Data & Information – Test 4 (1.5 hours)

18 June 2018, 13:45–15:15

Program: Technical Computer Science / Business & IT
Module: Data & Information (201700279)
Module Coordinator: Klaas Sikkel
Responsible Teachers: Maurice van Keulen / Thomas Hupperich

Please note:

- Please answer every question on a different sheet of paper (the answers will be distributed to different person for grading).
- You are not allowed to bring any study materials to the test; essential excerpts from the study materials are available as appendices. You don't need a calculator.

Grade = #points/10

Question 1: Security (40 points)

1.1

Users of a Web platform are given the feature to write private messages to other users. The following PHP code snippet is part of a script that implements this functionality. The fields *from* and *message* in the variable *$_PM* are input given by the sender. When a users receives a message, these fields will be displayed in the browser by the following code:

```
echo "New private message from " . $_PM['from'] . " : [" . $_PM['message'] . "]";
```

a) What kind of vulnerability could this code yield?

b) How could an attacker exploit this vulnerability

c) What could an attacker achieve by exploiting this vulnerability?

1.2

The following code is part of an authentication script presenting the fields *username* and *password* to a user who wants to log in. After these fields are entered the script will check if there are any entries in the *$result* array and if so, log the user in as the first entry.

```
$user = $_POST['username'];
$pass = $_POST['password'];
$query = "SELECT * FROM users WHERE username = ' " . $user . " '
                    AND password = ' " . secure_hash($pass) . " ';";
$result = mysql_query($query);
```

a) Is the above code insecure and if so, why?

b) Give an example attacker input (the fields *username* and *password*) which will log the attacker in as the user "*admin*" without the attacker having to know the correct password. The user "*admin*" is the first user in the table *users*.

c) What is the best way, a developer can prevent such an attack?

1.3

A website implements advertisements of an external provider. This provider is malicious and delivers the following code within one of its ads:

```
<img src="http://serious-bank.com/transfer.php?IBAN=NL1337h4x0r&amount=1000"
                                                    width="0" height="0" border="0">
```

The *Serious Bank* is a reliable bank with a disastrous IT development department that does not implement any security mechanisms and website actions are generally handled via GET requests.

a) What kind of vulnerability may be in this scenario?

b) What would happen if a user is presented the malicious ad with the given code? In what case would this attack be successful and in which case would it fail?

c) How can such an attack be prevented by *Serious Bank*?

Question 2: XML & JSON (40 points)

We base ourselves on the movie database (see Appendix 2 for the schema of the movie db).

Tip: See Appendix 1 for an informal syntax of SQL including types, functions and operators important for xml and json handling.

a) [SQL/XML] Given the SQL-query below, adapt it using the SQL/XML standard such that it produces the result as XML. The result should have one row per actor, which contains one column 'xml' containing an element "actor" with attributes "pid" and "moviecount" and as contents the name of the actor.

```
SELECT p.pid, p.name, COUNT(m.mid) AS moviecount
FROM acts a, person p, movie m
WHERE a.pid = p.pid
  AND a.mid = m.mid
GROUP BY p.pid, p.name
```

An example result looks like:

xml
<actor pid="6526" moviecount="4">Marlon Brando</actor>
<actor pid="6529" moviecount="7">Robert Duvall</actor>
<actor pid="6538" moviecount="1">Gianni Russo</actor>
…

b) [SQL/XML] Adapt the query of the previous question, such that the name of the actor is also an attribute and that the contents of the actor-element contain the names of the movie (s)he played in as movie-elements.

An example result looks like:

xml
<actor pid="6526" name="Marlon Brando" moviecount="4"> <movie>Apocalypse Now</movie> <movie>On the Waterfront</movie> <movie>Godfather, The</movie> <movie>Streetcar Named Desire, A</movie> </actor>
<actor pid="6529" name="Robert Duvall" moviecount="7"> <movie>Network</movie> <movie>MASH</movie> <movie>Sling Blade</movie> <movie>Godfather: Part II, The</movie> <movie>Godfather, The</movie> <movie>To Kill a Mockingbird</movie> <movie>Apocalypse Now</movie> </actor>
<actor pid="6538" name="Gianni Russo" moviecount="1"> <movie>Godfather, The</movie> </actor>
…

c) [XML querying] A table ‘x’ contains an attribute ‘xml’ which is of type ‘xml’ and contains the actor elements as in the example result of 2(b) above.

i. Give the XPath that produces all names of actors who only played in the movie “Godfather, The”.

ii. What is the result of the query “`SELECT xpath('/movie/@name', x.xml) FROM x`” if the table ‘x’ contains only the three rows as shown in 2(b) above. Describe both the contents as well as the type(s) of the resulting column(s).

d) [JSON construction] Given the SQL-query below which produces a list of JSON objects with attributes ‘id’, ‘name’, and ‘roles’. Adapt the query, such that the ‘roles’ attribute is not an array of strings (array of role names), but an array of json-objects each containing both the role name as well as the movie name.

```
SELECT json_build_object('id',p.pid, 'name',p.name,
                         'roles',jsonb_agg(a.role))
FROM acts a, person p
WHERE a.pid = p.pid
GROUP BY p.pid
```

An example result looks like:

json_build_object
{"id" : 6526, "name" : "Marlon Brando", "roles" : [{"role": "Don Vito Corleone", "movie": "Godfather, The"}, {"role": "Colonel Walter E. Kurtz", "movie": "Apocalypse Now"}, {"role": "Terry Malloy", "movie": "On the Waterfront"}, {"role": "Stanley Kowalski", "movie": "Streetcar Named Desire, A"}]}
{"id" : 6529, "name" : "Robert Duvall", "roles" : [{"role": "Tom Hagen", "movie": "Godfather, The"}, {"role": "Tom Hagen", "movie": "Godfather: Part II, The"}, {"role": "Arthur \"Boo\" Radley", "movie": "To Kill a Mockingbird"}, {"role": "Lieutenant Colonel Kilgore", "movie": "Apocalypse Now"}, {"role": "Mr. Childers", "movie": "Sling Blade"}, {"role": "Major Frank Burns", "movie": "MASH"}, {"role": "Frank Hackett", "movie": "Network"}]}
{"id" : 6538, "name" : "Gianni Russo", "roles" : [{"role": "Carlo Rizzi", "movie": "Godfather, The"}]}
…

e) [JSON querying] Given the SQL-query below which obtains the “id” of all actors with name “Robert Duvall” from a table “x” which has an attribute “json” containing json-objects as in the example result of the previous question. Rewrite the WHERE-clause such that the result of the query remains the same, but it now uses the “@>” operator instead of the “->>” operator.

```
SELECT x.json->>id
FROM x
WHERE x.json->>name = “Robert Duvall”
```

f) [JSON] For each of the statements below, say whether they are true or false.

i. A JSON array can contain JSON scalar values

ii. A JSON array can contain JSON objects

iii. A JSON array can contain JSON arrays

Question 3: Tree-shaped data / Pathfinder (20 points)

We base ourselves again on the movie database (see Appendix 2 for the schema of the movie database).

Tip: See Appendix 1 for an informal syntax of SQL.

a) [Pathfinder] Given the small XML-document below, assign to each of the nodes its pre-order and post-order rank. Write down the full resulting table according to the Pathfinder document table structure: pre, post, level, kind, name, value.

```
<actor pid="6526" name="Marlon Brando" moviecount="4">
   <movie>Apocalypse Now</movie>
   <movie>On the Waterfront</movie>
   <movie>Godfather, The</movie>
   <movie>Streetcar Named Desire, A</movie>
</actor>
```

b) [Dewey numbering] Given an XML node with Dewey number 1.4.3. Which of the following statements is true?

   i. The parent of this node has Dewey number 4.3.

   ii. This node has 2 preceding siblings.

   iii. It is possible to determine the level of the node from its Dewey number.

c) [Pathfinder] Translate the XPath
`//movie[text()=”Apocalypse Now”]/parent::actor/@name`
according to the Pathfinder approach to an SQL-query that produces the right result given the table of the previous question. The result of the query on this table should be the pre-order rank of the resulting ‘name’ attributes. According to XPath semantics, the result should be duplicate-free and in document order. Obviously, the query should also work on any XML- fragment that is structured in this way.

d) [Pathfinder] PostgreSQL stores XML and JSON values in a column of a table as a special type. With the Pathfinder approach such values are shredded, i.e., stored in a table with a fixed set of attributes. What is the main purpose of doing so? Explain your answer.

Appendix 1: Informal syntax of SQL

In the informal syntax, we use the following notations

- A | B to indicate a choice between A and B
- [A] to indicate that A is optional
- A* to indicate that A appears 0 or more times
- A+ to indicate that A appears 1 or more times
- 'A' to indicate that the symbol A is literally that symbol

We are not precise in punctuation in the syntax, but this is irrelevant in this exam anyway.

SQL

createtable: CREATE TABLE tablename '(' columndef+ constraint* ')'
createview: CREATE VIEW viewname AS query
query: SELECT (column [AS colname])+ FROM (tablename [AS colname])+ WHERE condition
[GROUP BY column+] [ORDER BY column+]

columndef: colname type [NOT NULL] [UNIQUE] [PRIMARY KEY] [REFERENCES tablename (colname+)]
constraint: PRIMARY KEY (colname, ...)
| FOREIGN KEY(colname, ...) REFERENCES tablename(colname, ...) | CHECK (condition)
column: [tablename '.'] colname | '*'
sqlxml: XMLELEMENT([NAME colname] , column*) | XMLATTRIBUTES(column*) | XMLFOREST(column*)
| XMLAGG(column*)

Examples of condition:
column = value [(OR | AND) [NOT] column <> value]
| column IS [NOT] NULL
| column [NOT] IN (value, ...)
...
xmljson-functions: xpath(constant,…), unnest(…), to_jsonb(…), jsonb_build_object(…,…,…),
json_build_array(…,…,…), json_agg(…)
json operators: @>, ->, ->>, ||, -

Appendix 2: Database schema for movie database

- **Movie**: mid INTEGER PRIMARY KEY, name TEXT, year NUMERIC(4,0), plot_outline TEXT, rating NUMERIC(2,1)
- **Person**: pid INTEGER PRIMARY KEY, name TEXT
- **Acts**: mid INTEGER, pid INTEGER, role TEXT
- **Directs**: mid INTEGER, pid INTEGER
- **Writes**: mid INTEGER, pid INTEGER
- **Genre**: mid INTEGER, genre TEXT
- **Language**: mid INTEGER, language TEXT
- **Certification**: mid INTEGER, country TEXT, certificate TEXT
- **Runtime**: mid INTEGER, country TEXT, runtime TEXT