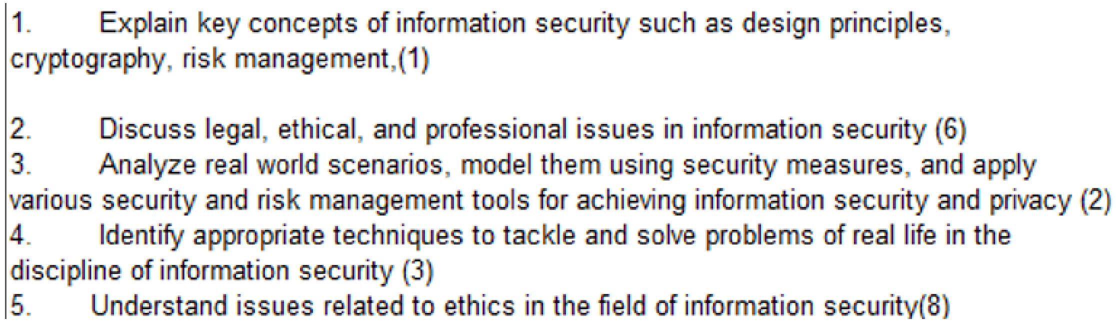
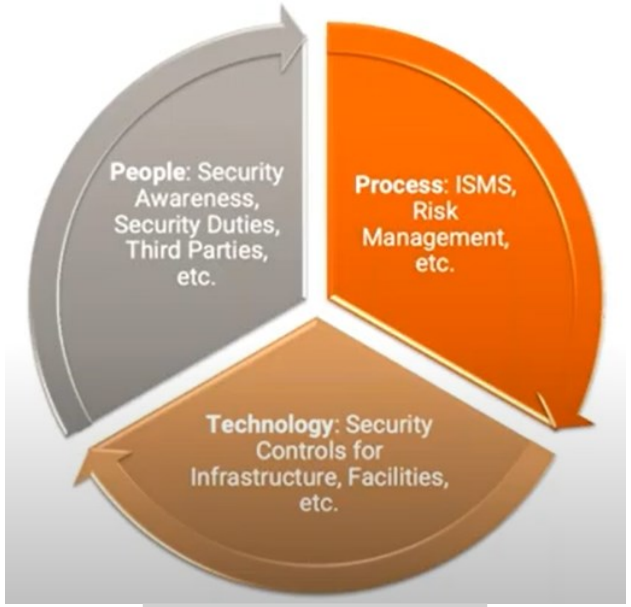
CS 3002 Information Security

Fall 2022



1. Explain key concepts of information security such as design principles, cryptography, risk management,(1)

2. Discuss legal, ethical, and professional issues in information security (6)
3. Analyze real world scenarios, model them using security measures, and apply various security and risk management tools for achieving information security and privacy (2)
4. Identify appropriate techniques to tackle and solve problems of real life in the discipline of information security (3)
5. Understand issues related to ethics in the field of information security(8)


People: Security Awareness, Security Duties, Third Parties, etc.
Process: ISMS, Risk Management, etc.
Technology: Security Controls for Infrastructure, Facilities, etc.

ISO/IEC 27001: 2013

Week # 8 – Lecture # 19, 20, 21

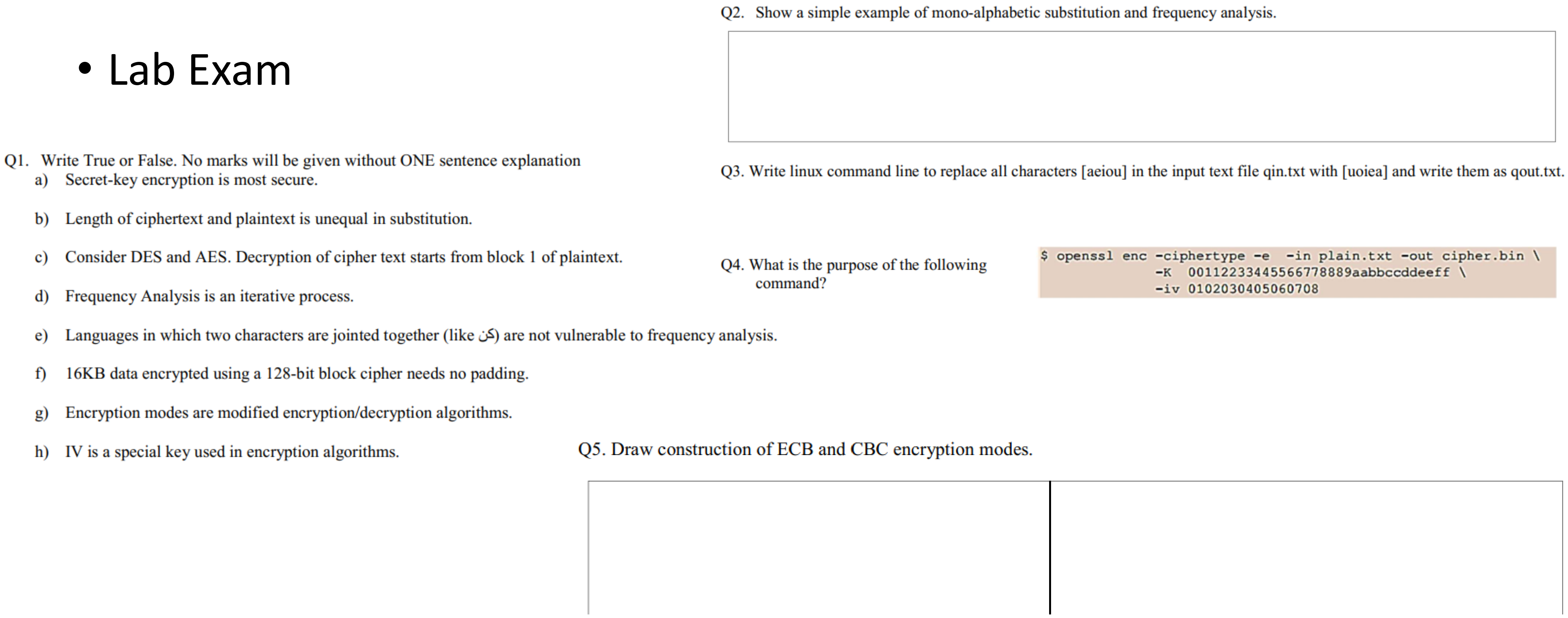
14th, 15th, 16th Rabi ul Awwal, 1444

11th, 12th, 13th October 2022

Dr. Nadeem Kafi Khan

Lecture # 19 - LAB

- Lab Exam



Q1. Write True or False. No marks will be given without ONE sentence explanation

a) Secret-key encryption is most secure.

b) Length of ciphertext and plaintext is unequal in substitution.

c) Consider DES and AES. Decryption of cipher text starts from block 1 of plaintext.

d) Frequency Analysis is an iterative process.

e) Languages in which two characters are jointed together (like کن) are not vulnerable to frequency analysis.

f) 16KB data encrypted using a 128-bit block cipher needs no padding.

g) Encryption modes are modified encryption/decryption algorithms.

h) IV is a special key used in encryption algorithms.

Q2. Show a simple example of mono-alphabetic substitution and frequency analysis.

Q3. Write linux command line to replace all characters [aeiou] in the input text file qin.txt with [uoiea] and write them as qout.txt.

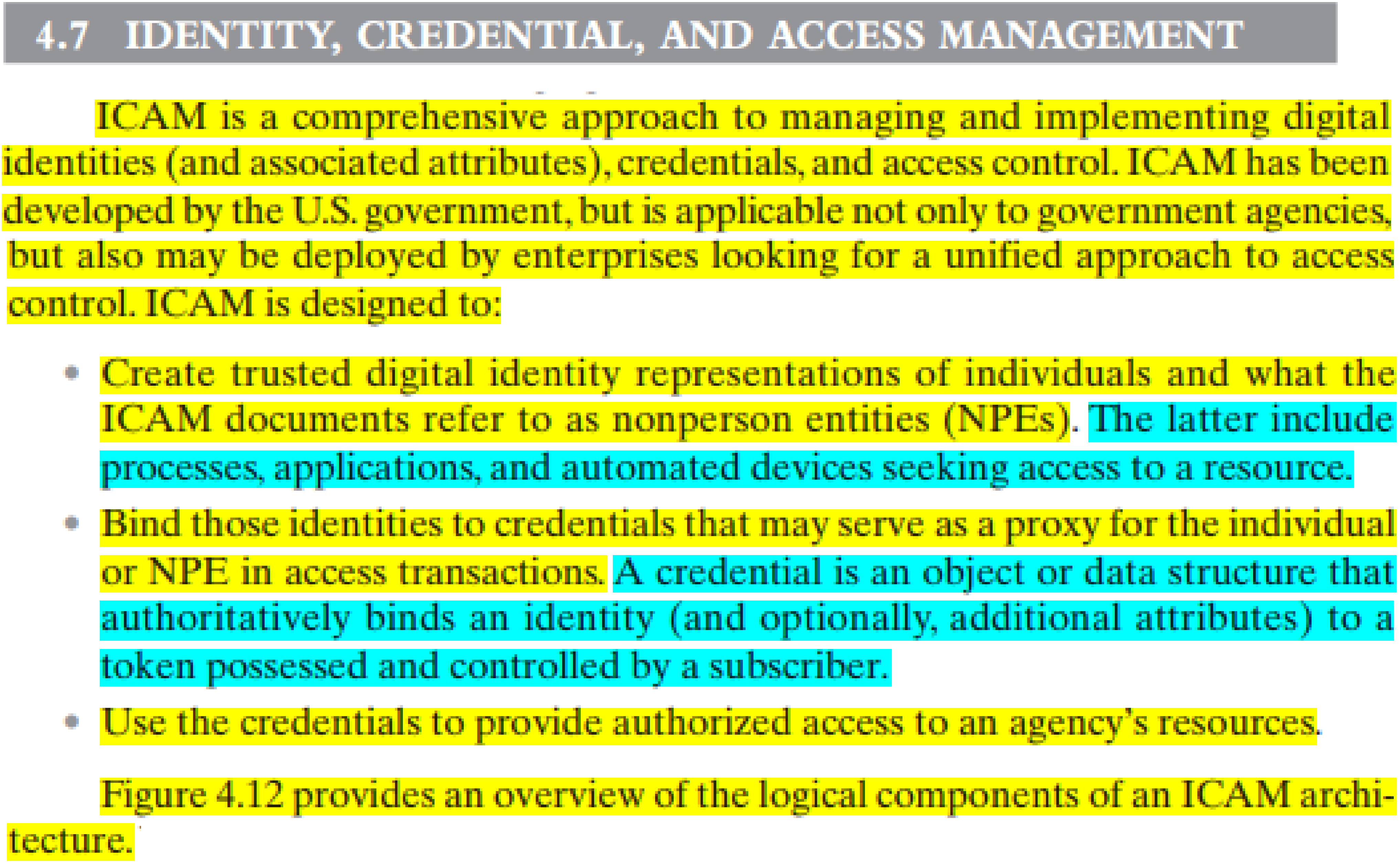
Q4. What is the purpose of the following command?

```
$ openssl enc -ciphertype -e  -in plain.txt -out cipher.bin \
              -K  00112233445566778889aabbccddeeff \
              -iv 0102030405060708
```

Q5. Draw construction of ECB and CBC encryption modes.

Lecture # 20

- Identify, Credential and Access Management (ICAM)
  - Figure 4.2 discussed thoroughly and section was given as reading assignment
- Database Security
- RDBMS
- SQL (as code) execution
- Web Application scenario discussed for SQL Injection



4.7 IDENTITY, CREDENTIAL, AND ACCESS MANAGEMENT

ICAM is a comprehensive approach to managing and implementing digital identities (and associated attributes), credentials, and access control. ICAM has been developed by the U.S. government, but is applicable not only to government agencies, but also may be deployed by enterprises looking for a unified approach to access control. ICAM is designed to:

- Create trusted digital identity representations of individuals and what the ICAM documents refer to as nonperson entities (NPEs). The latter include processes, applications, and automated devices seeking access to a resource.
- Bind those identities to credentials that may serve as a proxy for the individual or NPE in access transactions. A credential is an object or data structure that authoritatively binds an identity (and optionally, additional attributes) to a token possessed and controlled by a subscriber.
- Use the credentials to provide authorized access to an agency's resources.

Figure 4.12 provides an overview of the logical components of an ICAM architecture.

Self Reading as per the explanation during the lecture

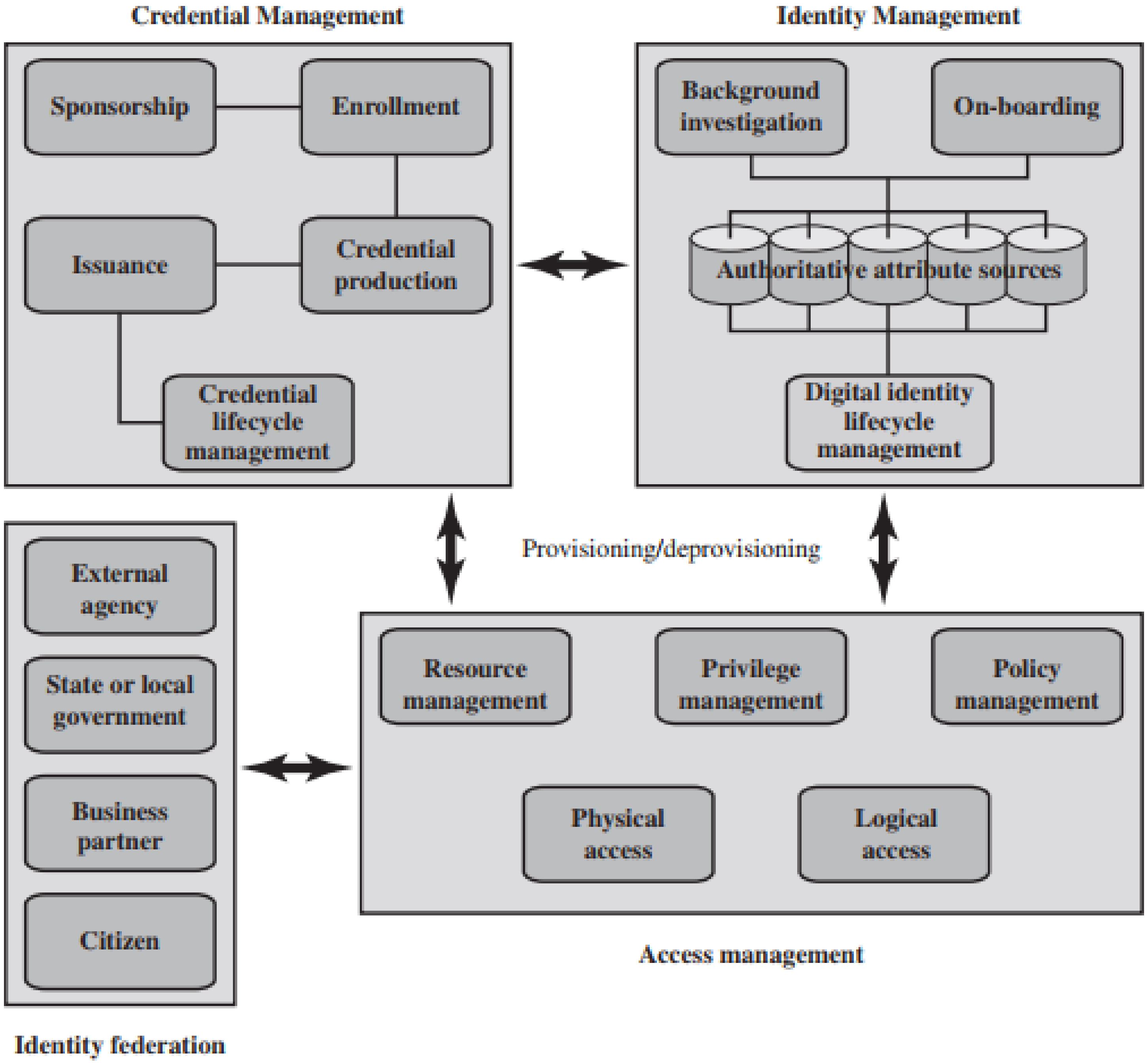
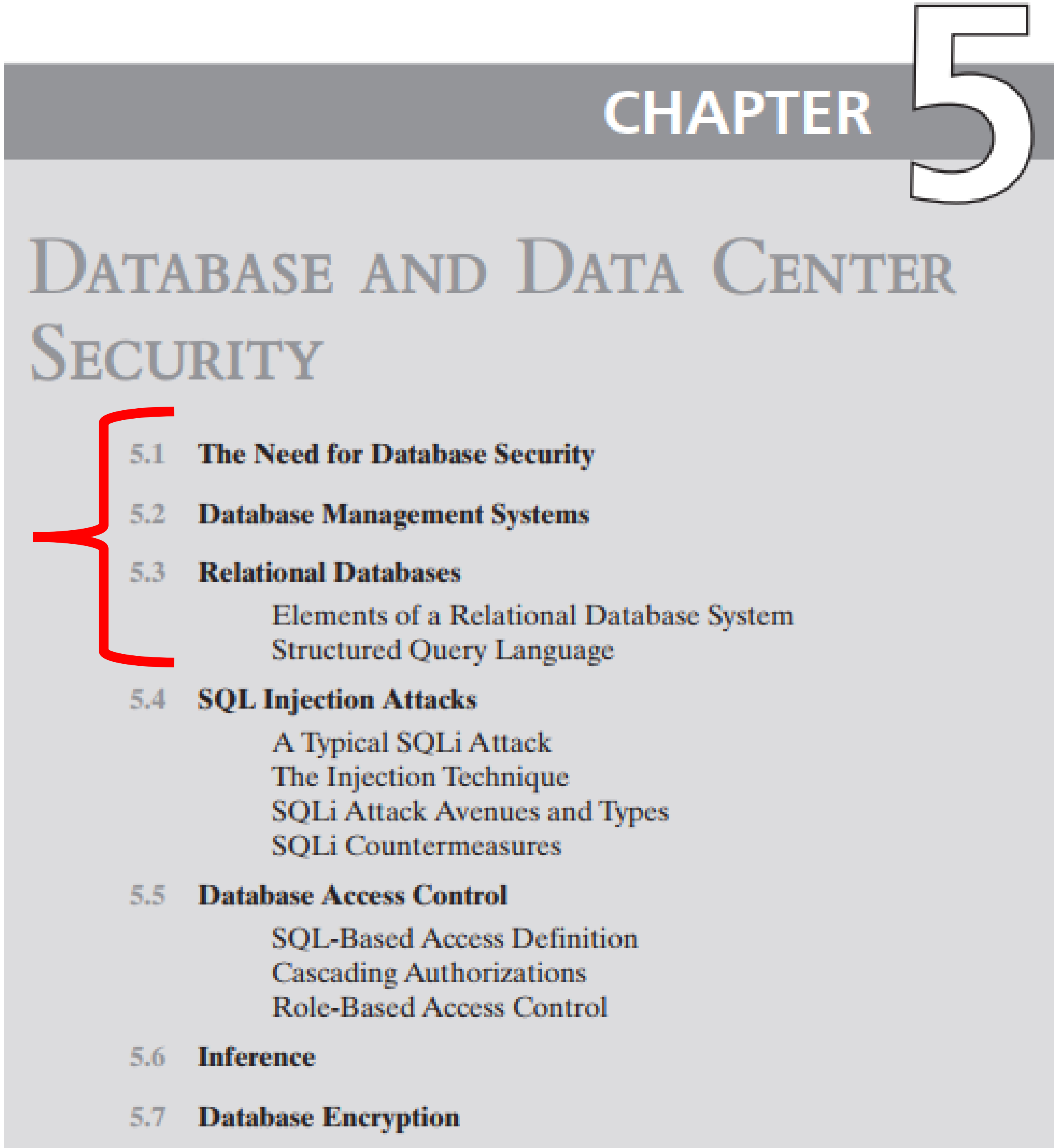


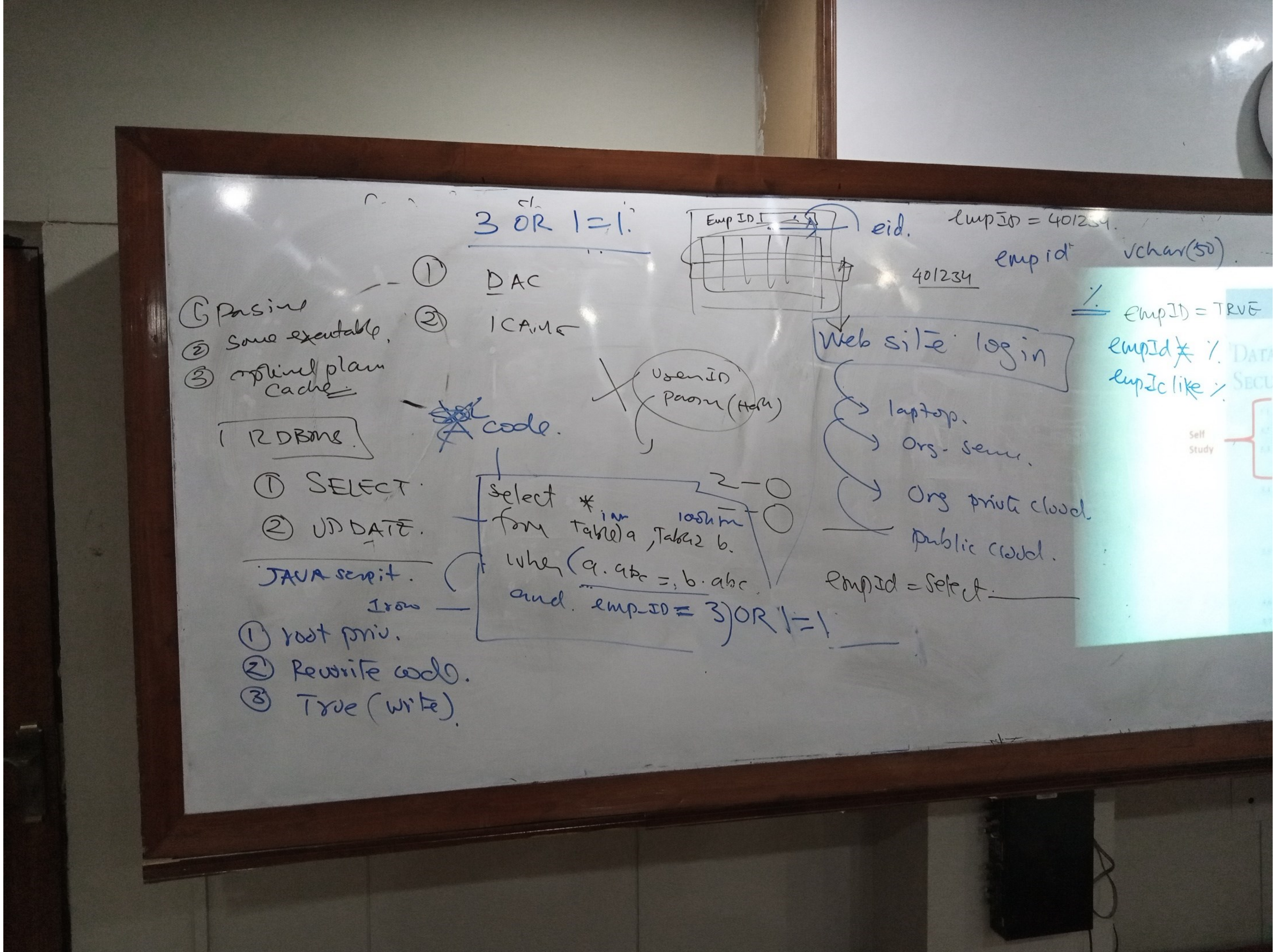
Figure 4.12 **Identity, Credential, and Access Management (ICAM)**

Self Study



CHAPTER 5

Database and Data Center Security



3 OR 1=1.

① DAC

② ICAME

① Pasing
② Some executable.
③ optimised plan cache

RDBMS

① SELECT.
② UPDATE.

JAVA script.

~~Sql~~ code.

UserID
Passw (Hash)

```
select *
from Table1 a, Table2 b.
when (a.abc =, b.abc
and. emp_ID = 3)OR 1=1
```

1 row

1 m

100h m

2 — ○

① root priv.
② Rewrite code.
③ True (write).

Emp ID

eid.

empID = 401234.

empid vchar(50).

401234

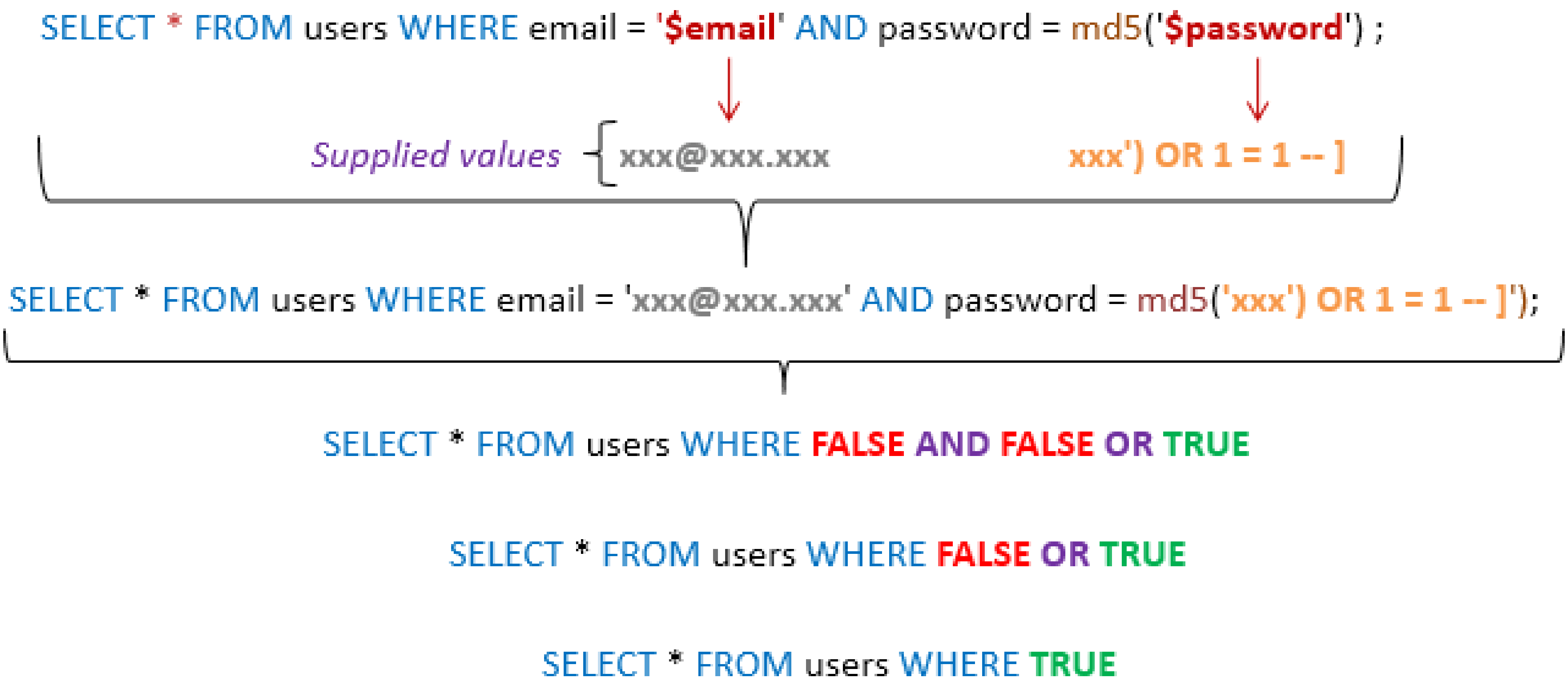
Web site login

laptop.
Org - serv.
Org privt cloud.
public cloud.

empId = Select

%

empID = TRUE
empId * %.
empIc like %.



SELECT * FROM users WHERE email = '$email' AND password = md5('$password') ;

Supplied values xxx@xxx.xxx xxx') OR 1 = 1 --]

SELECT * FROM users WHERE email = 'xxx@xxx.xxx' AND password = md5('xxx') OR 1 = 1 --]');

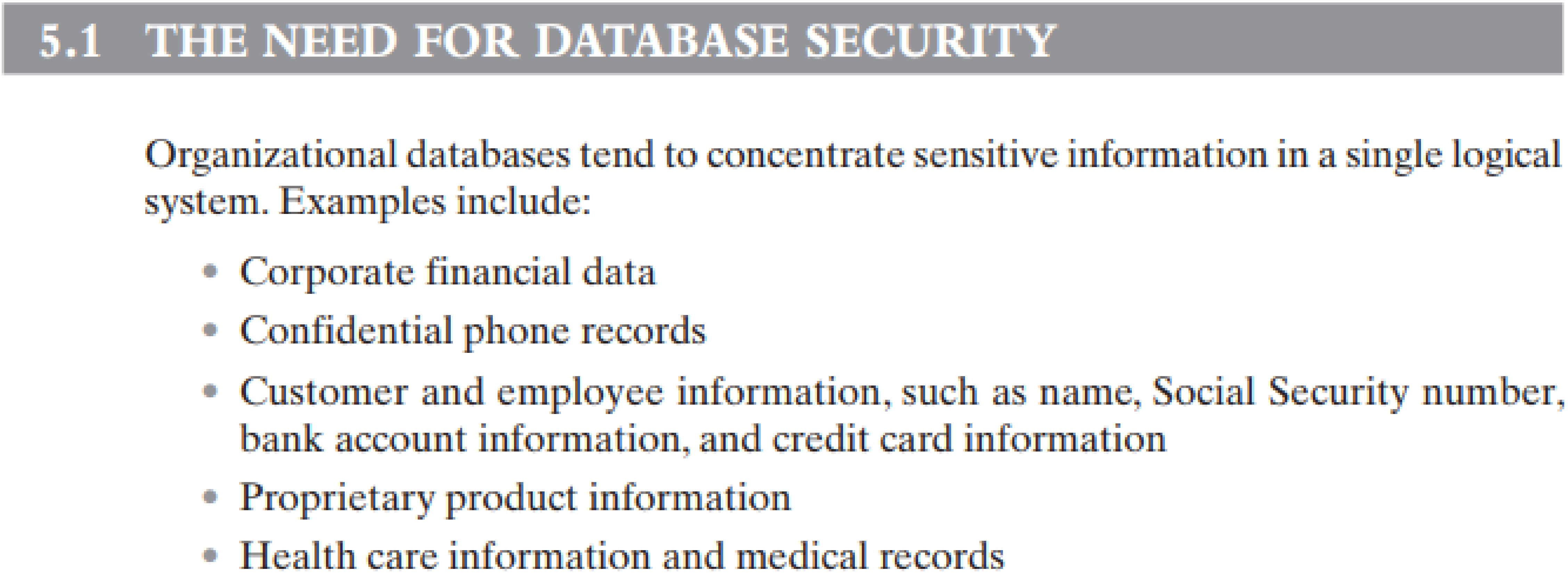
SELECT * FROM users WHERE FALSE AND FALSE OR TRUE

SELECT * FROM users WHERE FALSE OR TRUE

SELECT * FROM users WHERE TRUE

Lecture # 21

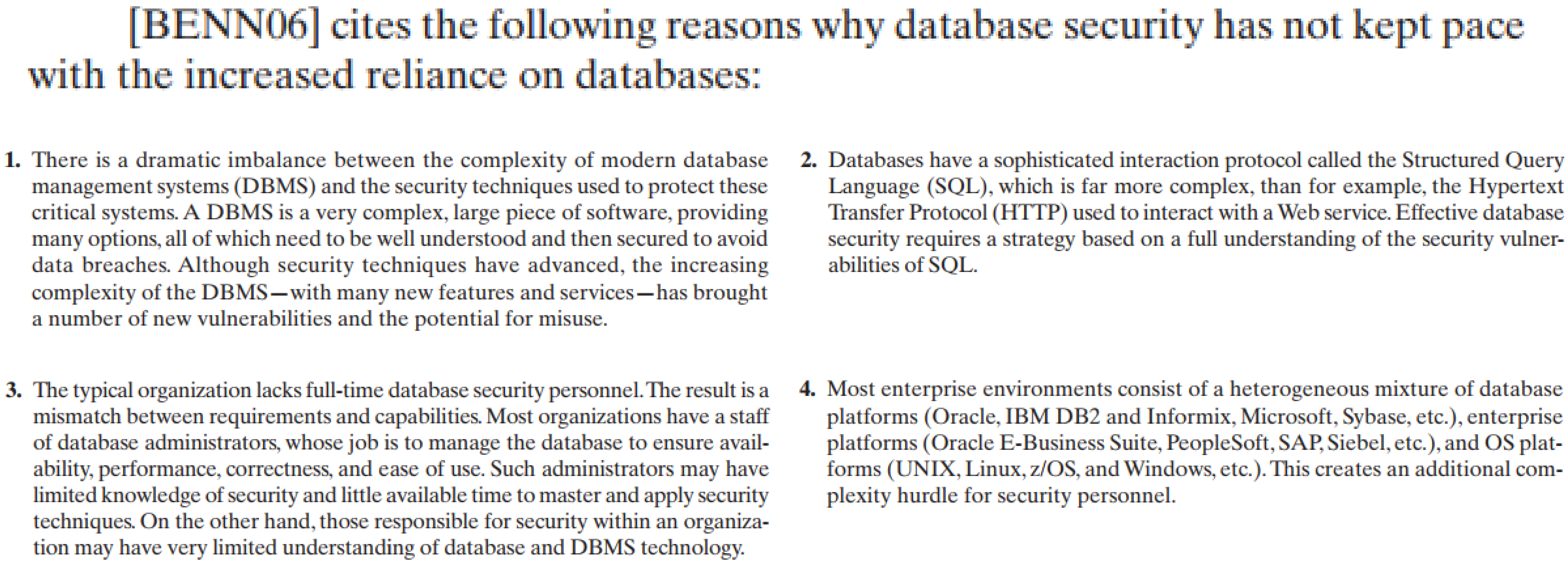
- Database security
- SQL Injection Attack
- Steps involved in a typical SQL Injection attack
- The Injection techniques
- SQL Injection attack types
    - In-band, out-of-band, inferential



5.1 THE NEED FOR DATABASE SECURITY

Organizational databases tend to concentrate sensitive information in a single logical system. Examples include:

- Corporate financial data
- Confidential phone records
- Customer and employee information, such as name, Social Security number, bank account information, and credit card information
- Proprietary product information
- Health care information and medical records



[BENN06] cites the following reasons why database security has not kept pace with the increased reliance on databases:

1. There is a dramatic imbalance between the complexity of modern database management systems (DBMS) and the security techniques used to protect these critical systems. A DBMS is a very complex, large piece of software, providing many options, all of which need to be well understood and then secured to avoid data breaches. Although security techniques have advanced, the increasing complexity of the DBMS—with many new features and services—has brought a number of new vulnerabilities and the potential for misuse.

2. Databases have a sophisticated interaction protocol called the Structured Query Language (SQL), which is far more complex, than for example, the Hypertext Transfer Protocol (HTTP) used to interact with a Web service. Effective database security requires a strategy based on a full understanding of the security vulnerabilities of SQL.

3. The typical organization lacks full-time database security personnel. The result is a mismatch between requirements and capabilities. Most organizations have a staff of database administrators, whose job is to manage the database to ensure availability, performance, correctness, and ease of use. Such administrators may have limited knowledge of security and little available time to master and apply security techniques. On the other hand, those responsible for security within an organization may have very limited understanding of database and DBMS technology.

4. Most enterprise environments consist of a heterogeneous mixture of database platforms (Oracle, IBM DB2 and Informix, Microsoft, Sybase, etc.), enterprise platforms (Oracle E-Business Suite, PeopleSoft, SAP, Siebel, etc.), and OS platforms (UNIX, Linux, z/OS, and Windows, etc.). This creates an additional complexity hurdle for security personnel.

Important SQL Syntax

COMMENTS: --

Example: SELECT * FROM `table` --selects everything

LOGIC: ‘a’=‘a’

Example: SELECT * FROM `table` WHERE ‘a’=‘a’

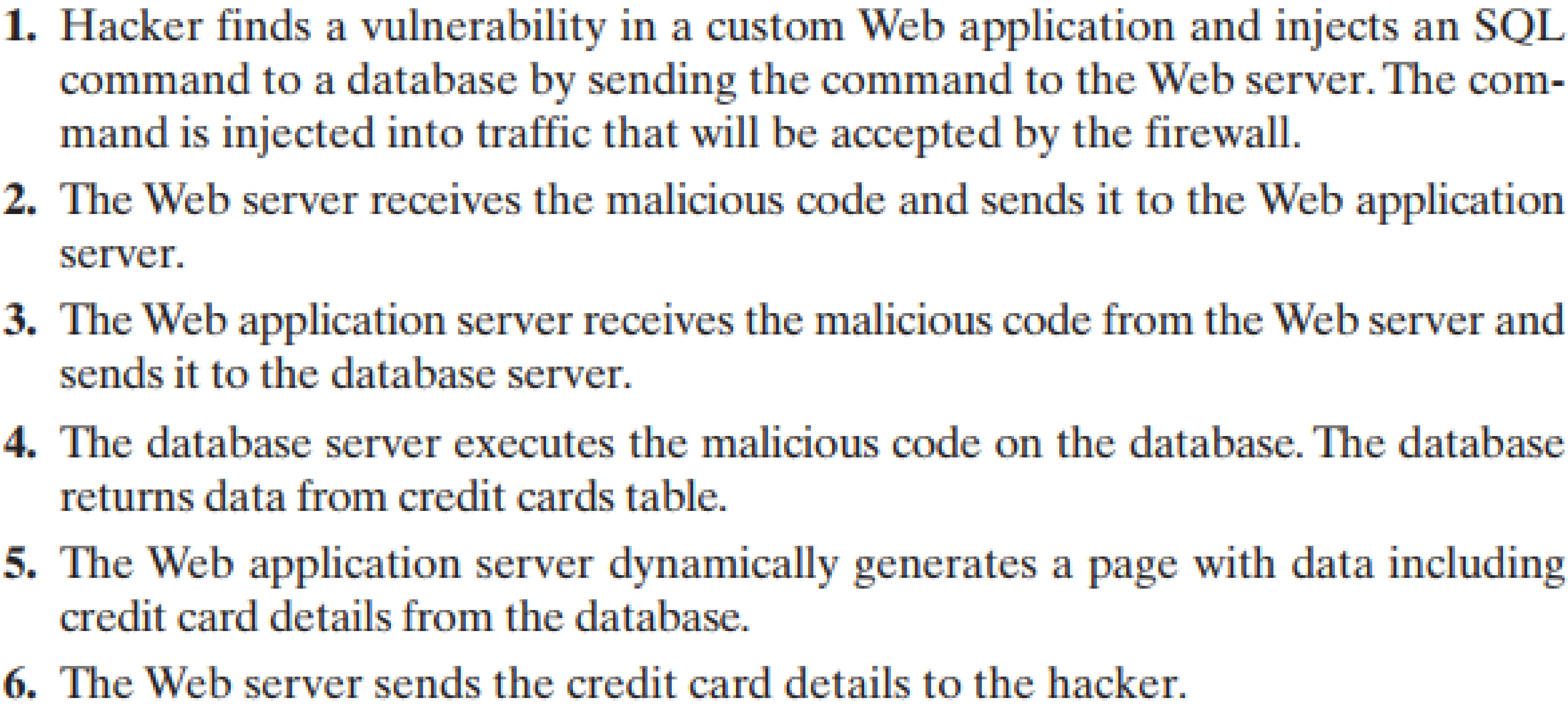
MULTI STATEMENTS: S1; S2

Example: SELECT * FROM `table`; DROP TABLE `table`;

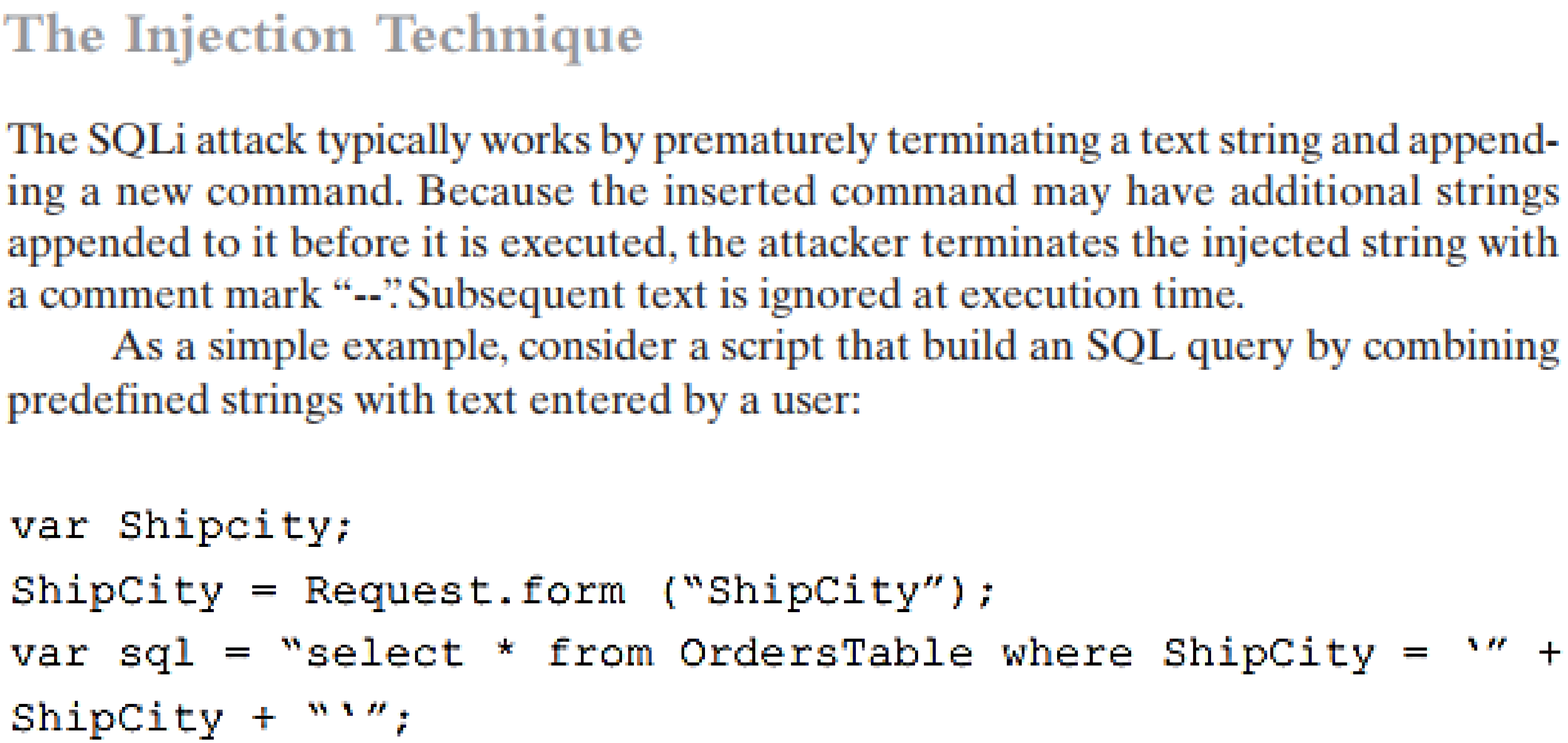
5.4 SQL INJECTION ATTACKS

- The SQL injection (SQLi) attack is one of the most prevalent and dangerous network-based security threats.
- An SQLi attack is designed to exploit the nature of Web application pages.
  - Most current websites have dynamic components and content. Many such pages ask for information, such as location, personal identity information, and credit card information. This dynamic content is usually transferred to and from back-end databases that contain volumes of information.
  - An application server webpage will make SQL queries to databases to send and receive information critical to making a positive user experience.
- In such an environment, an SQLi attack is designed to send malicious SQL commands to the database server.
  - The most common attack goal is bulk extraction of data.
  - Attackers can dump database tables with hundreds of thousands of customer records.
- Depending on the environment, SQL injection can also be exploited to modify or delete data, execute arbitrary operating system commands, or launch denial-of-service (DoS) attacks.

A Typical SQLi Attack



1. Hacker finds a vulnerability in a custom Web application and injects an SQL command to a database by sending the command to the Web server. The command is injected into traffic that will be accepted by the firewall.
2. The Web server receives the malicious code and sends it to the Web application server.
3. The Web application server receives the malicious code from the Web server and sends it to the database server.
4. The database server executes the malicious code on the database. The database returns data from credit cards table.
5. The Web application server dynamically generates a page with data including credit card details from the database.
6. The Web server sends the credit card details to the hacker.



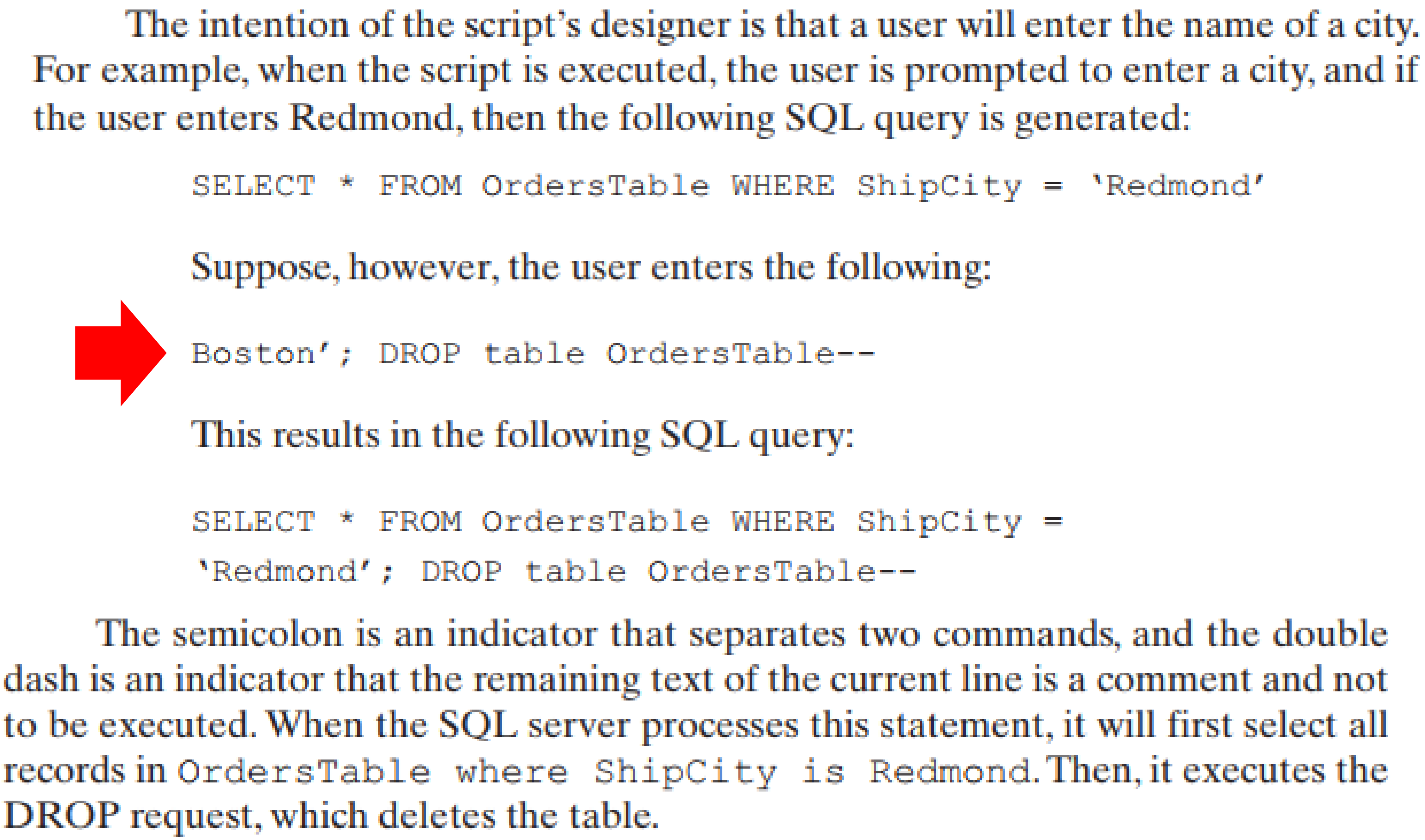
The Injection Technique

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.

As a simple example, consider a script that build an SQL query by combining predefined strings with text entered by a user:

```
var Shipcity;
ShipCity = Request.form (“ShipCity”);
var sql = “select * from OrdersTable where ShipCity = ‘” +
ShipCity + “‘”;
```

The Injection Technique



The intention of the script’s designer is that a user will enter the name of a city. For example, when the script is executed, the user is prompted to enter a city, and if the user enters Redmond, then the following SQL query is generated:

```
SELECT * FROM OrdersTable WHERE ShipCity = ‘Redmond’
```

Suppose, however, the user enters the following:

```
Boston’; DROP table OrdersTable--
```

This results in the following SQL query:

```
SELECT * FROM OrdersTable WHERE ShipCity =
‘Redmond’; DROP table OrdersTable--
```

The semicolon is an indicator that separates two commands, and the double dash is an indicator that the remaining text of the current line is a comment and not to be executed. When the SQL server processes this statement, it will first select all records in `OrdersTable where ShipCity is Redmond`. Then, it executes the DROP request, which deletes the table.

3 – Tier Architecture

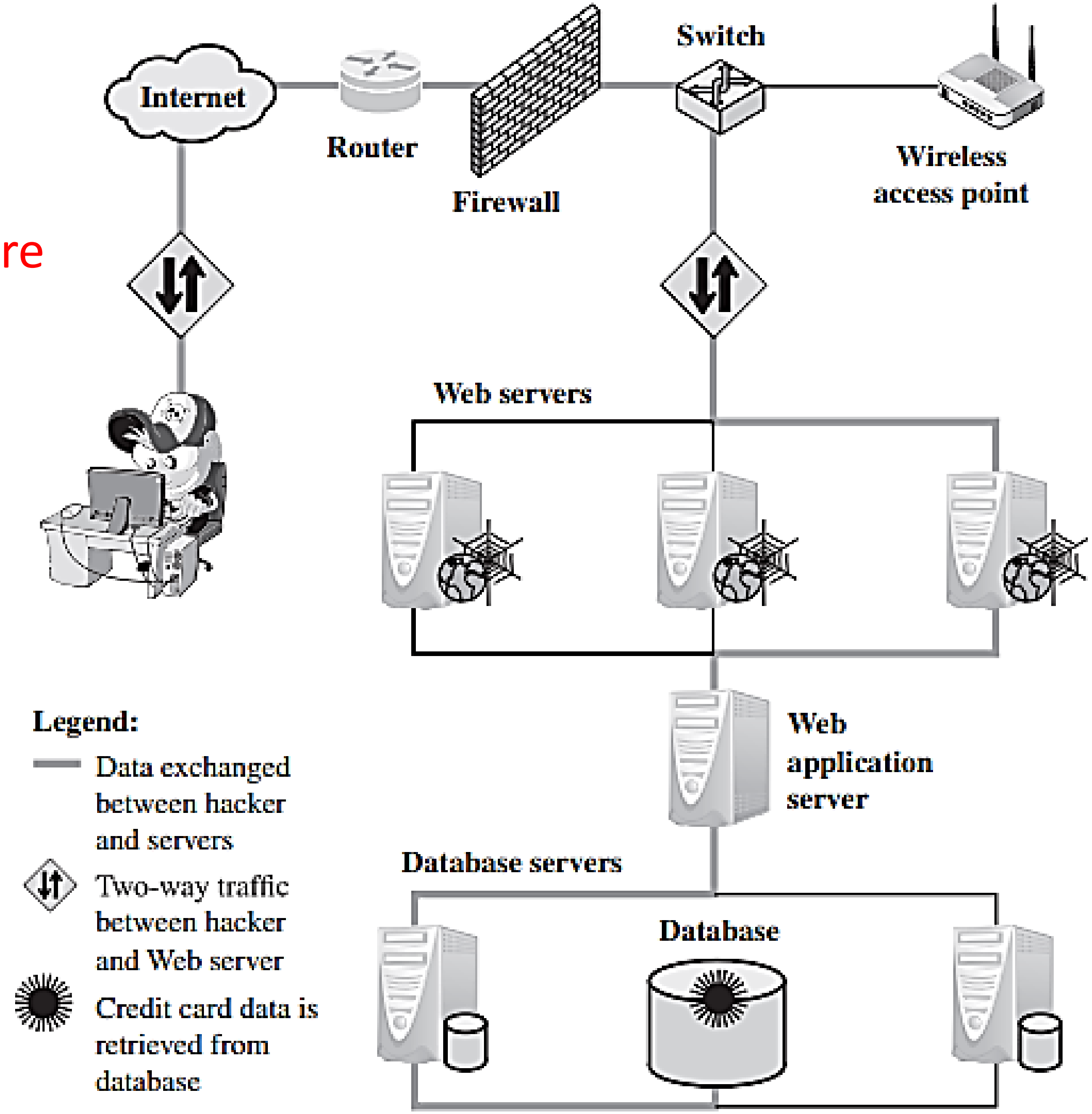
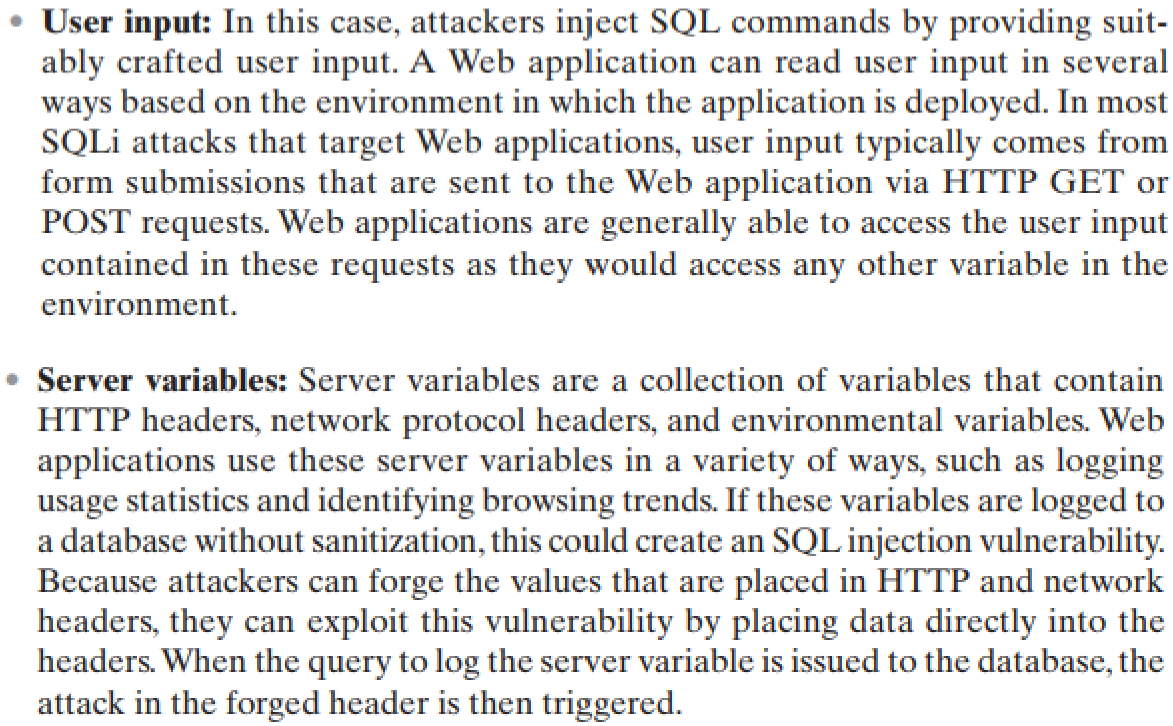

Internet
Router
Firewall
Switch
Wireless access point
Web servers
Web application server
Database servers
Database
Legend:
Data exchanged between hacker and servers
Two-way traffic between hacker and Web server
Credit card data is retrieved from database

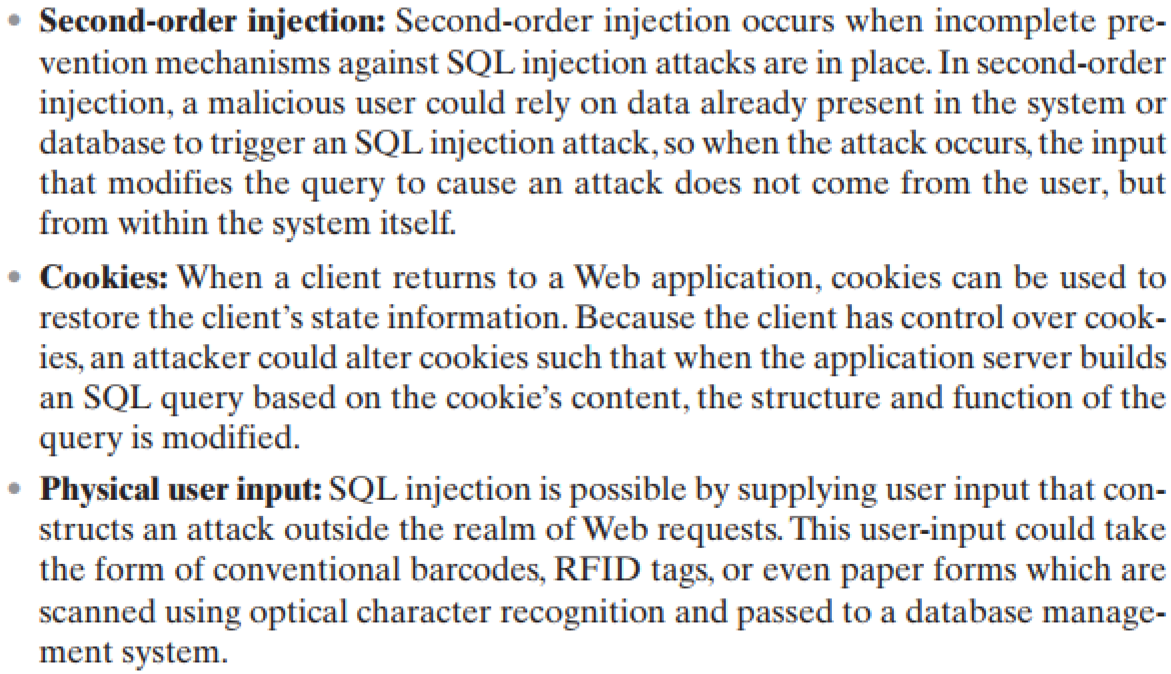
Figure 5.5 **Typical SQL Injection Attack**

SQLi Attack Avenues and Types



- **User input:** In this case, attackers inject SQL commands by providing suitably crafted user input. A Web application can read user input in several ways based on the environment in which the application is deployed. In most SQLi attacks that target Web applications, user input typically comes from form submissions that are sent to the Web application via HTTP GET or POST requests. Web applications are generally able to access the user input contained in these requests as they would access any other variable in the environment.

- **Server variables:** Server variables are a collection of variables that contain HTTP headers, network protocol headers, and environmental variables. Web applications use these server variables in a variety of ways, such as logging usage statistics and identifying browsing trends. If these variables are logged to a database without sanitization, this could create an SQL injection vulnerability. Because attackers can forge the values that are placed in HTTP and network headers, they can exploit this vulnerability by placing data directly into the headers. When the query to log the server variable is issued to the database, the attack in the forged header is then triggered.



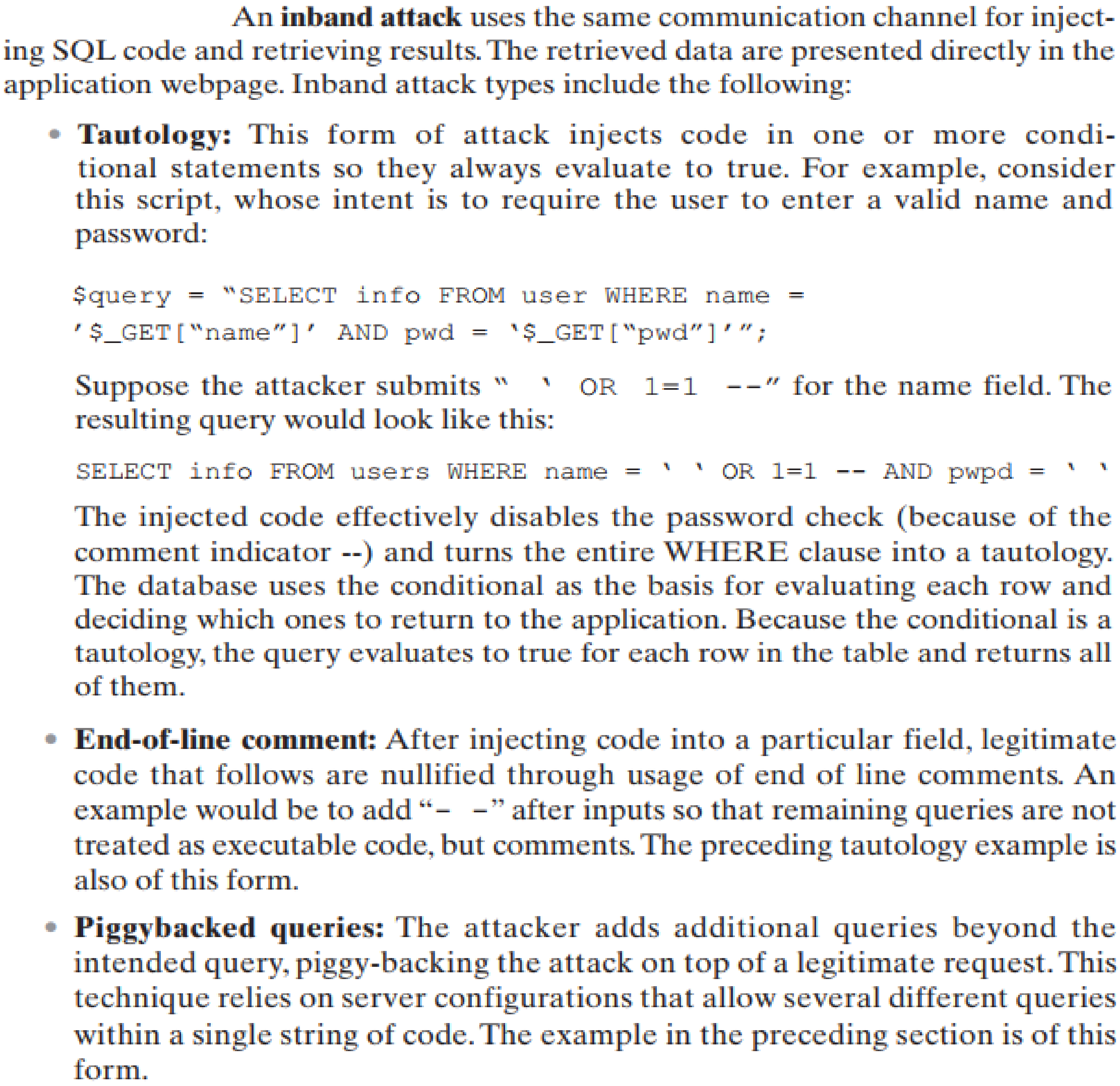
- **Second-order injection:** Second-order injection occurs when incomplete prevention mechanisms against SQL injection attacks are in place. In second-order injection, a malicious user could rely on data already present in the system or database to trigger an SQL injection attack, so when the attack occurs, the input that modifies the query to cause an attack does not come from the user, but from within the system itself.

- **Cookies:** When a client returns to a Web application, cookies can be used to restore the client's state information. Because the client has control over 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:** SQL injection is possible by supplying user input that constructs an attack outside the realm of Web requests. This user-input could take the form of conventional barcodes, RFID tags, or even paper forms which are scanned using optical character recognition and passed to a database management system.

Attack types can be grouped into three main categories: inband, inferential, and out-of-band.

- An **in-band** attack uses the same communication channel for injecting SQL code and retrieving results. The retrieved data are presented directly in the application webpage.
- With an **inferential attack**, 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.
- In an **out-of-band attack**, data are retrieved using a different channel (e.g., an e-mail 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.



An **inband attack** uses the same communication channel for injecting SQL code and retrieving results. The retrieved data are presented directly in the application webpage. Inband attack types include the following:

- **Tautology:** This form of attack injects code in one or more conditional statements so they always evaluate to true. For example, consider this script, whose intent is to require the user to enter a valid name and password:

  ```
  $query = “SELECT info FROM user WHERE name =
  ’$_GET[“name”]’ AND pwd = ‘$_GET[“pwd”]’”;
  ```

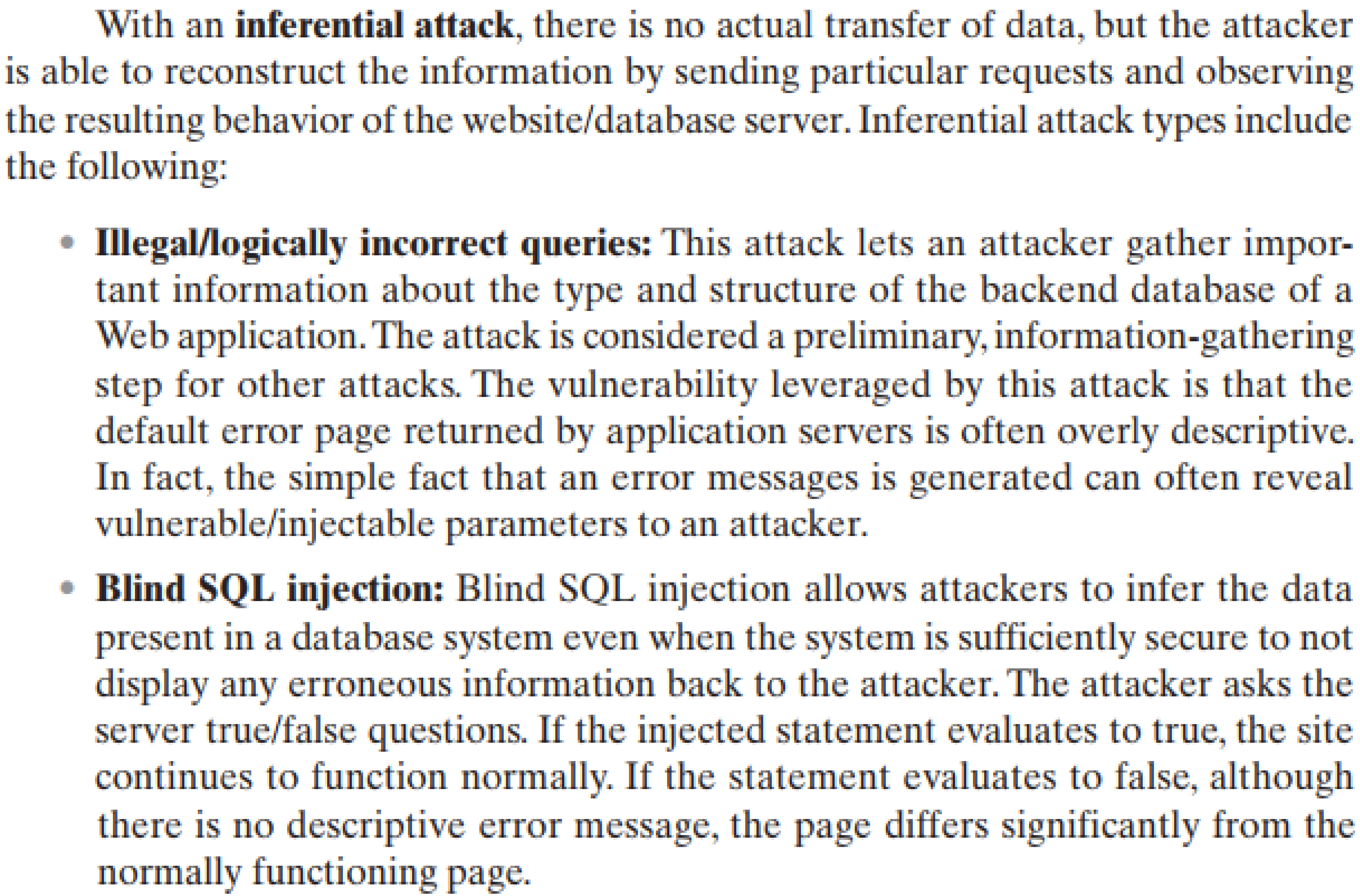
  Suppose the attacker submits “ ‛ OR 1=1 --” for the name field. The resulting query would look like this:

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

  The injected code effectively disables the password check (because of the comment indicator --) and turns the entire WHERE clause into a tautology. The database uses the conditional as the basis for evaluating each row and deciding which ones to return to the application. 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. An example would be to add “- -” after inputs so that remaining queries are not treated as executable code, but comments. The preceding tautology example is also of this form.

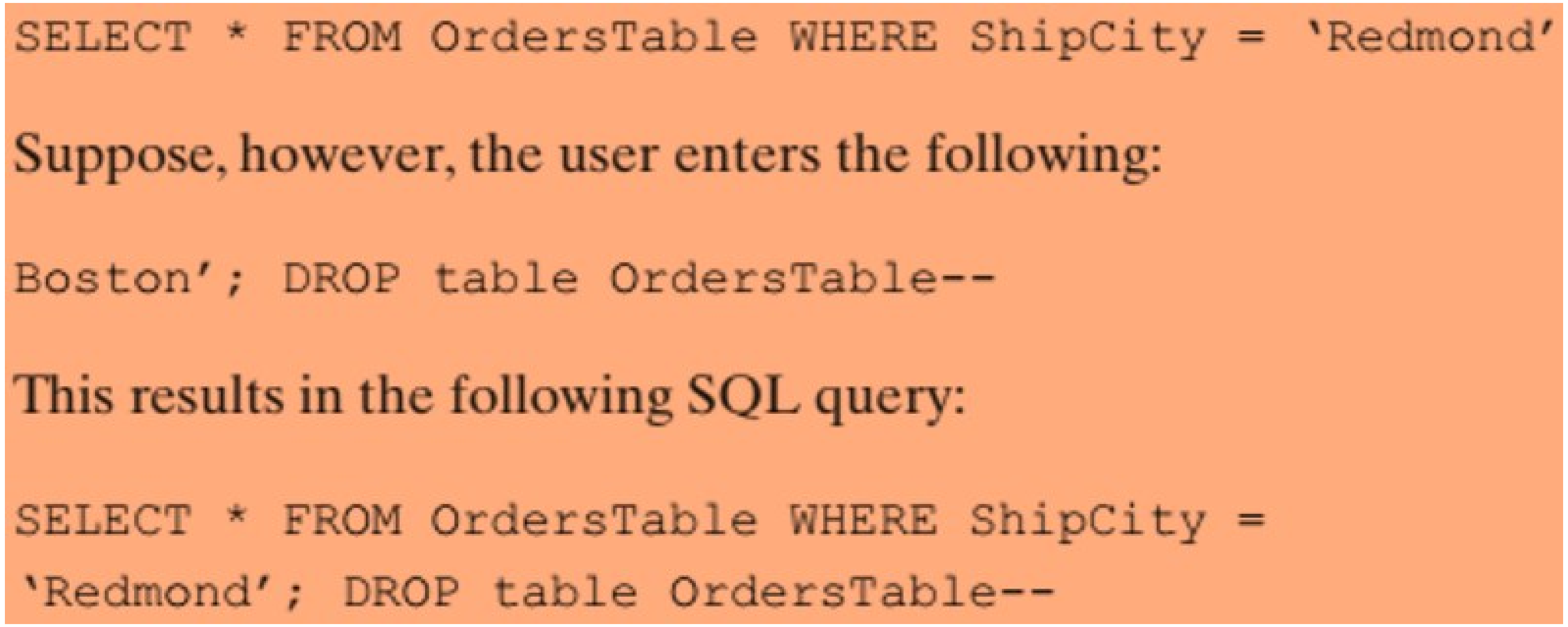
- **Piggybacked queries:** The attacker adds additional queries beyond the intended query, piggy-backing the attack on top of a legitimate request. This technique relies on server configurations that allow several different queries within a single string of code. The example in the preceding section is of this form.



With an **inferential attack**, 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 include the following:

- **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. The vulnerability leveraged by this attack is that the default error page returned by application servers is often overly descriptive. In fact, the simple fact that an error messages is generated can often reveal vulnerable/injectable parameters to an attacker.
- **Blind SQL injection:** 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. The attacker asks the server true/false questions. If the injected statement evaluates to true, the site continues to function normally. If the statement evaluates to false, although there is no descriptive error message, the page differs significantly from the normally functioning page.

SQL Injection – Summary



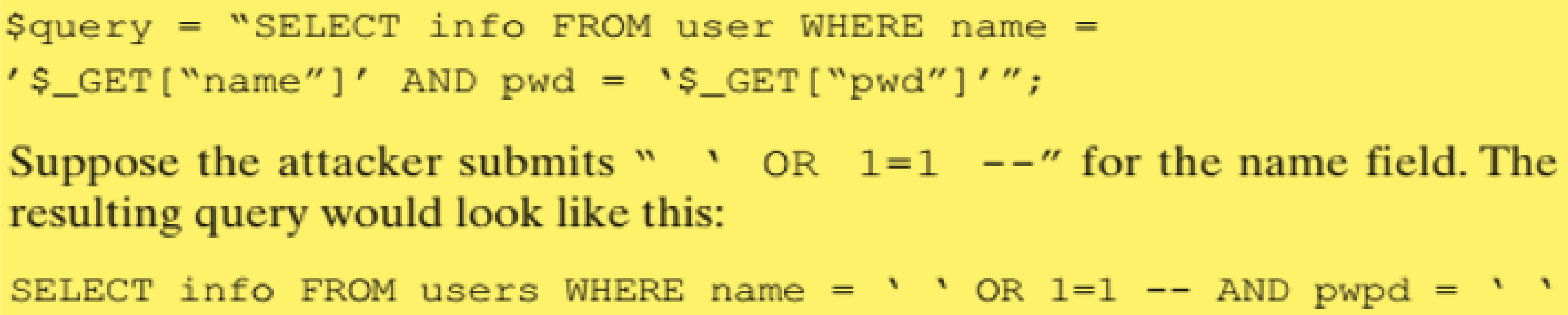
```
SELECT * FROM OrdersTable WHERE ShipCity = ‘Redmond’
```

Suppose, however, the user enters the following:

```
Boston’; DROP table OrdersTable--
```

This results in the following SQL query:

```
SELECT * FROM OrdersTable WHERE ShipCity =
‘Redmond’; DROP table OrdersTable--
```

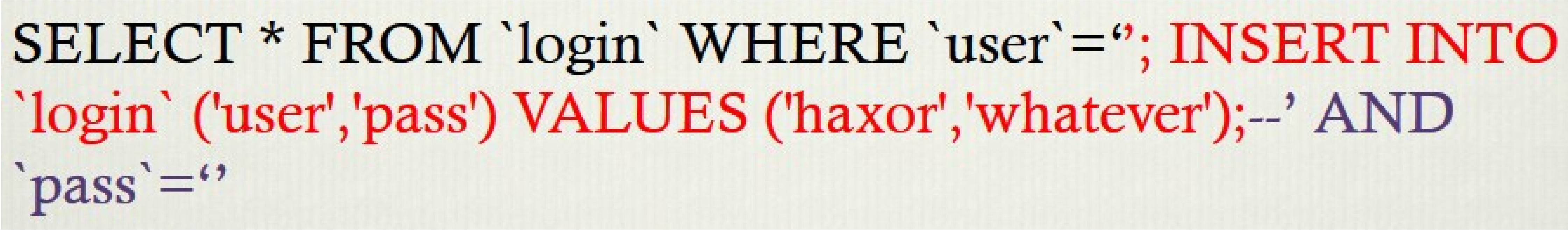


```
$query = “SELECT info FROM user WHERE name =
’$_GET[“name”]’ AND pwd = ‘$_GET[“pwd”]’”;
```

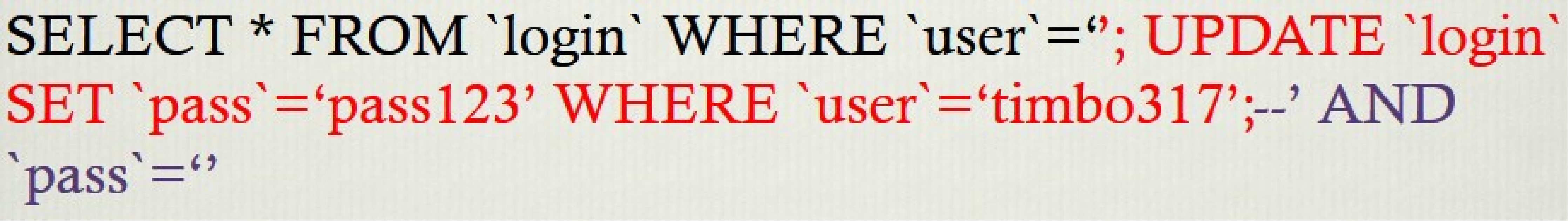
Suppose the attacker submits “ ‘ OR 1=1 --” for the name field. The resulting query would look like this:

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

SQL Injection – Summary

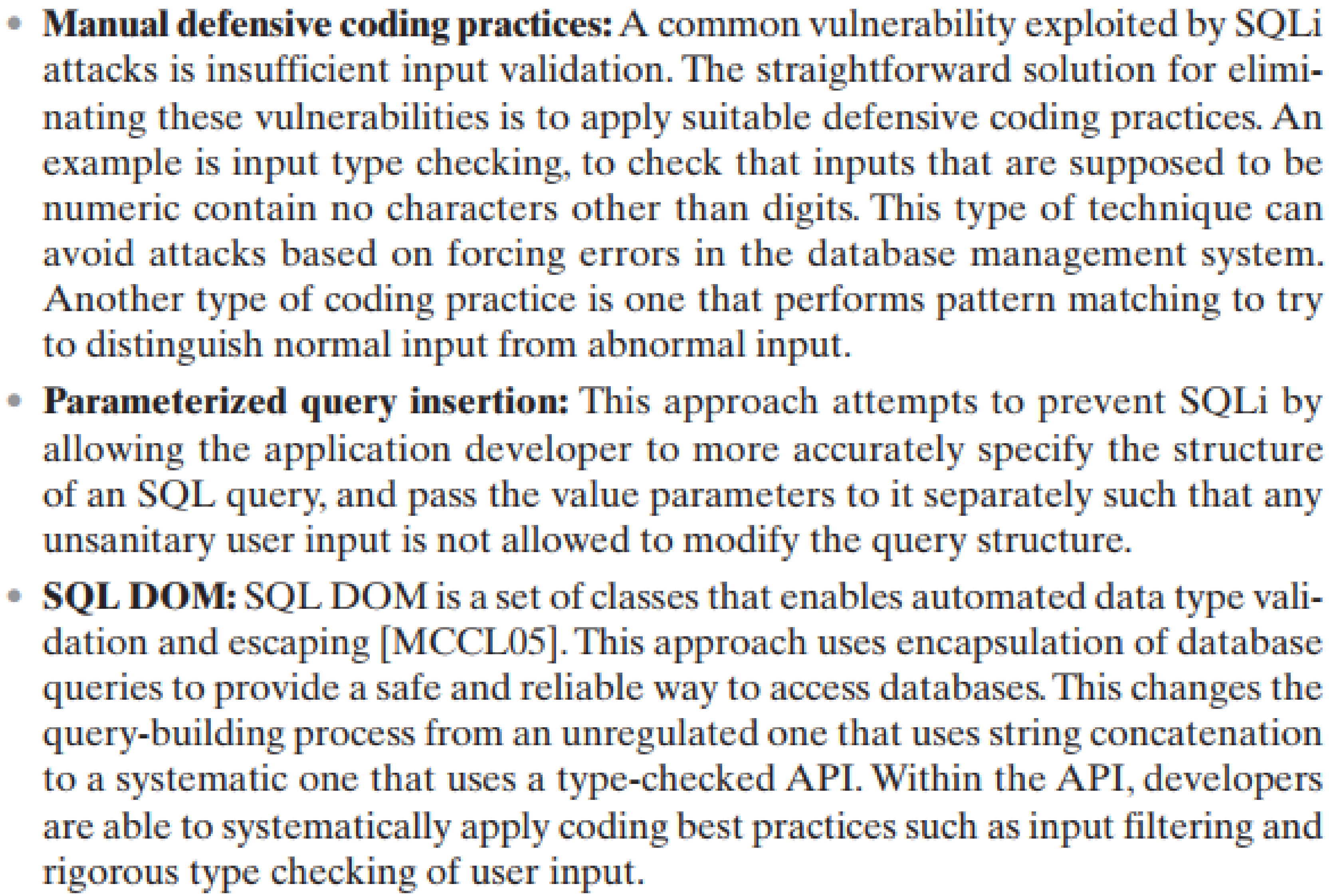


```
SELECT * FROM `login` WHERE `user`=‘’; INSERT INTO
`login` ('user','pass') VALUES ('haxor','whatever');--’ AND
`pass`=‘’
```



```
SELECT * FROM `login` WHERE `user`=‘’; UPDATE `login`
SET `pass`=‘pass123’ WHERE `user`=‘timbo317’;--’ AND
`pass`=‘’
```

SQLi Countermeasures (1)



- **Manual defensive coding practices:** A common vulnerability exploited by SQLi attacks is insufficient input validation. The straightforward solution for eliminating these vulnerabilities is to apply suitable defensive coding practices. An example is input type checking, to check that inputs that are supposed to be numeric contain no characters other than digits. This type of technique can avoid attacks based on forcing errors in the database management system. Another type of coding practice is one that performs pattern matching to try to distinguish normal input from abnormal input.
- **Parameterized query insertion:** This approach attempts to prevent SQLi by allowing the application developer to more accurately specify the structure of an SQL query, and pass the value parameters to it separately such that any unsanitary user input is not allowed to modify the query structure.
- **SQL DOM:** SQL DOM is a set of classes that enables automated data type validation and escaping [MCCL05]. This approach uses encapsulation of database queries to provide a safe and reliable way to access databases. This changes the query-building process from an unregulated one that uses string concatenation to a systematic one that uses a type-checked API. Within the API, developers are able to systematically apply coding best practices such as input filtering and rigorous type checking of user input.

Prepared Statement: Example

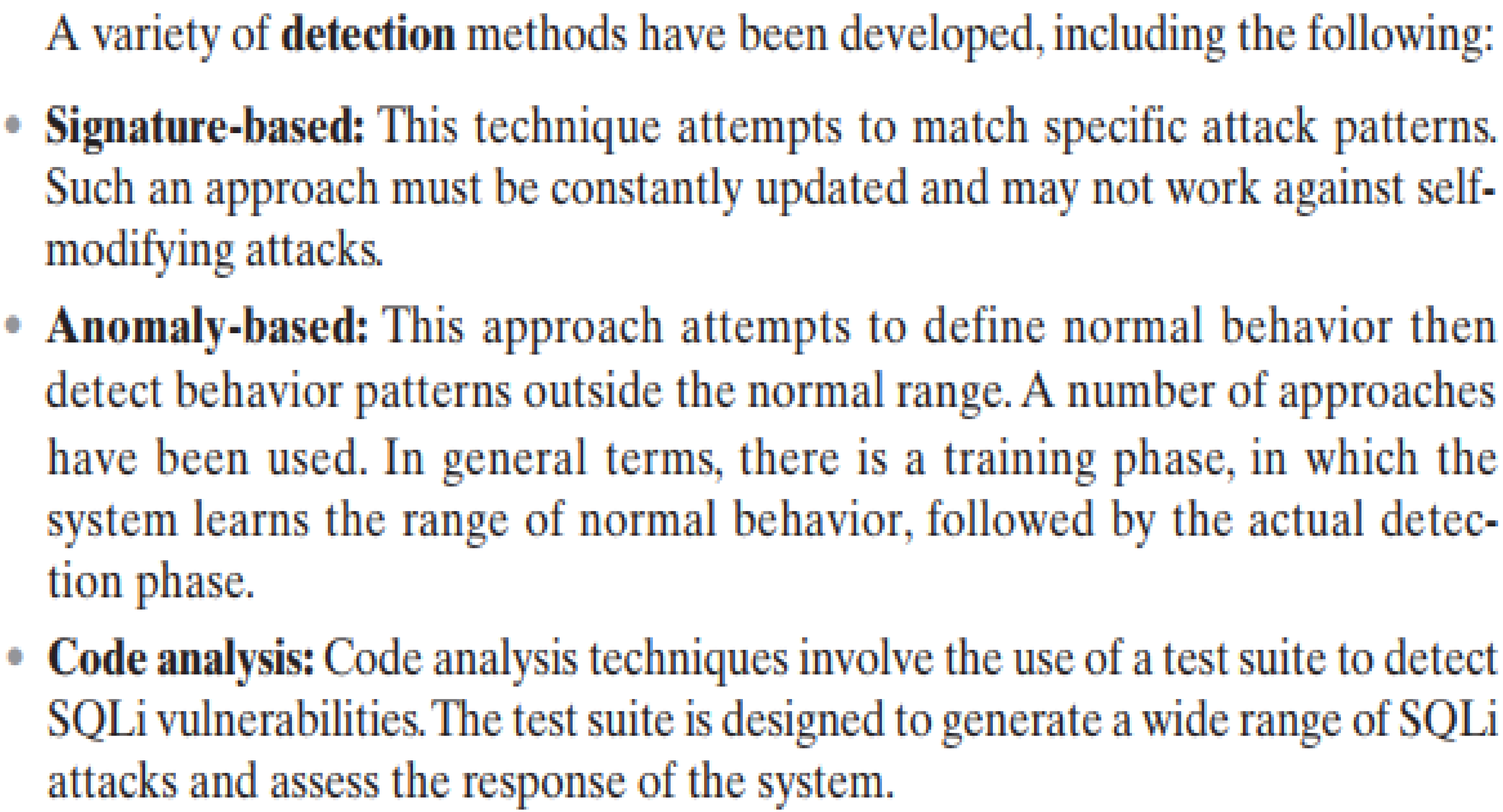
http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html

```
PreparedStatement ps =
    db.prepareStatement("SELECT pizza, toppings, quantity, order_day "
                 + "FROM orders WHERE userid=? AND order_month=?");
ps.setInt(1, session.getCurrentUserId());
ps.setInt(2, Integer.parseInt(request.getParamenter("month")));
ResultSet res = ps.executeQuery();
```

Bind variable (data placeholder)

- Query parsed without parameters
- Bind variables: ? placeholders guaranteed to be data (not control). Bind variables are typed (int, string, …)
- Prepared statements allow creation of static queries with bind variables → preserves the structure of intended query

SQLi Countermeasures (2)



A variety of **detection** methods have been developed, including the following:

- **Signature-based:** This technique attempts to match specific attack patterns. Such an approach must be constantly updated and may not work against self-modifying attacks.
- **Anomaly-based:** This approach attempts to define normal behavior then detect behavior patterns outside the normal range. A number of approaches have been used. In general terms, there is a training phase, in which the system learns the range of normal behavior, followed by the actual detection phase.
- **Code analysis:** Code analysis techniques involve the use of a test suite to detect SQLi vulnerabilities. The test suite is designed to generate a wide range of SQLi attacks and assess the response of the system.