions we talked about previously is that those algorithms are actually designed to run as fast and as memory efficient as they can.

[00:09:26] And so we obviously like our, our algorithms to run fast and using as little memory as possible.

[00:09:32] But when you're talking about password hashing, you would actually like to use a hashing algorithm that um is slow or expensive in some other way that makes it hard to run a lot of hashes in a short amount of time.

[00:09:46] So, what we're really saying here is that if we use a hash function on our passwords that is slow or slower, it's costly to run it, then that's going to make it harder for people to hack into our password database because they're not going to be able to, to loop through a bunch of passwords and calculate their hashes very efficiently.

[00:10:05] And so it, it's actually to our advantage to use different hash functions um than previously discussed when we're, when we're hashing passwords.

[00:10:17] Now there's um several, I mean, there's a lot of these, these password hashing algorithms out there in the world, but there are a few that are most commonly used today.

[00:10:29] And so those are listed at the bottom of this slide. So, we have Argon two, we have S crypt, and we have B crypt.

[00:10:36] Now, in the chess project, you use the B crypt algorithm to hash your user passwords.

[00:10:43] And so the reason we selected B crypt is it's one of the most popular algorithms for doing this.

[00:10:49] And so B crypt is a, a nice algorithm um that people use a lot, but there's other choices and there's always new ones being created.

[00:10:58] So stay tuned and just stay current on, on what people are doing out there.

[00:11:06] All right, let's look at a pass password hashing example.

[00:11:10] So let's go back to Intel.

[00:11:18] Now, if you did the phase four of the chest project, you've already implemented this code in your, your chest server.

[00:11:27] So when you, you hash your user passwords, so let's pay a little bit more attention to what this code actually does.

[00:11:34] Um We're using uh a library that implements the decrypt algorithm.

[00:11:42] And the first thing we do to encrypt a password or to hash a password is we create an instance of the B crypt password encoder.

[00:11:53] And then um the password we're trying to hash is, is, is too many secrets.

[00:11:59] And so to hash that all we have to do is call the encode method on the, the B crypt encoder and pass it the password that we want to hash and that'll give us back the hash value.

[00:12:10] So that's simple enough with one line of code. You can, you can securely hash a password.

Start visual description. The professor shows a code example of using the Bcrypt library to hash passwords, explaining the steps involved in creating a secure hash. End visual description.

[00:12:15] Now, the thing to know about this is that um what B crypt actually does when you encode a password is it doesn't just hash the password.

[00:12:24] What it also does is it, it generates a random salt value.

[00:12:30] So what it actually does is it, it hashes the password appended with the random salt that it, it generates.

[00:12:39] And so what you, you get back is a string that has the hash value of the password and the, and the salt combined.

[00:12:48] And then uh the string that they return has not only the hash value of that, but it also contains the salt.

[00:12:55] So they did you know, generate a random salt when we called in code.

[00:12:58] And so we need to be able to remember what that salt value is so that we can um validate this password later.

[00:13:05] And so the value that B crypt returns actually has the salt in that string as well as the hash of the password and the salt together.

[00:13:16] And so when you're using B crypt, you don't have to have a separate salt column in your, in your user table, you can just have a hashed password table realizing that those hashes actually contain the salt values as well.

[00:13:30] Now, that's how you hash a password.

[00:13:31] The other thing you need to concern yourself with is when somebody tries to log in, how can we validate that they gave us the correct password.

[00:13:39] So, um in this case, you know, we have three passwords that somebody might try to type in C A too many secrets and password.

[00:13:47] And the only one that's valid here is, is too many secrets because that's the one we, we hashed up here.

[00:13:53] And so if we loop uh through those passwords, what we'll do is we'll call the, the matches algorithm on the B crypt encoder.

[00:14:02] And so B crypt is actually going to verify for us that the password matches the hash in our database.

[00:14:09] And so what we're going to do is we're going to take the password that the user typed in.

[00:14:13] We're going to also take the hash value that's stored in our database for that user.

[00:14:17] And then B crypt will, will go through its algorithm to, to rehash the password that they gave us and compare the, the new hash with the existing hash, make sure they're equal.

[00:14:28] And so you can kind of see here that if, if we run this program.

[00:14:34] So if we run that, you can see here that um the hash value for the too many secrets password, um has several components to it.

[00:14:45] It's hard to tell what it is. But if you, if you read the documentation, part of this string is the hash and then part of it is the salt and then part of it references other things.

[00:14:56] But you can see here that in the output that um it was able to tell that too many secrets was the right password, and that cow and password were not right.