Software Security and Dependability

ENGR5560G

Lecture 07

Database Security

Dr. Khalid A. Hafeez

Spring, 2025

Objectives

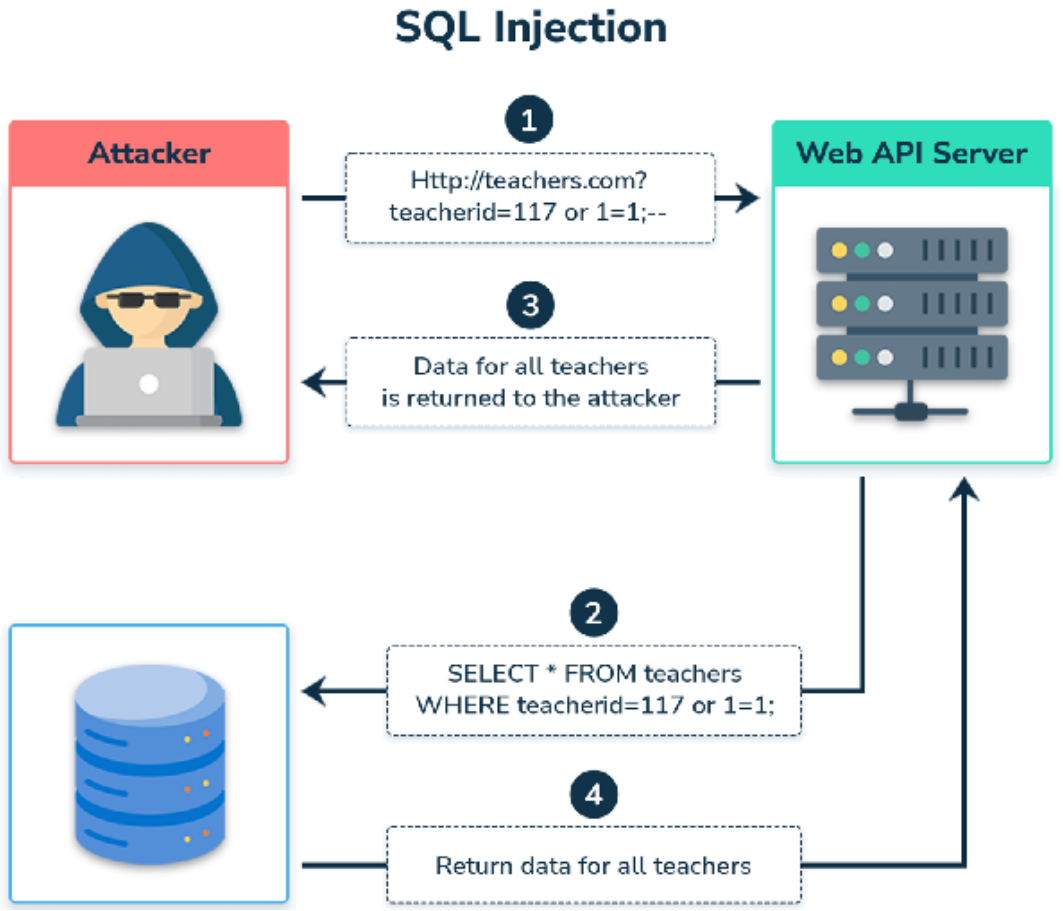
- Understand the unique need for database security, separate from ordinary computer security measures.
- Present an overview of the basic elements of a database management system.
- Present an overview of the basic elements of a relational database system.
- Compare and contrast different approaches to database access control.
- Explain how inference poses a security threat in database systems.
- Understand the nature of statistical databases and their related security issues.
- Discuss the use of encryption in a database system.
- Understand the unique security issues related to data centers.

Why database security?

- Organizational databases tend to concentrate sensitive information in a single logical system.
- **Reasons database security has not kept pace with the increased reliance on databases are:**
  - There is a dramatic imbalance between the complexity of modern database management systems (DBMS) and the security technique used to protect these critical systems
  - Databases have a sophisticated interaction protocol, Structured Query Language (SQL), which is complex
  - Effective database security requires a strategy based on a full understanding of the security vulnerabilities of SQL
  - The typical organization lacks full-time database security personnel
  - Most enterprise environments consist of a heterogeneous mixture of database platforms, enterprise platforms, and OS platforms, creating an additional complexity hurdle for security personnel
  - The increasing reliance on cloud technology to host part or all the corporate database

SQL Injection Attacks (SQLi)

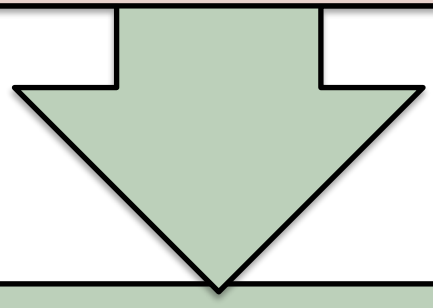
- SQLi is one of the most prevalent and dangerous network-based security threats
- Sends malicious SQL commands to the database server
- **Most common attack goals:**
  - Bulk extraction of data
  - Modify or delete data
  - Execute arbitrary operating system commands
  - Launch denial-of-service (DoS) attacks
- The attack is viable when user input is either:
  - incorrectly filtered for string literal escape characters embedded in SQL statements
  - user input is not strongly typed, and thereby unexpectedly executed.



SQLi Injection Technique

The SQLi attack typically works by prematurely terminating a text string and appending a new command

Because the inserted command may have additional strings appended to it before it is executed the attacker terminates the injected string with a comment mark "- -"



Subsequent text is ignored at execution time

SQLi - example

- **Consider a script that build an SQL query by combining predefined strings with text entered by a user:**

```
Var Shipcity;
Shipcity= Request.from (“ShipCity”);
Var sql= "select * from OrdersTable Where ShipCity= '" + ShipCity +"'";
```

- The script’s designer expect that the user enters a city: such as ‘Toronto’, then, the following SQL query will be generated:

```
SELECT * FROM OrdersTable WHERE SHIPCITY=‘Toronto'
```

- Suppose the user enters the following: Toronto'; DROP table OrdersTable- -
- This results in the following query:

```
SELECT * FROM OrdersTable WHERE SHIPCITY=‘Toronto'; DROP table OrdersTable- -'
```

  - The semicolon separates two commands
  - The double dash indicates that the remaining text of the current line is a comment and not to be executed
  - When the server executes this statement; it first selects all records in OrdersTable where shipCity=‘Toronto’. Then it executes the DROP request which deletes the table.

SQLi Attack Avenues

User input

- Attackers inject SQL commands by providing suitable crafted user input

Server variables

- Attackers can forge the values that are placed in HTTP and network headers and exploit this vulnerability by placing data directly into the headers

Second-order injection

- A malicious user could rely on data already present in the system or database to trigger an SQL injection attack, so when the attack occurs, the input that modifies the query to cause an attack does not come from the user, but from within the system itself

Cookies

- An attacker could alter cookies such that when the application server builds an SQL query based on the cookie's content, the structure and function of the query is modified

Physical user input

- Applying user input that constructs an attack outside the realm of web requests (user-input could take the form of conventional barcodes, RFID tags)

SQLi Attack types

- Attack types can be grouped into three main categories:
  - Inband,
  - Inferential,
  - Out-of-band.

SQLi Attack types - Inband Attacks:

- Uses the same communication channel for injecting SQL code and retrieving results and presenting them in the application Web page
- **Inband attack types:**
  - **Tautology**: injects code in one or more conditional statements so that they always evaluate to true.

    Example:

    $query = "SELECT info FROM user WHERE name ='$_GET["name"]' AND pwd = '$_GET["pwd"]'";

    If the attacker submits " ' OR 1=1 --" for the name field, the resulting query will be:

    SELECT info FROM users WHERE name = ' ' OR 1=1 -- AND pwpd = ' '

    Because the conditional is a tautology, the query evaluates to true for each row in the table and returns all of them.

  - **End-of-line comment**: After injecting code into a particular field, legitimate code that follows are nullified through usage of end of line comments (--). Such as the previous two examples.

  - **Piggybacked queries**: The attacker adds additional queries beyond the intended query, piggy-backing the attack on top of a legitimate request. This happens on server that allow several different queries within a single string of code.

SQLi Attack types - Inferential Attacks:

- There is no actual transfer of data, but the attacker is able to reconstruct the information by sending particular requests and observing the resulting behavior of the Website/database server
- **Inferential attack types:**
  - Illegal/logically incorrect queries
    - This attack lets an attacker gather important information about the type and structure of the backend database of a Web application
    - The attack is considered a preliminary, information-gathering step for other attacks
  - Blind SQL injection
    - Allows attackers to infer the data present in a database system even when the system is sufficiently secure to not display any erroneous information back to the attacker
      - Example: The attacker asks the server true/false questions.

SQLi Attack types - Out-of-Band Attacks:

- Data are retrieved using a different channel
  - Example: an email with the results of the query is generated and sent to the tester
- This can be used when there are limitations on information retrieval, but outbound connectivity from the database server is lax

SQLi Countermeasures:

- Because SQLi attacks are so prevalent, damaging, and varied both by attack avenue and type, a single countermeasure is insufficient
- **Three types:**

Defensive coding

- Manual defensive coding practices
  - Input type checking
  - Pattern matching
- Parameterized query insertion
- SQL DOM
  - SQL DOM is a set of classes that enables automated data type validation and escaping

Detection

- Signature based
  - Match with specific attack patterns
- Anomaly based:
  - define normal behavior and then detect behavior patterns outside the normal range
- Code analysis
  - use of a test suite to detect SQLi vulnerabilities.

Run-time prevention

- Check queries at runtime to see if they conform to a model of expected queries

Database Access Control

- Database access control system determines:
  - If the user has access to the entire database or just portions of it
  - What access rights the user has (create, insert, delete, update, read, write)

- DBMS can support a range of administrative policies, including the following:
  - Centralized administration: A small number of privileged users may grant and revoke access rights.
  - Ownership-based administration: The owner (creator) of a table may grant and revoke access rights to the table.
  - Decentralized administration: the owner of the table may grant and revoke authorization rights to other users, allowing them to grant and revoke access rights to the table.

SQL Access Control

- SQL provides two commands for managing access rights: **GRANT & REVOKE**
  - **GRANT**: Used to grant one or more access rights or to assign a user to a role
    - Format:

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

    - Example: GRANT SELECT ON ANY TABLE TO ricflair

    - TO clause: specifies the user or role to which the rights are granted.
    - A PUBLIC value indicates that any user has the specified access rights.
    - The optional IDENTIFIED BY clause specifies a password that must be used to revoke the access rights of this GRANT command.
    - The GRANT OPTION indicates that the grantee can grant this access right to other users, with or without the grant option.

SQL Access Control

- **The following is a typical list of access rights:**
  - **Select**: Grantee may read entire database; individual tables; or specific columns in a table.
  - **Insert**: Grantee may insert rows in a table; or insert rows with values for specific columns in a table.
  - **Update**: Semantics is similar to INSERT.
  - **Delete**: Grantee may delete rows from a table.
  - **References**: Grantee is allowed to define foreign keys in another table that refer to the specified columns.

SQL Access Control

- **REVOKE**: Revokes the access rights

- Syntax:

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

  - Example: REVOKE SELECT ON ANY TABLE FROM ricflair

SQL Access Control

- Cascading Authorizations:
  - The grant option enables an access right to cascade through a number of users.
  - When user **A** revokes an access right, any cascaded access right is also revoked, unless that access right would exist even if the original grant from **A** had never occurred.

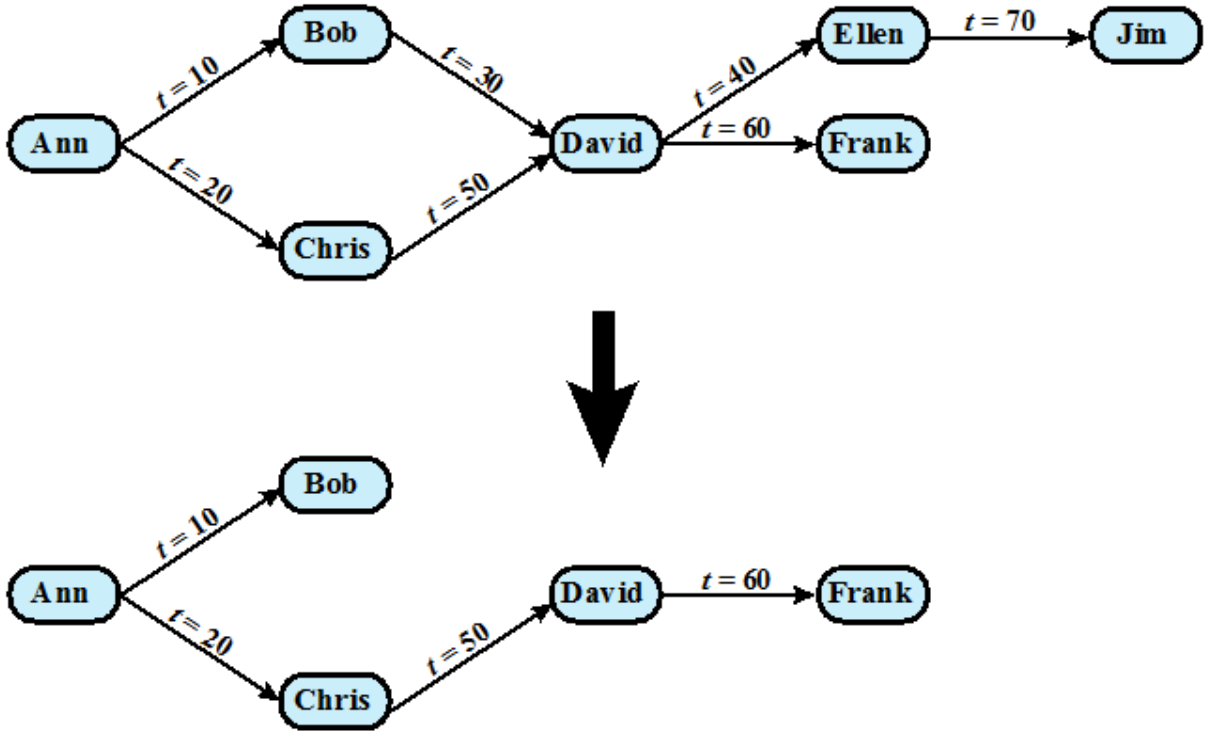


Figure 5.6 Bob Revokes Privilege from David

Role-Based Access Control (RBAC)

- RBAC eases administrative burden and improves security (natural fit for databases)
- A database RBAC needs to provide the following capabilities:
  - Create and delete roles
  - Define permissions for a role
  - Assign and cancel assignment of users to roles
- Categories of database users in discretionary access control environment:

Application owner	End user	Administrator
• An end user who owns database objects (tables, columns, rows) as part of an application	• An end user who operates on database objects via a particular application but does not own any of the database objects	• User who has administrative responsibility for part or all of the database

Inferential Attack

- **Inference** is the process of performing authorized queries and deducing unauthorized information from the legitimate responses received.
  - Two inference techniques can be used to derive additional information:
  1. Analyzing functional dependencies between attributes within a table or across tables
  2. Merging views with the same constraints.

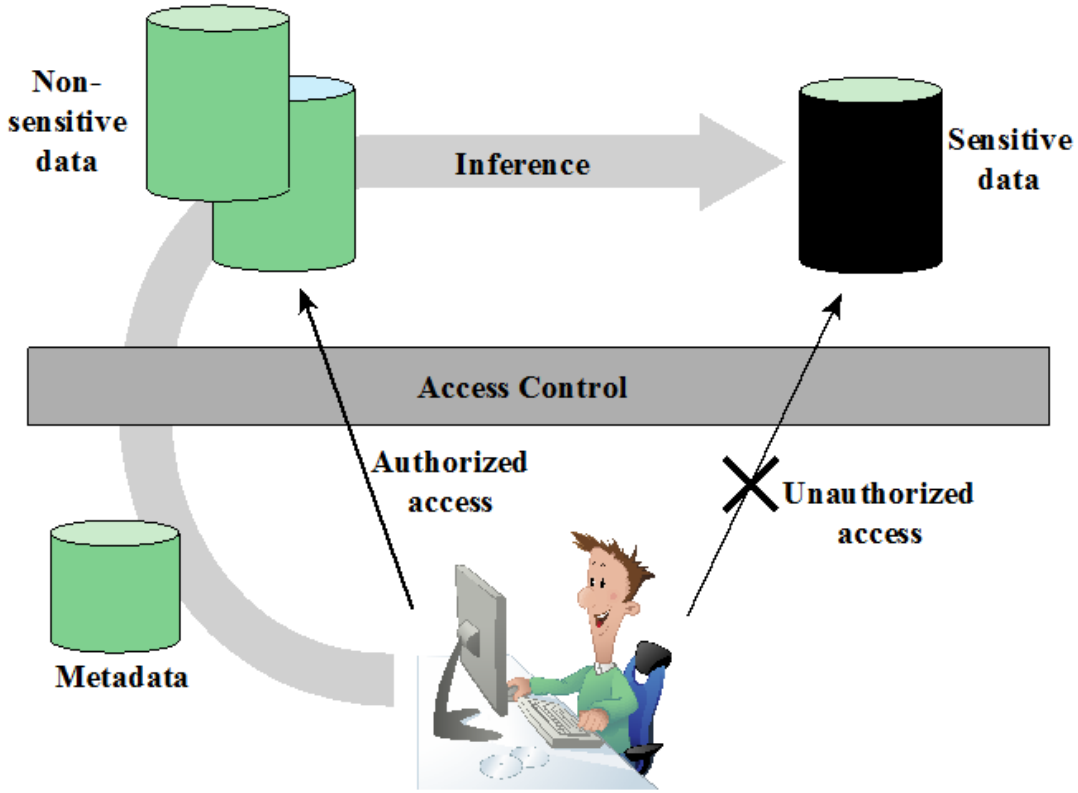


Figure 5.7 Indirect Information Access Via Inference Channel

Inferential Attack - example

- Suppose that users of these views are not authorized to access the relationship between name and Salary.
- A user who knows the structure of the Employee table and who knows that the view tables maintain the same row order as the Employee table is then able to merge the two views to construct the table shown in Figure c.
- This violates the access control policy that the relationship of attributes Name and Salary must not be disclosed.

Name	Position	Salary ($)	Department	Dept. Manager
Andy	senior	43,000	strip	Cathy
Calvin	junior	35,000	strip	Cathy
Cathy	senior	48,000	strip	Cathy
Dennis	junior	38,000	panel	Herman
Herman	senior	55,000	panel	Herman
Ziggy	senior	67,000	panel	Herman

(a) Employee table

Position	Salary ($)
senior	43, 000
junior	35,000
senior	48,000

Name	Department
Andy	strip
Calvin	strip
Cathy	strip

(b) Two views

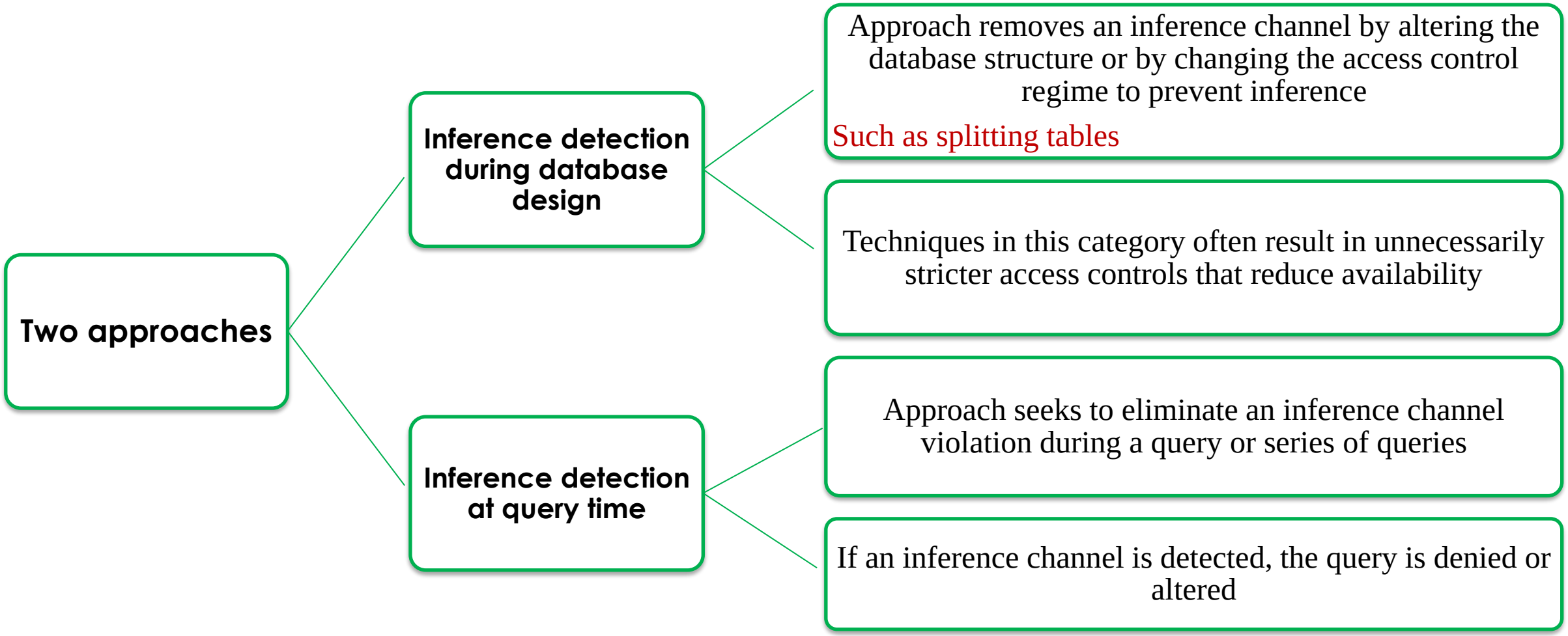
Name	Position	Salary ($)	Department
Andy	senior	43,000	strip
Calvin	junior	35,000	strip
Cathy	senior	48,000	strip

(c) Table derived from combining query answers

```
CREATE view V1 AS SELECT Position, Salary FROM Employee WHERE Department = ”strip”
CREATE view V2 AS SELECT Name, Department FROM Employee WHERE Department = ”strip”
```

Inferential Attack Detection

- Two approaches to dealing with the threat of disclosure by inference:



- Some inference detection algorithm is needed for either of these approaches (ongoing research)

Inferential Attack Detection

- Example:

  Employees (Emp#, Name, Address) ← Clerk role
  Salaries (S#, Salary) ←
  Emp-Salary (Emp#, S#) ← Admin role

  - If later, we need to add the start-date of the employee

    Employees (Emp#, Name, Address)
    Salaries (S#, Salary, Start-Date)
    Emp-Salary (Emp#, S#)

    ← Inference problem

  - Solution:

    Employees (Emp#, Name, Address, Start-Date)
    Salaries (S#, Salary)
    Emp-Salary (Emp#, S#)

    ← solution

Database Encryption

- The database is typically the most valuable information resource for any organization
  - Therefore, it is protected by multiple layers of security, including:
    - Firewalls, authentication, general access control systems, DB access control systems, database encryption
    - Encryption becomes the last line of defense in database security
  - Can be applied to the entire database, at the record level, the attribute level, or level of the individual field

- **Disadvantages to encryption:**
  - Key management
    - Authorized users must have access to the decryption key for the data for which they have access (key management is a complex task)
  - Inflexibility
    - When part or all the database is encrypted, it becomes more difficult to perform record searching

Database Encryption

- An attractive solution is to outsource the DBMS and the database to a service provider.
  - A straightforward solution to the security problem in this context is to encrypt the entire database and not provide the encryption/decryption keys to the service provider.
    - This solution by itself is inflexible.
- To provide more flexibility, it must be possible to work with the database in its encrypted form.
  - **Four entities are involved:**
    - Data owner: organization that produces data to be made available for controlled release
    - User: human entity that presents queries to the system
    - Client: front end that transforms user queries into queries on the encrypted data stored on the server
    - Server: an organization that receives the encrypted data from a data owner and makes them available for distribution to clients

Database Encryption

- Database Encryption Scheme: example
  - The user issues an SQL query for fields from one or more records with a specific value of a primary key
  - The query processor at the client encrypts the primary key, modifies the query accordingly, and send it to the server
  - The server processes the query with the encrypted key and returns the encrypted results.
  - The query processor decrypts the data and returns the result