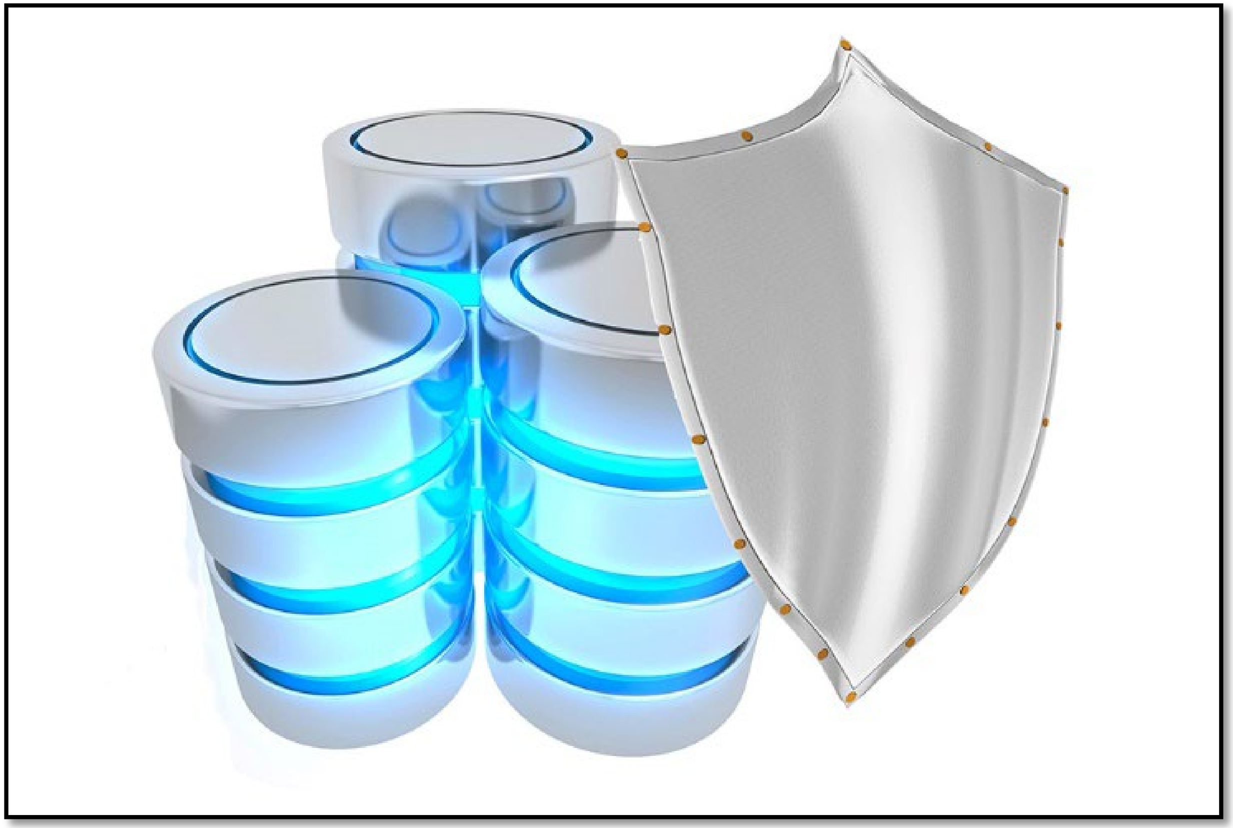
COMP3052.SEC Computer Security

Session 14: Database Security



Acknowledgements

- Some of the materials we use this semester may come directly from previous teachers of this module, and other sources ...
- Thank you to (amongst others):
  - Michel Valstar, Milena Radenkovic, Michael Pound, Dave Towey…

This Session

- Database Security
  - Privileges
  - SQL Security
  - Views
- Statistical Database Trackers
- SQL Injection

Introduction

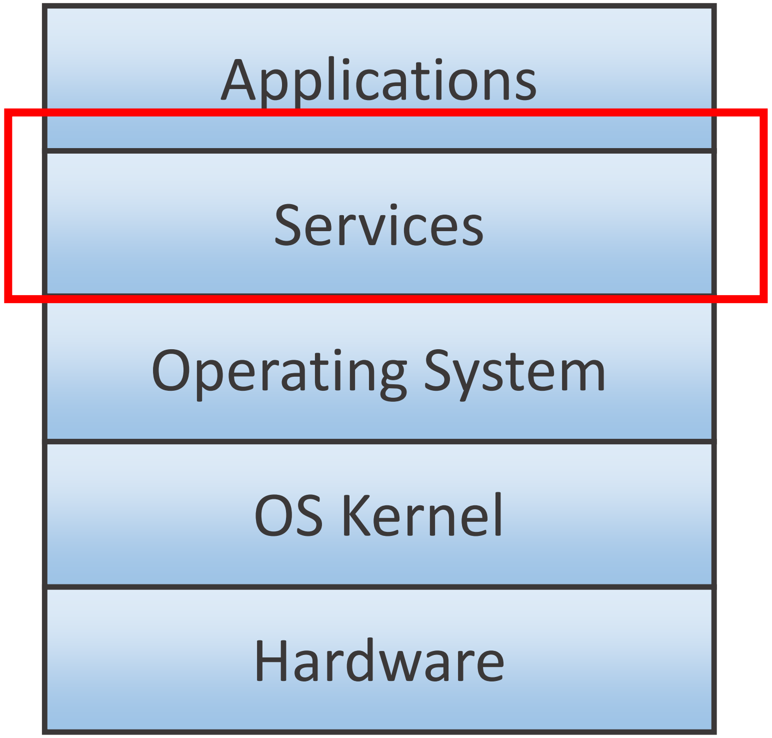
- Database security is concerned with information
    - Can look at the content
    - More man (or woman) than machine oriented

Protecting Information

- Protecting sensitive information is hard
- Attackers may be interested in many types of information:
    - Exact data
    - Bounds
    - Existence / Negative results
    - Probable value
- Balance protecting sensitive information and utility

Security Model

- DBMS security is defined at the services layer, above the kernel and OS
- Some security may be in the application layer, e.g., view-based policies
- DBMS enforces access control policies and maintains consistency



SQL Security

- Three Entities
  - Users
  - Actions
  - Objects
- Users invoke actions on objects
- Newly created objects are owned by the creator
- Privileges can be granted:
  - Granter, Grantee, Object, Action, Grantable

Privilege Granting / Revoking

```
GRANT SELECT, UPDATE (Day, Flight)
ON TABLE Diary
TO Sam, Zoe
WITH GRANT OPTION
```

```
REVOKE UPDATE
ON TABLE Diary
FROM Sam
```

- Grant revocation cascades to all grantees of revoked grantee – safer than not doing this

View-based Security

- Views are derived relations:

```
CREATE VIEW pharm_order AS
  SELECT DrugDB.Name, SUM(Total)
    FROM Patients, DrugDB
    GROUP BY (DrugDB.Name)
WITH CHECK OPTION
```

Why use views?

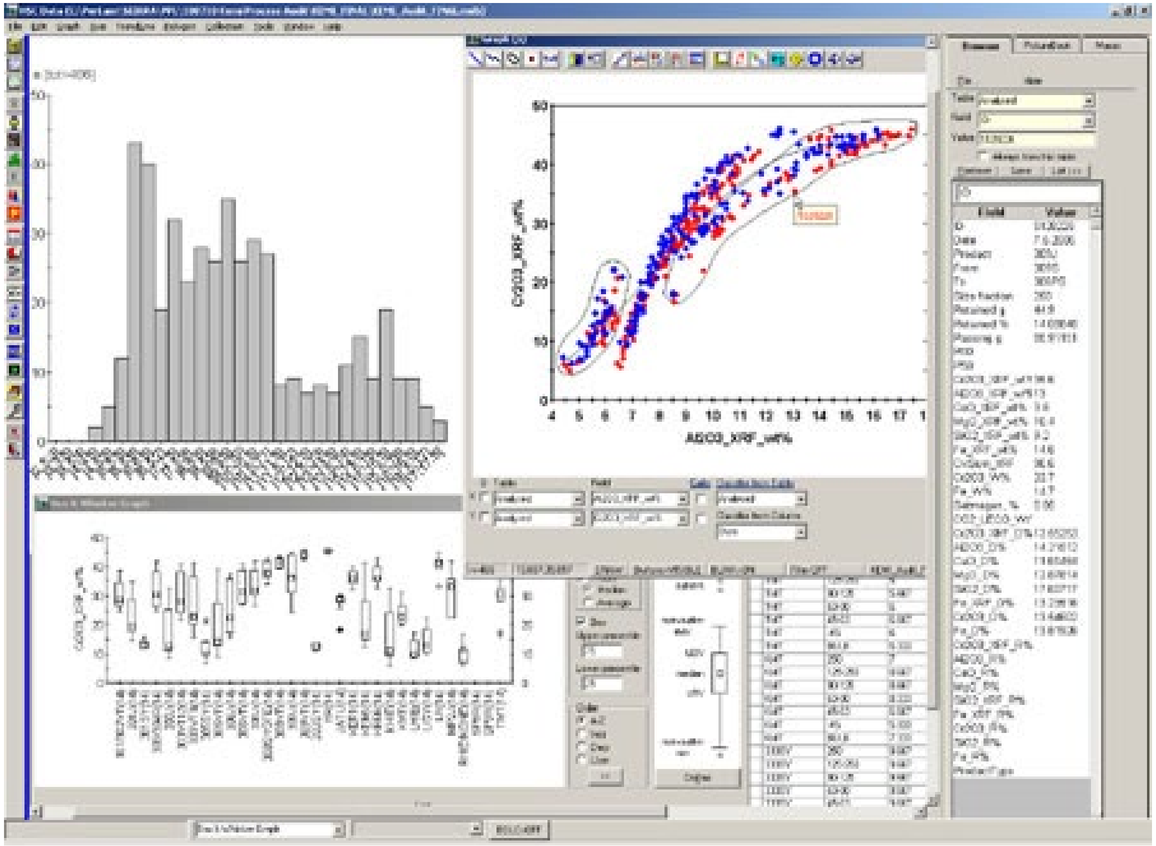
- Views are a flexible way of creating policies closer to application requirements
- Views can enforce context-dependent and data-dependent policies
- Views can implement controlled invocation
- Data can be easily reclassified

Why not?

- INSERT / UPDATE actions depend on the CHECK options, else might be blind inserts
- Definitions must be correct in order to capture intended security policy
- Completeness and consistency are not achieved automatically
- Can quickly become very inefficient

Statistical Databases

- Where access to data is restricted, access to aggregates might still be permitted:
  - COUNT
  - SUM
  - AVG
  - MAX
  - MIN



Inference

- Since individual items are sensitive, we cannot permit access
- Statistical queries are useful, but by definition refer to the data
- Some queries can reveal information on the underlying data – Covert Channel

Tracking

- Direct attack
  - Aggregate is computed to capture information of individual data elements
- Indirect
  - Combines information from several aggregates
- Tracker Attack
  - Generalised indirect attack

Salaries

- S = The sum of all salaries in the department
- T = The sum of all salaries for the department except those that have “Head of School” as “Position”
- Boss’ salary = S – T

Do not allow sets of just one

Salaries

- S = Sum of all salaries
- T = Sum of all salaries of women, and anyone whose first name is Albert
- U = The sum of all men’s department salaries
- Albert’s salary = T + U – S

Do not allow conditions that refer to just one

Salaries

- S = Sum of all salaries
- Number of department heads named Albert is not allowed
- T = sum of all salaries for those named Albert
- U = The sum of all salaries for department heads
- V = The sum of all salaries for those who are not department heads, and not named Albert
- Salaries of DHs named Albert = V + T + U - S

Further Defences

- Data swapping – Swap records but keep stats the same
- Noise addition – Alter aggregate output (a little)
- Table splitting – Separate data completely
- User tracking – Log queries

SQL Injection Attacks

- It’s common for user input to be read (e.g., in a web form) and then used within an SQL query:

  https://insecure-website.com/products?category=Gifts

```
$query = "SELECT * FROM products WHERE category =
'Gifts' AND released = 1";
```

- Unexpected user input can completely rewrite the query. An attacker can construct an attack like:

  https://insecure-website.com/products?category=Gifts’--

```
$query = "SELECT * FROM products WHERE category =
'Gifts'--' AND released = 1";
```

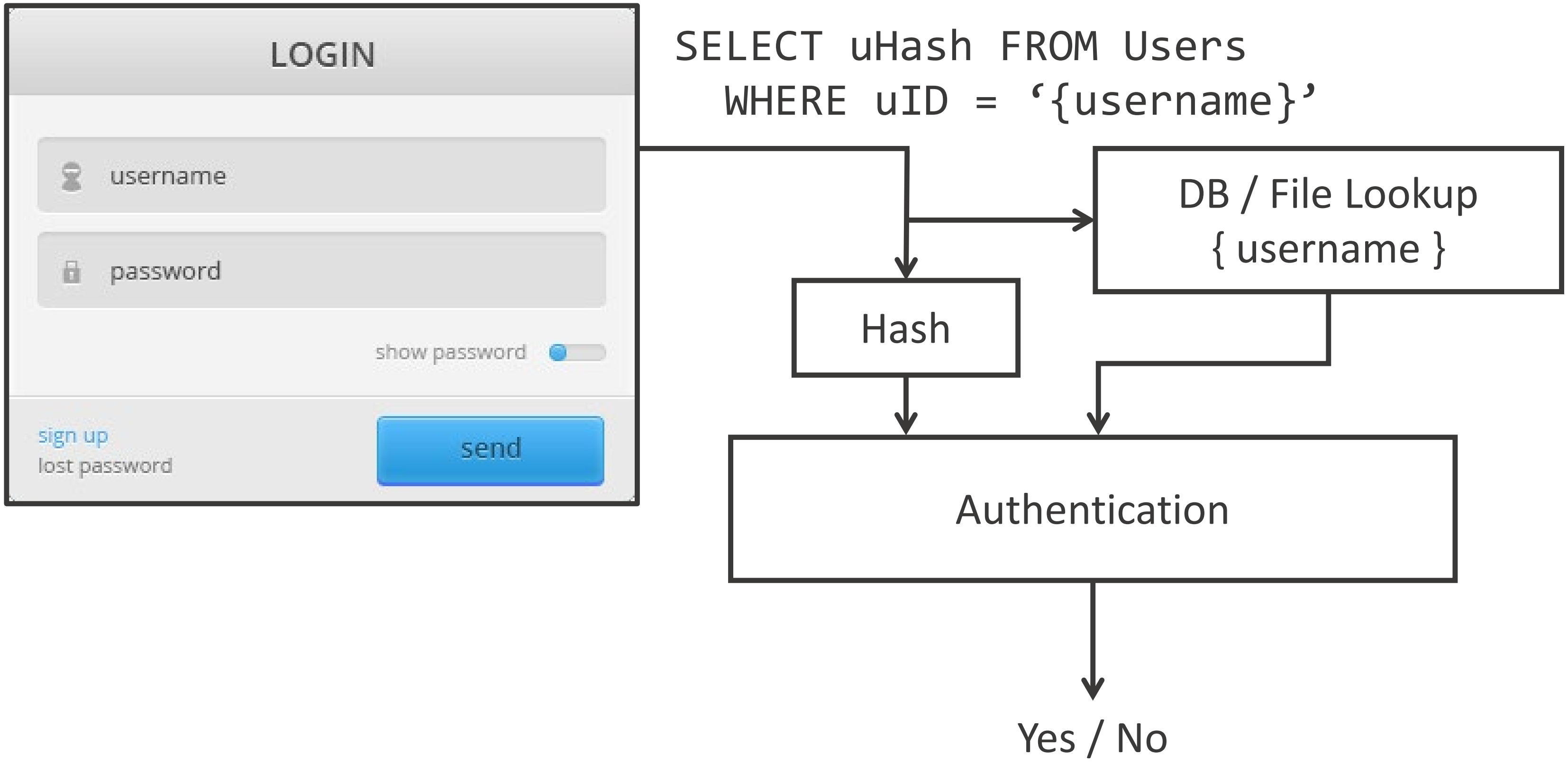
- It no longer includes AND released = 1. This means that all products are displayed, including unreleased products

SQL Injection Attacks

- An application or website is vulnerable to an injection if it doesn’t filter SQL control characters:
  - ' represents the beginning or end of a string
  - ; represents the end of a command
  - /*…*/ represent comments
  - -- represents a comment for the rest of the line

SQL Injection Attacks

- Login pages will request hashes from the database



```
SELECT uHash FROM Users
  WHERE uID = ‘{username}’
```

DB / File Lookup
{ username }

Hash

Authentication

Yes / No

Retrieving from other DB Tables

- An attacker can leverage a SQL injection vulnerability to retrieve data from other tables within the database

- This is done using the **UNION** keyword

```
$query = "SELECT name, description FROM products
WHERE category = 'Gifts’ UNION SELECT username,
password FROM users--";
```

- This will cause the application to return all usernames and passwords along with the names and descriptions of products

UNION

- UNION appends (not joins) two tables together
  - They must have the same number of columns

```
http://shop.com/search.php?terms=hammers+nails
```

Returns a table of items and prices and quantity of any items matching the terms hammers and nails

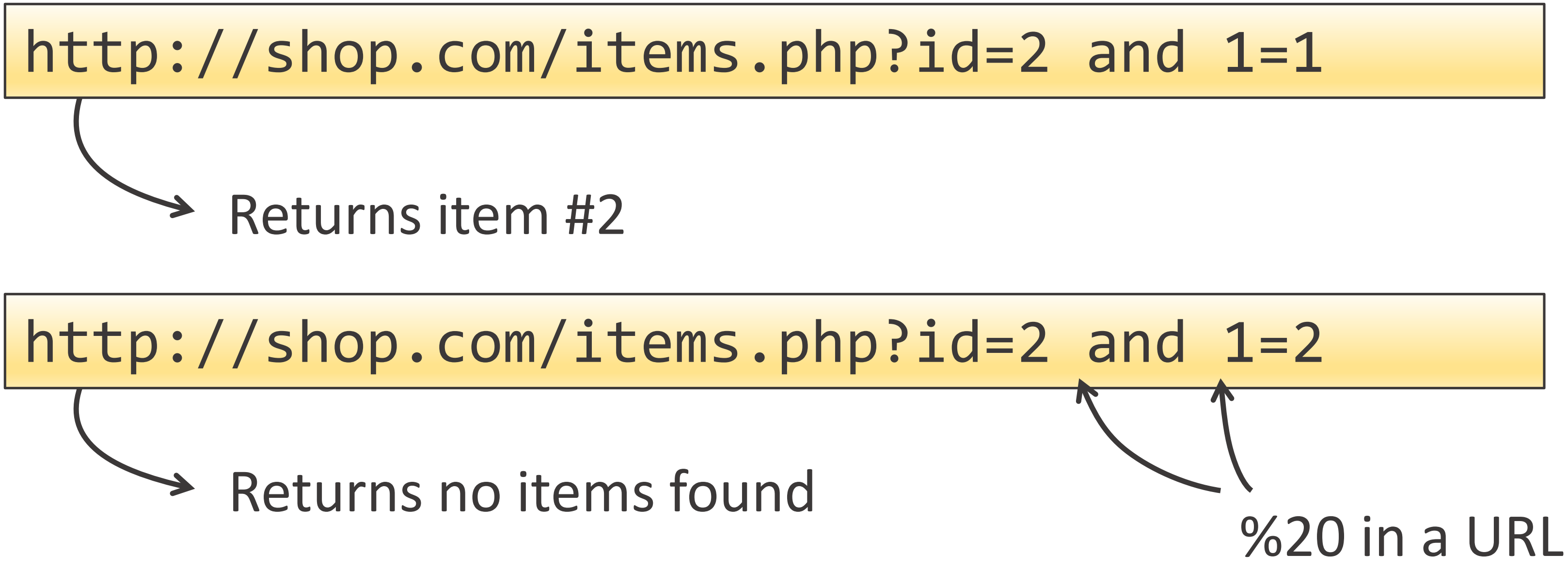
```
http://shop.com/search.php?terms=hammers+' UNION SELECT
1,ids,hashes FROM users;--
```

Appends the user table!

Blind SQL Injection

- Most servers won’t directly output SQL errors to the screen
- A blind SQL injection performs database analysis without any actual output

```
http://shop.com/items.php?id=
```



```
http://shop.com/items.php?id=2 and 1=1
```

Returns item #2

```
http://shop.com/items.php?id=2 and 1=2
```

Returns no items found

%20 in a URL

Fingerprinting the DB

- Some commands are specific to an individual DBMS:

```
http://shop.com/items.php?id=2; waitfor delay '0:0:10'--
```

Waits for a while on MS SQL Server but not MySQL

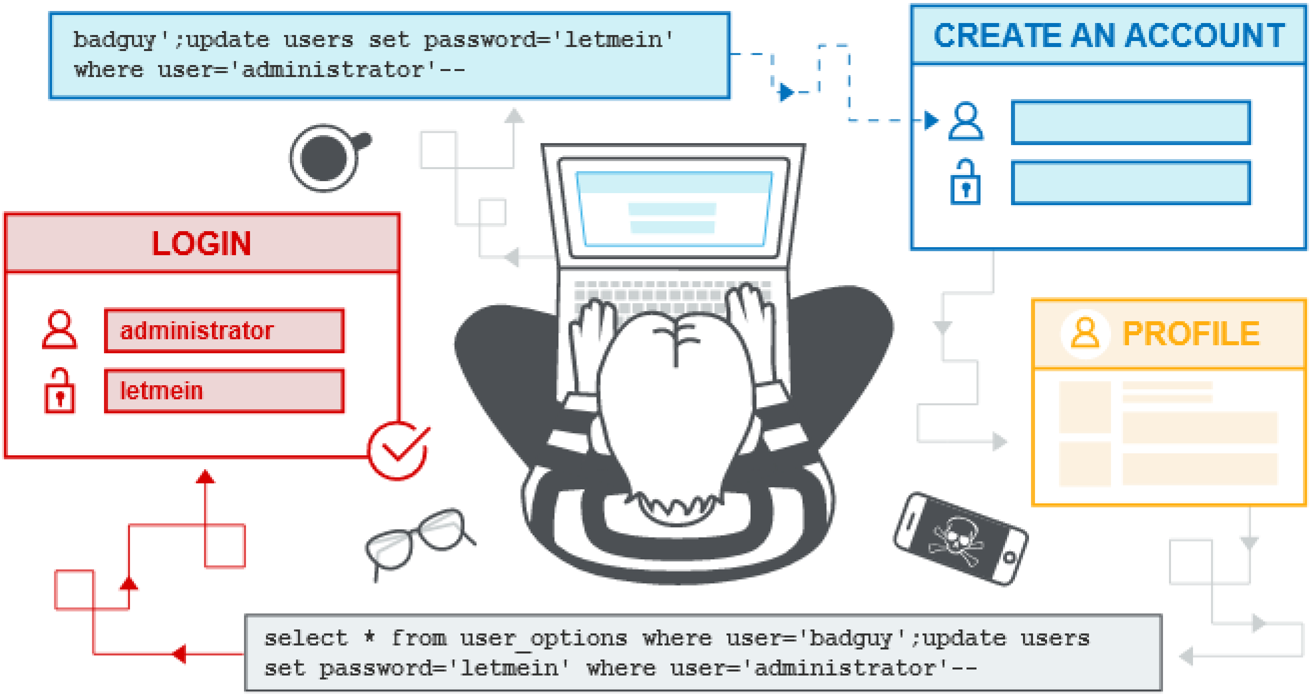
- Once you know the DB, access the system tables:

```
http://shop.com/items.php?id=2 AND 1=(SELECT COUNT(*)
FROM information_schema.tables WHERE TABLE_NAME='users')
```

If an item returns, there is a table named ‘users’ in MySQL

Second Order SQL Injection

- Entry points may be checked for special characters, but internal functions?
- Store the exploit in one pass, then have it executed later



```
badguy';update users set password='letmein'
where user='administrator'--
```

```
select * from user_options where user='badguy';update users
set password='letmein' where user='administrator'--
```

Summary

- Database Security
  - Privileges
  - SQL Security
  - Views
- Statistical Database Trackers
- SQL Injection (https://portswigger.net/web-security/sql-injection)

Gollmann Chapter 9, 10.5.2