Database Security
Lesson Introduction

- Understand the **importance of securing data stored in databases**
- Learn how the **structured nature of data** in databases **impacts security mechanisms**
- Understand attacks and defenses that **specifically target databases**

We've been talking about the importance of protecting sensitive data. Actually, a lot of our sensitive data is stored in databases. When you hear about a major data breach, chances are that a database was compromised. So, in this lesson, we're going to explore issues that are unique to databases. How do we do access control for data that is stored in a database? Are there new kind of attacks that are possible, and if yes, are there new defenses that we need to use to protect the data that is stored in databases?

Importance of Database Security

Why securing data stored in databases so important and different?

- Databases store massive amounts of sensitive data
- Data has structure that influences how it is accessed
- Accessed via queries or programs written in languages like **SQL (Structured Query Language)**
- Transactional nature of queries (updates or reads)
- Derived data or database views

Databases must store sensitive data, and there must be something different about securing data that is stored in a database compared to what we've been talking about before.

So what we discussed before is data stored in files, for example. When we did access control we said you have read/write permissions on files. Databases we're going to discuss are somewhat different. They also store data, but it's different and the fact that they store lots of data that is sensitive, that's what makes this topic important.

So databases actually do store massive amounts of sensitive data. This could be, if you are a company, could be your customer data. In fact, when we talk about some company getting hacked and large amounts of their customer data, such as social security number, or date of birth, or address, credit card numbers, when those are being stolen by hackers, databases store data that we can see needs to be accessed in a certain way.

And the reason we're able to do that is because databases can store what is called data in a structured form. And the kind of examples that I just talked about, the data that we have about customer records or health records, that data does have structure, and the structure actually influences how it's going to get accessed. So, I may ask, for example, information about all customers who purchased some number of things in the last two months. Asking this kind of a query or formulating this king of a query or a question that we want to run against the database, we make use of the structure that is there in this data to do that.

So, it really do is that think about a database storing structure data and then we can basically access it through programs called queries that we submit to it. These programs are queries written in a language that is designed for accessing data stored in databases. One common one is SQL, which stands for

Structured Query Language. So we're going to talk about accessing data from databases using SQL quite a bit.

- Transactional nature of queries (updates or reads)

This is another thing that is different about databases. Databases also store data that is persistent, like files. And of course we need integrity of the data. One way to do that is we have this transactional nature of queries of programs that we run on the database. A transaction means either the program runs completely, does everything it's supposed to do, or whatever it did gets undone. Transaction either, we have the zero or one property, either it's done completely or not done at all kind of a thing. By running a partial transaction, for example, a query could leave the database in a state where it is not consistent and that impacts the integrity.

- Derived data or database views

Finally database is another thing that's kind of different about them is that we can create a virtual database, what is called a database view. This is really derived data that's there in the database. So for example, you may want to choose a subset of the information that's stored in the database. So you can run a query against the database, and the result could be another sort of virtual database, also called a database view. And then users can have access to that view.

Database Threats Quiz

Choose the best answer.

Oracle, a major database vendor, sponsored a database security study which identified key security threats. In your view, which of the following is the biggest threat...

- ☐ External hackers
- ☐ Insiders and unauthorized users

Oracle, one of the largest database vendors, actually sponsored a database security survey. I think it was 2014, to basically ask people in various companies, organizations, about threats to data that is stored in the databases. And they list the number of threats that they're worried about, and this question is actually about, what are some of these threats, which are the most serious or the biggest threats? The two options I have here are external hackers, and the second was insider and unauthorized users.

The people who are responsible for securing the data are actually more worried about insiders and unauthorized users.

- ☐ External hackers
- ☑ Insiders and unauthorized users

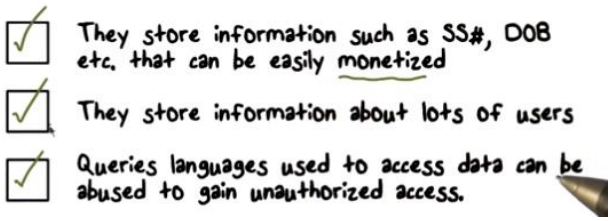
Database Hacking Quiz

Mark all applicable answers.

Databases are attractive targets for hackers because...

- ☐ They store information such as SS#, DOB etc. that can be easily monetized
- ☐ They store information about lots of users
- ☐ Queries languages used to access data can be abused to gain unauthorized access.

The next question, actually, is about why databases are so attractive, or why they are attractive targets for hackers. There are a couple of different options. And you should think about each and mark if you think that is a good reason that would explain what makes a database highly attractive to malicious hackers.



The first option should be checked. If it is attractive because you're not just getting information about a few people, but lots of people. Finally, query languages that are used to access data can be abused to gain unauthorized access, okay? So we talked about software security and vulnerable software and things like that. And, actually, we're going to see that I mentioned that SQL is used to write the programs for the queries that we run against databases, and people exploit the way we formulate those queries for example, and I will again access to databases via that. So the attack rectors, and if they make it easy for them to gain access to the data, of course that makes the target attractive because it's easy to get to it.

Relational Database Systems (RDBS)

- Relational model based database systems are **widely used in real-world environments**
- A relational database consists of relations or tables
- A **table is defined by a schema and consists of tuples**
- Tuples store attribute values as defined by schema
- **Keys used to access data in tuples**

We're going to focus on relational databases because this model is what the databases that are widely used, Oracle for example. The databases that are widely used in real world, they actually use this relational model. So that's one reason we're going to focus on it. It's also been around and investigated heavily and so on.

So what does a relational database, what's the data model here? Well, a relational database consists of what is called relations. Another name for relation is a table. Remember a table is basically, it's sort of like rows and columns. It's a metrics kind of a structure. So relation is nothing but sort of a table that has a bunch of columns, and bunch of rows.

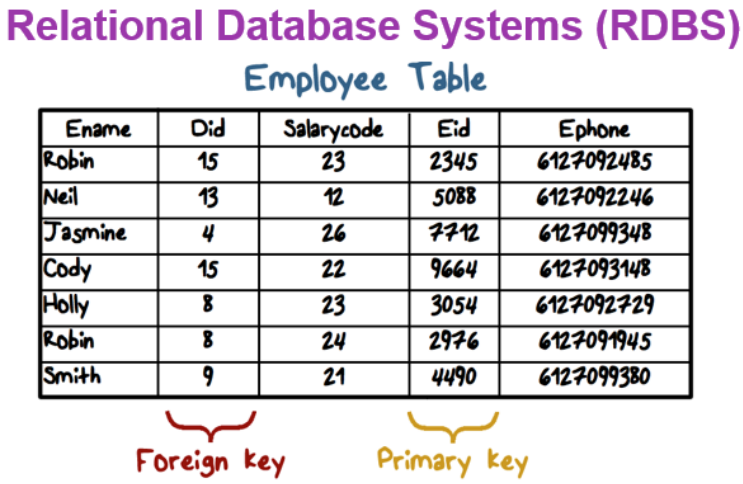
So what columns we have, what kind of data items we're going to have in the various columns, in the database or in a relation. That's defined by a schema that we have for that relation. A schema basically says what each column is. And what is the nature of the data? Is it a name, it's an address, it's a social security number, whatever it is. The columns defined by what we call attributes. So the values that we store in the various columns actually are attribute values. Think of about the schema defines what these various columns of the table are or the various attributes are. And value for a given attribute is the attribute value.

- Tuples store attribute values as defined by schema

We talked about columns or attributes, but of course, then we have rows in a table. The given row is going to contain value for each of the attributes as defined by the schema that we're talking about. So, it's column, row, cell of a metrics in our table basically has a value. So, these rows are called Tuples. So tuples basically, as I said, are attribute values that make up that row of the table. And what these values are, again, is defined by the schema that we have.

- **Keys used to access data in tuples**

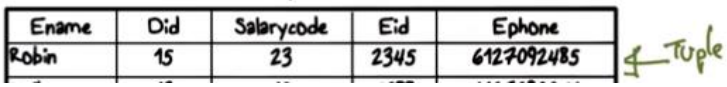
It's a table and the attribute values and the rows are called tuples and things like that. Some of those attribute values are kind of special, they're called keys. The various types of keys, primary and foreign, and stuff like that. But keys are basically values that you uniquely define a given tuple or a role. So, if the database stores information about eight employee there's going to be a tuple for each employee. Information about that employee is going to be stored in the table or the relation given employees information will define one tuple. And if you identify employees by, lets say, their social security number, then that's going to be the key value in that tuple. And it's uniquely identifies the tuple because social security numbers are different for different users.



So we're going to look at an example of a relation or a table that we just talked about. This example that we have here is- the name of this relation or table is Employee because it store Information about employees that we have, maybe in a certain company.

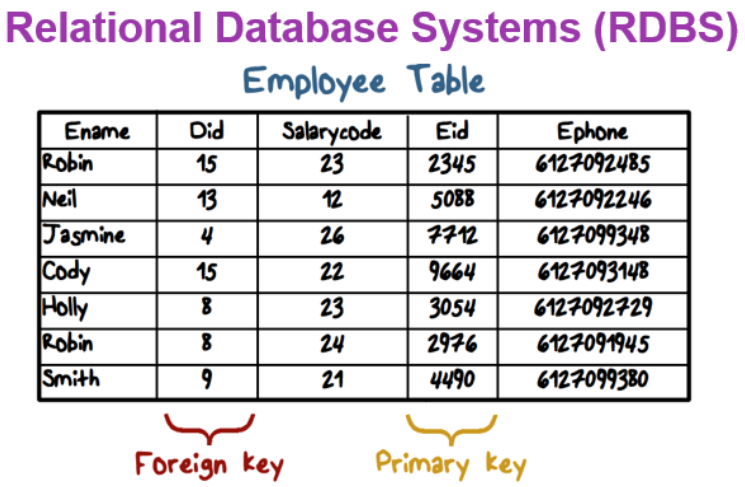
So, the columns or the attributes schema that we're talking about is defined as follows. You're going to have an employee name, going to have an identifier for the department in which they work, that's Did, the salary we don't have the value, but we have a code for the salary. We have the employee's ID and then we have their phone. The information about a given employee consists of their name, the department, the salary code, their ID, and their phone numbers, so you can reach them.

Now, in this particular organization, we have a bunch of employees. Robin, Neil, Jasmine, Cody, Holly, and so on. So for each of them we're going to have a tuple. So if you look at sort of a given row in this table here, that is what we are calling a tuple. And we uniquely identify those by the employee ID. So in case of Robin for example 2345 is the primary key value for employee Robin. So we are going to see primary keys are how we actually say what set of employees. You may want to get information about them. Things like that. So this is the key, and these are the key values that we have here.

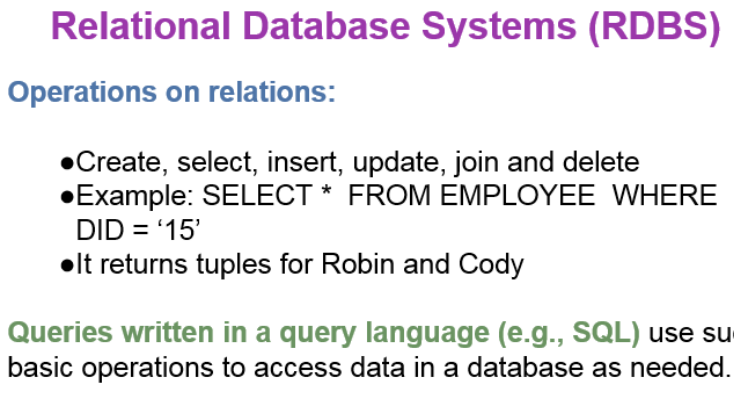


So as you can see, for each employee that information makes up a tuple in the table. The collection of those instructed organize this way. That's what makes a table. The various attributes that we have which define what's stored in this table is based on this schema that we have here.

I have something else here which a foreign key and the reason I included that here is because sometimes you may want to do operations across different tables. So foreign key basically is going to be key in some other table, and we can do what is called joins for example. Put data together from multiple tables. Well you can say, this is the foreign key that should match a primary key in another table. And then I want to pull information about Robin's department salary code and employee ID from here and her Social Security, address, other things from different table and things like that. So there maybe the department IDs that keep for example and won't be unique if you're trying to pull Social Security numbers and things like that.



So foreign key is essentially a primary key somewhere else and somehow, sometimes you may have to sort of combine information from multiple tables and we use these keys for doing that. And so some of the fields may be the attribute values that we have here. You're not going to use them as the key for accessing Tuples from here, but when you referred another relation and operation across multiple tables, that's where they get used. Again, if you've done your database scores, you probably have learnt a lot more about it than what we need here.



Now that we have seen what a relational database looks like, we said it's a collection of relations or tables, and we looked at a table, and tuples, and at the various attributes and so on.

So, what kind of operations can you perform on tables? Of course you can create a table. We're talking about selecting certain information out of a table of relations. So for example, I can say everybody who is in a certain department, we can read that information out of that relation. That's what a query selection will do, and we going to see examples of that will return. So you can select information, you can of course insert tuples, or rows, you can update attribute values that are in there. You can actually perform joint operations across multiple tables and, of course, you can delete tuples, delete tables and eventually delete a database that may be a collection of tables. You know, create, delete are sort of similar to what we have seen before, for example, files. But some of the things like select, insert, join, these are kind of new to how you can manipulate and access data that is stored in a data base.

What I do for, we said we write queries or programs to access data that is stored in a data base and those programs or queries can actually make use of these operations that we're talking about. So this is saying select, star means everything, so all attribute values from the employee table that we had where the department ID is 15. So we are saying basically tell us, give us the information about every employee who works in department that has ID 15. So this is an example of how I might access information that is told in the relation or the table that we just saw. And this is an example of a read query, because all we're doing is reading selected pieces of information that is there in that table. And returning that to whoever is making this request.

- Example: SELECT * FROM EMPLOYEE WHERE DID = '15'

So actually if you are going to run this query against the table that we just saw. You'll see that the department ID is 15 for two employees, Robin and Cody. So we are going to get the tuples for Robin and Cody. In response to, when we run this query against the database that we have. So selection has happened because, of course, we had a lot more employees. Now we're only selecting the ones who work in the department with ID 15, who happen to be these two people.

- It returns tuples for Robin and Cody

And these are return of the programs, the queries that return in a language, a query language. We talked about SQL before, and we'll see some more examples of that. But basically these programs or queries that we're talking about in this language allow you to use these basic operations. That you can perform on tables of relations and you can use these operations in the programs that you write or the queries that you submit to access the data that is of interest to you.

Queries written in a query language (e.g., SQL) use such basic operations to access data in a database as needed.

Key Value Quiz

Choose the best answer:

Two tuples (rows) in a relation can have the same primary key value.

- ☐ Yes
- ☐ No

So now that we talked a little bit about the relational data model, and talked about tables, and talked about keys, and so on. So let's try a quiz. Two tuples in a relation or table, we are also calling this a table, can have the same primary key value. Is that possible or not? Is it possible to have the same exact value for the primary key in more than one tuple in a table?

The answer to this question actually is no. Remember we did say that primary key uniquely identifies a tuple. Uniquely identifies means that given primary key value, there should only be one tuple. If it's your social security number or your employee ID, of course, that has to be unique.

- ☐ Yes
- ☑ No

Database Views Quiz

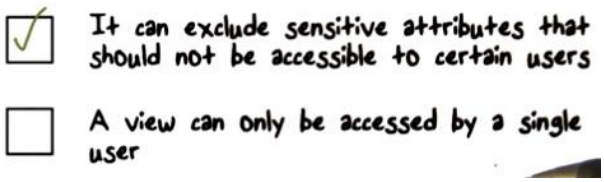
Choose the best answer.

We can use a database view to enhance data security because...

- ☐ It can exclude sensitive attributes that should not be accessible to certain users
- ☐ A view can only be accessed by a single user

So a view is a virtual table. It's a derived table from data that is in table that you store. You can run a query on it, and actually can produce set of tuples, maybe certain attribute values have been removed, or something like that. And that table is what is called a database view. So, the question here is saying, we can use such a virtual table, or derived table, or database view, to enhance security, because one of these two reasons. So choose the answer you think is the best one.

Let's look at the answers. The first one says we can exclude a query that we run to derive the data from a table that we have in the database, could exclude certain sensitive attributes like social security number or something like that. And users can only be given access to the view where this sensitive data is not there. This is a new derived virtual table we're talking about, so it doesn't have all the attribute values, in particular the highly sensitive ones. So this answer is the correct one. The next one is not correct because it says view can only be accessed by a single user. We actually didn't talk about that, doesn't have to be. Have a virtual table, or, it depends on what sort of access control you have in place, but it doesn't have to be accessed by a single user.



Database Access Control

Two commands: GRANT and REVOKE

```
GRANT                {privileges | role}
[ON                      table]
TO                       {user | role | public}
[IDENTIFIED BY password]
[WITH                GRANT OPTION]
```

Example: GRANT SELECT ON ANY TABLE TO Alice

Now that we have an idea of what a database is, in particular, a relational database that we been looking at, well, now we want to go back to what is relevant to securing it. What are the things that we have talked about?

Remember, we are going to have users who want to access the database, we would have to do authentication so we know who the request is coming from. And once you do authentication, then comes access control. So, let's talk about how access control is done in databases.

So actually, the two basic commands and different variations of SQL. Details may differ but we are basically talking about the key concepts. So you can grant or revoke. So this is sort of an example of how access could be granted. So, this says grant either set of privileges. So, for example, selecting some tables. So, privilege may be to do a select, for example. So you can grant a privilege on a table or you can grant a role to a certain user. That could be in the context of a given table. That's the resource we're talking about. This is what you can do with the resource. This access has been granted to a certain user, it could be granted to a role. We talked about role based access control, so if you have that it could be

granted the role. Public is granted to everyone. So this is the world, every user that we have in the system.

So some of these things are optional, for example what's in this parenthesis here, so if you don't specify a table, that's basically saying on all tables. Okay, so you can either specify a particular table or if you don't then this is optional part here. If it's omitted, then that would mean all the tables.

Similarly there's an optional thing here that says identified by password, so that means if you ever revoke this access that's being granted as a result of this. You will need the password to do that. So somebody else can't revoke it if they don't have the password.

And another thing that you can do here which is also optional is, that whoever you granting this access to. The user or to the role, they can actually further propagate, so that's the GRANT option, saying, well, I give you access, you can give this access to somebody else, if this is specified. If you have the access with the GRANT option.

So an example of this would be for example a statement that says you grant the select privilege on any table, so we are omitting this choosing a particular table, because this is optional, to, the user part here is Alice. Remember we didn't include the grant option here, that means Alice will not be able to further propagate this select access that she has on any table to somebody else. We're going to talk about securing data, of course, you have to worry about access control. So access control, to make good things happen, of course authorized users need to have access. So this is how you can grant access to them.

Example: GRANT SELECT ON ANY TABLE TO Alice

We talked about both a mandatory access control and discretionary access control. If your company controls centrally who should have access to what kind of things they should be able to do with various databases and tables and the databases they have, then of course it'll be centrally managed access. So that would be mandatory access control. Discretionary means if you created a table, then of course you are the owner and you can decide who else can access it. But this is how access is granted.

Database Access Control

Privileges can be for operations such as SELECT, INSERT, UPDATE OR DELETE.

```
REVOKE          {privileges | role}
[ON                 table]
FROM            { user | role | PUBLIC}
```

Example: REVOKE SELECT ON ANY TABLE FROM Alice

So we saw the example of a privilege, but privileges can also be operations such as you can insert, update, delete tables and insert new tables and things like that. All the different things we said we can do under the basis. These are operations you can perform and basically access control says a certain user role is allowed to perform that operation or not.

So the other side is, the side of grant is of course you may want to revoke. So revoking is again privilege we were talking about. The example we had was select, but it could be one of the other ones or you being able to take on a certain role optionally on a table. And you take it from a user or a role or from everybody. Granting is giving someone access, revoking is taking that access back, and an example would be, revoke the select privilege on any table, so we omit this from user Alice. Granting access

rights to perform various operations you can do on a table, and revoking those, this is how we manage access control in databases.

Database Access Control Quiz

Choose the best answer.

Alice has SELECT access to a table and she can propagate this access to Bob when...

- ☐ Alice was granted this access with GRANT option
- ☐ She can always propagate an access she has

In this particular question we're saying Alice had SELECT access. So, someone has, if it's centrally done, admin or whoever's responsible for controlling access to various databases, has given SELECT access to Alice to a table. And she can propagate this access to Bob. So Alice wants to share the data she can pull out of the database through the SELECT operation with Bob. When can she do that? There are two options here. So you choose the one that you think would allow Alice to propagate this access rights she has to Bob.

If Alice was granted this access with GRANT option, only then, can she properly fully propagate it to Bob. If she wasn't granted access with a GRANT option, she cannot propagate it. So, the second option is not correct. She cannot always propagate. The access right must come with a GRANT option, for it to be propagated.

- ☑ Alice was granted this access with GRANT option
- ☐ She can always propagate an access she has

Cascading Authorizations Quiz

Choose the best answer.

Cascading authorizations occur when an access is propagated multiple times and possibly by several users. Assume that Alice grants access to Bob who grants it further to Charlie. When Alice revokes access to Bob, should Charlie's access be also revoked?

☐ Yes ☐ No

Cascading authorizations occur when access is propagated multiple times, as we said, from Alice to Bob to Charlie to whoever else. And it's just a matter here where Alice is granting it to Bob, who further grants it to Charlie. Now Alice is revoking access from Bob. So, we're saying, what should happen to Charlie's access? Remember, Charlie got access because of Bob. When we revoke it from Bob, how should we handle Charlie's access? That's the question. Question's asking, should we also revoke Charlie's access or we should not?

Well the answer is yes because remember that Charlie got access because Bob had access. When we revoke it from Bob, Bob no longer has access. And if Charlie had access because Bob did and Bob doesn't have it, then Charlie shouldn't have it either so this revocation also has to cascade. I should say that sometimes somebody can get access from multiple such grant paths. So Alice, Bob, Charlie, but it could be John granting access to somebody who grants it to Charlie. Then they're two distinct paths and then relocation becomes a little bit more interesting.

☑ Yes ☐ No

DAC or MAC Quiz

Choose the best answer.

Database access control can be managed centrally by a few privileged users. This is an example of...

- ☐ DAC
- ☐ MAC

We talked about discretionary access control and mandatory access control. So here it says database access control is managed centrally by a few privileged users. We may have used the term trusted users in the past. It's saying is this an example of discretionary access control, DAC, or mandatory access control, MAC?

Well, this is actually a case of mandatory access control, because, if you create a relation, well you don't decide who has access to it. If that was the case, then it would be discretionary access control at your discretion. But we know that database and tables that are there in the database, access control is done by trusted users centrally. So that is mandatory access control.

Attacks on Databases: SQL Injections

- Malicious SQL commands are sent to a database
- Can impact **both confidentiality (extraction of data) and integrity (corruption of data)**
- In a web application environment, typically a **script takes user input and builds an SQL query**
- Web application **vulnerability can be used to craft an SQL injection**

We said databases, we need to talk about securing access to them, because there's something different about databases, while the structure in the data and how we access it. And we talk about access control, which is sort of something that we had discussed before. So, are there sort of unique kind of attacks that are possible on databases?

So, that's what we want to explore next. What kind of threats are possible, because either the structure of the data that we have in the database, or the way we access it through the query languages like SQL, for example. So, we're going to talk about a couple of different possible attacks. The first one we're talking about is what is called SQL injection. We'll see that these are essentially someone exploiting vulnerabilities in the code that makes up the query that is submitted to a database. Query is written SQL, then it's an SQL query and attack is called SQL injection.

So what really is an SQL injection is a malicious command. This is a command this is presented to the database. So that the database is going to actually run it. Malicious because it's going to allow someone to do something that they are not authorized to do. Results, of course, are not going to be consistent with the kind of security that you're looking to provide for this database.

These kind of injection attacks when successful, they can disclose large amounts of data. We talked about disclosures of customer data and so on earlier. So if you do that, of course, you are impacting confidentiality of the database. You're extracting data from the database that is being made public or going to somebody who shouldn't have access to it. These injection attacks can also corrupt or delete the data that's there, some set of tuples in a table or something like that that could impact integrity of

the data. So injection attacks can corrupt, which is integrity or disclose, which is confidentiality, and both of these bad outcomes are possible when it kind of attacks. We're going to discuss what they are, are successful.

- In a web application environment, typically a **script takes user input and builds an SQL query**

For us to understand what these SQL injection attacks are, we have to sort of learn something else, which is a lot of times the databases are in the back end of the system. The front end is some sort of web application environment that you have, okay? And the user interacts with the web application, and presents, we're going to do an example in a minute, some sort of request to the application. All that application then translates to a query that goes out to the database. And the query is typically generated by a script, based on the user input. So user wants something done, provides input for it. There's a script that takes that input, generates a query, and then submits that SQL query, for example, to the database. So this is the kind of environment that we're talking about.

- Web application **vulnerability can be used to craft an SQL injection**

And what happens is that there is a vulnerability in the web application itself. This is web application, it's coded, it's software. So, this is software vulnerability. And SQL injection attacks, basically, exploit that vulnerability to craft this injection attack that we're talking about.

Attacks on Databases: SQL Injections

Example: Return information about items shipped to a certain city specified user in a web application that uses forms

- **Script code:**

```
Var Shipcity;
Shipcity = Request.form ("Shipcity");
Var sql = "select * from OrdersTable where
Shipcity = '" + Shipcity + "'";
```

So, just to make this idea of SQL injections concrete, and the script and the web application and so on, let's do an example.

So, the example is the database that stores data about what is shipped where and things like that. It's a collection of the orders, for example, that a company may have received. Okay, orders that, whenever an order is placed, we create a tuple.

Of course, one piece of information or attribute in that tuple is going to say where the item is going to be shipped. And this is, let's say specified by the users. User says, we're saving something, and I want it sent to the city. The way the web application supports user attraction is maybe using a form. And you fill the form with this needed information including the city that we're talking about. So, the user interacts with the form, fills in the various fields, including the city that we have here. And then, we said this web application now has to generate a query.

So the script code that generates the query is included as sort of the example here. So the code is going to be SQL query that we have to generate. So the script basically is saying from the form you read the user input, which is the city where we're going to ship a certain item. So this statement shipcity is basically saying, from the form you read this value and that's if you want to send it to New York, for example, that's what we're going to have here.

Once we have that, we're saying generate the SQL query, okay, and code. So, the query's going to say select everything from OrdersTable where the attribute Shipcity is what this input is provided to us. So we're going to add that. We'll see the resulting query is actually going to be select* from OrdersTable where Shipcity is, if you put New York City here, it'll be shipcity = New York.

- **Script code:**

```
Var Shipcity;
Shipcity = Request.form ("Shipcity");
Var sql = "select * from OrdersTable where
Shipcity = '" + Shipcity + "'";
```

Okay, so that's an example. I think the only thing that's important here is that the user, the injection attack, the way it is going to work is that somebody is going to provide bad input. So we talked about buffer overflows through bad input before. It's not quite the same, but it's kind of like that. And injection is going to occur because of bad input. So the idea here is that you are getting the input here from the user. Based on that query, and this query is what's going to be submitted to the database.

SQL Injection Example

- User enters REDMOND,
- Script generates SELECT * FROM OrdersTable Where Shipcity = 'Redmond'.
- What if user enters Redmond' ; DROP table OrdersTable; ?
- In this case, SELECT * FROM OrdersTable WHERE Shipcity = 'Redmond' ; DROP OrdersTable is generated
  - Malicious user is able to inject code to delete the table
- Many other code injection examples exist

What we just said, with the web application and the reading input from a form and then generating a query, let's do an example based on that.

So let's say this web application is running, and user is going to provide input. The input they provide is, let's say the city Redmond. So if that's the case, the script is going to generate the query that says select star from OrdersTable. If you go back and look at the code where basically it's going to replace the placeholder that we had for Shipcity by this input that was just provided by the user. The script generating this. And this is what is going to get submitted to the database.

Now let's get to what kind of mischief a user who's malicious may do. So, what if the user, instead of providing Redmond, is going to provide something that says Redmond Code, semi colon, drop table orders table, drop, this is the particular table that we're talking about. What if they entered this whole thing?

- What if user enters Redmond' ; DROP table OrdersTable; ?

Okay, so before we look at this maybe a little bit of SQL here. So semicolon separates the statement, so this statement will end here. And the next statement is a drop table, orders table and drop results in basically, assuming that you have access to do that, deletion off this particular table. So what happened here is that the user is actually entering a lot more input than what we expect, which is just a city name. So what happens when the user actually provides this as input? They're injecting this SQL injection that we're talking about, you'll see where the name now comes from. In this case the query that we're going to generate is going to be this, actually. So, what this is what we got before. This is what we're going to get now because here we're going to replace the Shipcity variable value that we read is this one.

- In this case, SELECT * FROM OrdersTable WHERE Shipcity = 'Redmond' ; DROP OrdersTable is generated

So now, essentially we're giving the database this query. And now you can see what really has happened here. The malicious user that we have is able to inject code to delete the table, and that is what we see here after the semicolon, we see a DROP OrdersTable, and that's what we generate, and this will result in deletion of this table.

- Malicious user is able to inject code to delete the table

So, this is the injection. We're injecting code through input that is normally expected by the web application. The vulnerability we have in the web application is that it's not checking input obviously. It is accepting this input for cities where the item is shipped. Getting a name of course, but it also there's lot more stuff that comes after that. It is accepting this input, and this input is not correct input when we looking for just the city name. So, by providing this input, it's able to inject malicious code to drop the table or to delete the table. This is going to impact its integrity. And that's actually a concrete example of what SQL injections are.

SQL Injection Defenses

- Input checking
  (golden rule – all input is evil)
- See **OWASP top 10 proactive controls** at https://www.owasp.org/index.php/OWASP_Proactive_Controls

So input checking is a defense that's generic for all software security. That applies here as well. And there are actually other kinds of vulnerabilities when it comes to web application security. OWASP is the non-profit open web application, security project. They have a list of top 10 vulnerabilities. They also talk about proactive controls that can help you address those vulnerabilities. And they always talk SQL injections, or SQL injections. So to avoid that of course, parameter checking, or argument checking, input checking, is something that is a good defense.

SQL Login Quiz

Mark all applicable answers.

A web application script uses the following code to generate a query:

Query = "SELECT accounts FROM users WHERE login = ' " + login + " ' AND pass = ' " + password + " ' AND pin = " + pin; The various arguments are read from a form to generate Query.

This query is executed to get a user's account information when the following is provided correctly...

☐ Login name ☐ Password ☐ PIN

In this question, we're saying in the following script is used to generate the query, so the query is going to be, we want to select account information from a table called users, and we want to do it for those users for whom we going to have the login, a password, and PIN. These three values going to be red, all these arguments going to be red from a form. And once we read them, we going to generate this query that is, this SQL query that is going to get submitted to the database, what input is necessary for it to generate the correct query. So there are three options here. Mark whichever ones you think are applicable. So it's really understanding this little script code that we have here that is used to generate the query that we going to submit.

So remember, I did say that something has to be read. So the input may be through a form where the user types in. And then we're going to use these three variables that we have. So these login, password and PIN the values that are input, those are going to be added to this query that we are producing here. So, this query to run properly, one thing to notice here is that there's an AND, so, the login value has to match. The password value has to match. And the pin value has to match, okay? We're sort of doing three-factor authentication here, if you like. So for the query to execute correctly, because of this conjunction we have, all three of the values have to be correct. Okay? So we have to have the correct login name, we have to have the correct password, and we also have to have the matching PIN for that login name. So all three are necessary. All three must be provided correctly because of this AND operator that we have. So for each each tuple that we have in the table, and the table here is users, we have to see that these three values match the values that are there in the tuple. And when they do, that tuple is going to be selected, and we're going to return the account's attribute from that. This is how the script is going to generate a query that is going to produce the result that I just talked about.

SQL Login Quiz #2

Choose the best answer.

Query = "SELECT accounts FROM users WHERE login = ' " + login + " ' AND pass = ' " + password + " ' AND pin = " + pin; The various arguments are read from a form to generate Query.

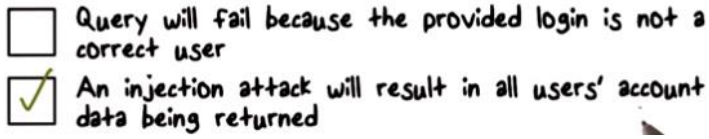
If a user types "'or 1 = 1 --" for login in the above query...

- ☐ Query will fail because the provided login is not a correct user
- ☐ An injection attack will result in all users' account data being returned

Now, we're talking about what the user is going to type in. So the user for the login field we have in the form, that's the value that goes to this variable that we have here, is actually typing something strange. So what your types is, in these codes, it's the space code, or 1 = 1. And it's important that these two dashes here that we have, they really say that everything that follows is a comment, and is to be ignored, is not to be executed. So the way to think about this is that, I'm not typing anything for password. I'm not typing anything for PIN. For login, I'm typing this information that I have here. So, if I did that, this script that we have here is going to generate a query. And that query is going to be run against the database that we have as we did before. So the quiz question here is asking, well what is going to happen when this query is run?

The first one says the query will fail because the provided login is not a correct user. Okay? So obviously what we're providing here is not, doesn't match in any tuple, the login value that is starting, restoring. But let's look at it a bit more closely. Although there isn't a match, but then we're running into this or. Okay? This is a logical expression. Boolean expression. But if either is a match, which will, you take a tuple, the way you would do select is that you look a row in the table or a tuple, you look at the login field and see if this value is going to match. So what value are we going to try to match? Well, we only match the value up to this point. There is no match here. So that's going to be false. And then we continue on this, and here we say there's an or, 1 = 1. Now this is always true. 1 = 1, always evaluates to true. So we're either saying there's a match, or 1 = 1, well together this is going to be true. So, although the match fails, but because of the or, the full expression that we have here is going to be true. And we're actually not even going to check for password and PIN because whatever we put there or did not, actually follows this two dashes that we said, actually is a common field, anyway. So the query is actually

only going to select, be that select accounts from users where log in equals this, Okay? That's what the query, that's going to be submitted to the database. That's what is going to get executed when it's submitted to the database. So the query actually will not fail, and the reason for that is that in this or expression, although the match fails, this side is going to be true, so the overall expression is going to be true. So the first one is not correct. It will not fail because it doesn't, login doesn't have to match. 1 = 1 is why this is going to be true. So the first one is not the correct answer. The correct answer is the second one. An injection attack will result in all users' account data being returned. This evaluates to true which means that tuple is one from which we should be returning that request for information, or the information that's being selected. Remember we return information that is selected from all tuples, for which this expression evaluates to true.



Inference Attacks on Databases

- **Certain aggregate/statistical queries can be allowed by all users.**
- Consider a student grade database with schema studentid, student_standing (junior or senior), exam1_score, exam2_score, final_grade.

Now, we're going to talk about a very different kind of an attack. This is called inference attacks and that can be performed or mounted against databases.

So the definition of an inference attack is actually fairly straight-forward. Inference attacks occur when somebody's able to use queries that they're authorized for, they're allowed to execute those queries. But by executing those queries, they're able to gain access to information for which they're not authorized. They're not allowed to access that information directly. So the way they gain access to that information is by executing a set of queries that are authorized, and then making an inference based on the results that are returned by those queries.

So a concrete example here, since we're talking to students in a course, is let's say a database that contains all your grades. So schema, maybe we have an attribute, that is the studentid, maybe the student_standing, whether a junior or senior and then scores. These might be numerical values between zero and 100, let's say, on exam1, exam2 and then the final_grade. The tuple, if you look at it, it's going to have a studentid, either junior or senior, their standing, the two exams, course and the final_grade. So if we consider this database, where we have a tuple for each student in the class, tuple as we just discussed, what kind of query can you allow on it which any user should be allowed to submit and get the result from it?

Inference Attacks on Databases

- Average score on an exam is a query that any student should be able to run.
- **Attacker wants to find exact score of some student.**
- **Inference attack when target takes the exam late**
  - Average score before target takes the exam
  - Average score after target takes the exam
  - **Target score can be easily found**

We're going to look at an example of inference attack with the database that we defined by the schema that we just discussed.

The query that we're going to consider here is a query that returns the average score on an exam. Okay, so any student should be able to find out. In fact, that's the first question after an exam, when you bring back the grades, students ask what was the average. So any student should be able to submit a query that returns the average, of the score, on that exam. And the average basically is, we take every student's grade, sum it up, and then divide it by the number of students. And that shouldn't tell us anything about a particular student. Remember what you're not authorized to access is somebody else's grade. This is an authorized query and our idea of inference attack is that when you use a query like this to gain access to information that you normally are not allowed to see.

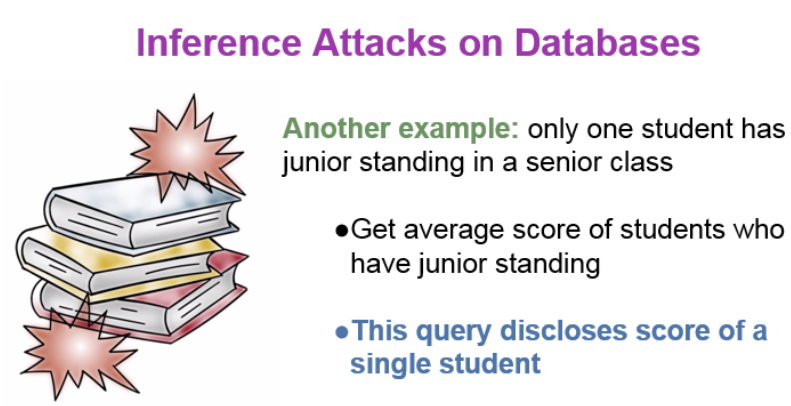
If you are an attacker, your goal would be to gain access to information that is not available to you. So that would be to find the exact score of some other student. It's not your score that you already know, but if you are an outsider, of course you're targeting a particular student, and you want to find out his or her exact score. That's the goal of the attacker or the information that we want the attacker to get access to.

Inference attacks sometimes require some additional outside or external information. So maybe the attacker does know when somebody takes, some person takes the exam late. Could be because they were sick or something like that, so the whole class took the exam on a certain date, and after a few days after that, perhaps, this student is going to take the exam. And this information is available to that hacker.

So that hacker is going to do in this case is run the average score query, before this one student, the one who takes the exam late actually takes the exam and his or her score is added to the database. So before the exam is taken, the score may be zero or something like that and after the exam, it's replaced by the correct value. So that attacker is running the average query which we said should be allowed before this one student takes the exam.

And the attacker knows when the student has taken the exam, maybe tries a few times, eventually sees maybe a different average value. Or, after a certain amount of time, we know that the exam must have been taken by everyone, including this one student who had missed it earlier. So it does the average again. Okay, so now I have two averages, and let's say this student is Alice. So the average score, not including Alice, and then average score of everyone that includes Alice. So based on these two values that are returned by this query that is allowed, I can now figure out what Alice's average is. Actually it's fairly easy to do that, if there are n number of students, the total score, if you add them all up, is n times the average. So here, since Alice hasn't taken, so it'll be n-1 times the average of the score before, the total of the scores before, and this is after Alice takes the exam. The difference is Alice's grade exactly. Okay so the idea here is that these two queries both are allowed, because they return the average. The

database system allows that hacker to run each one of these queries by running these queries at particular times. You have to think through when exactly you want to do that. We are able to make an inference and find the exact value of the target score, just by the difference that I just mentioned. So, this is the example of an inference attack where you use authorized queries to gain access to information that you're not authorized for. So, which is the exact score of the target, in this case the target was Alice.



So let's look at another example. Maybe you say, well, the one I just gave is not that realistic. What if everybody takes the exam at the same time? Well, that particular attack will not succeed in that case.

So we're going to craft a different kind of an attack, using the same authorized query which returns the average score on an exam. So let's say in this case, we have only one student with a standing that is junior. Because the course, let's say, is a senior course. And one smart junior student chose to take it or enroll in it. So it's a senior class and we only have one student who is a junior. It's useful information that we know where there's groups of students. Okay, you may ask for the average scorer, male students, female students, students who are juniors, students who are seniors, and things like that. Because in general, if there are a lot of junior students, maybe you want to know how did you perform relative to your peers who are also juniors.

So let's take the query that says, which also specifies the standing of a student, but it's still asking for average score. Let's say that's allowed. So here we get average score of students when the standing was junior. So that's where our query says, where standing equals junior. In that case, we're still running the average score query, so it will take the tuples where the standing is junior, and then compute the average on those and return that value.

Well, that's a problem if it does do that. Because in this case, the query's actually going to disclose the average. If you're doing average for one value, then you are actually, this average discloses the value itself. So this query, if there's only one student, of course the result it's going to return is going to be the score of the student who had junior standing.

So this is another example where I think the reason this happens is that we're computing the average which is an aggregate, which is authorized as we said before, over a small set of tuples for the exam score values in tuples that are selected. In extreme case, there's only one of these tuples. And in that case, it's the exact score. So this query again allows someone to make an inference, saying the student who's standing a junior, this, the average that we get is his or her score.

Defenses Against Inference Attacks

- Do not allow aggregate query results when the set of tuples selected is either too small or too large
- Transform data by removing identifying information
  - Deindentification
  - Anonymization
  - This has to be done with care

We saw examples of inference attacks. The two quick examples when the query was an average of an exam grade that was authorized but the result was disclosing somebody's grade, which we didn't want that to happen.

So obviously, we want to have defenses against these kind of inference attacks. These defenses are actually pretty hard. In general, if you can submit an arbitrary number of such queries you can always make an inference based on the results those queries return. But there are some things that we can do to reduce the likelihood of somebody finding sensitive information by using such queries with at least limited amount of effort. What kind of defenses can we have in place?

So remember the way we do these aggregate queries that are allowed is that the query sort of says, select a set of tuples. So, for example, that case we had where we looking for Junior's standing, we're going to pull out all the tuples where the standing attribute value is Junior. So we're going to pull out all the tuples that match this query in some sense. And then we're going to aggregate the exam score. So one way to make sure that inference is more difficult is that we don't do that when the number of tuples that are selected is too few. The idea here is when it's too few, extreme case when it's one, then we know that the average actually discloses the score that we have. But if it's a small number of tuples, the average is going to be close to the score of every person who is selected or whose tuples are selected. So average for a small set essentially tells us what those values are and we don't want to do this average score when the number is too small.

Well you can think of other cases where if you asking for some property, and that selected set of tuples is very large, then that sort of holds for everyone. And if it holds for everyone, then it holds for a given user as well. So both too small and too large actually lead to this inference problem. I'll let you think about the too large part. So you should see the aggregation is happening over a set of tuples. And does it include every tuple, or almost every tuple, in the database? Or, does it include very few? And depending on the nature of the query that we have, you may say that, we can't run that query because of the number of these tuples that it's going to aggregate all. So that's one way to do that.

The other defense that we typically do is saying well, we can transform the database and can remove all the identifying information. Think about the exam score that I was talking about. Let's say we dropped the student ID and standing, even if you're concerned about too few students with a certain standing, and then we just post in the exams score without the names or the IDs.

- Transform data by removing identifying information
  - Deindentification
  - Anonymization
  - This has to be done with care

[*** missing transcript ***]

<L1P7 #31 - Defenses Against Inference Attacks.srt>

SQL Inference Attack Quiz

Choose the best answer.

The database that stores student exam scores allows queries that return average score for students coming from various states. Can this lead to an inference attack in this system?

- ☐ Yes, depending on how many students come from each state
- ☐ No, it is not possible

Look at the two options and choose the one that you think is right. The question really is saying, is inference attack possible? So either yes or no.

- ☑ Yes, depending on how many students come from each state
- ☐ No, it is not possible

So think about, we're doing queries where we're computing the average and returning it and we're doing that only for students that come from a certain state. So, what if there's only one student from, say, a smaller state. Wyoming for example. So if there is only one student from Wyoming, this is kind of like the Junior standing discussion we had. There was only one student, the Junior standing. So there's only one student, we ask the query, saying give me the average of students who are from Wyoming. The average is going to be the same exact score of that student, one student, who comes from Wyoming. So, depending on how many students, if there are too few, extreme case only one, then, of course, the average is the score. So, Yes is the answer. It will not be possible if there are lots students from each state. But we didn't say that here so I would pick the first answer.

SQL Inference Attack Quiz #2

Choose the best answer.

Assume in (1), the data in the database is de-identified by removing student id (and other information such as names). Furthermore, the field that has the state of the student is generalized by replacing it with the US region (e.g., Midwest). The generalization ensures that there are at least two students from each region. Are inference attacks still possible?

- ☐ Yes
- ☐ No

We have the same database that we were talking about, but we're going to do a different kind of attack here. We're going to de-identify it by removing student ID, so de-identification, you take the database, drop the attribute that is student ID, and, if there are names, maybe we remove those, too. Furthermore, the field that has the state of the student because we just now saw in the previous question that, if there are too few students from a given state that could be a problem. So we're going to do generalization by replacing state with sort of the US region. East Coast, Midwest, West Coast, whatever way you want to do it. We're not asking for a specific state. That is too small and has only one or a small number of students. So this generalization is essentially replacing state by a larger geography, hoping that there'll be more students who would come from the geography. The generalization ensures that there are at least two students from each region. The problem where we only had one from Wyoming is going away. Are inference attacks still possible? Again, yes-no possibility, so think about it.

So we've done de-identification, we've done generalization, and then we're logging these queries that return the average value. Can let's say Alice and Bob both come from the Midwest? Is it possible for Alice to figure out Bob's grade? I'll seek and ask for the average score of students who come from the Midwest. So she knows the average of her and Bob's scores. She knows her own score. Based on that she's going to, it's easy to see how she will be able to compute Bob's score. So the answer is yes here. It is possible when there are only two students. Remember? We said a generalization ensures that there are at least two students. So let's say from Midwest that we're talking about, the two students, Alice and Bob. Okay so, Alice can get the average for herself and Bob. There were only two coming from this region. And she knows her own grade based on she can make an inference about what Bob's grade is, can you exactly compute Bob's grade. So yes, an inference attack. It's possible even when we do this with de-identification and generalization. If you do generalization course grant generalization with there more and more people, this likelihood would reduce. But clearly the example that we have here, this is certainly possible.

Database Security
Lesson Summary

- Used to **store lots of sensitive data** that can be accessed via programs (queries)
- Access control must be **based on operations allowed by databases**
- New attacks on databases arise due to their unique characteristics
- Defenses **must address such attacks**

We saw that how structure of the data that is stored in an inter-relational database changes the way we do access control. We also saw that certain kind of queries, in particular queries that return arrogate results, can leak potentially sensitive data via inference attacks. So we studied this new kind of attacks. Inference and injection attacks. And ways in which we can address those attacks for database systems.