Data & Information – Test 4 (1.5 hours)

21 June 2017, 13:45–15:15

Program: Technical Computer Science / Business & IT
Module: Data & Information (201300180)
Module Coordinator: Klaas Sikkel

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.

Grade = #points/10

Question 1: Security (40 points)

a) Consider the following fragment of PHP code:

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

It is called as part of a script that displays new private messages for users. The *from* and *body* fields in the $_PM variable are input by a user wishing to send a private message and stored in the database for later retrieval by the above script.

How could an insecure implementation of the above result in an attacker hijacking a victim's session? What do we call this kind of vulnerability?

b) Consider the following fragment of PHP code implementing part of a user-based authentication scheme:

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

This code is called as part of a login script presenting the user with a username and password field and which will check if there are any entries in the $result array and if so, log the user in as the first entry.

Is the above code insecure and if so, why? Give an example attacker input (in the *user* and *pass* fields) which will log the attacker in as the *admin* user without the attacker having to know the correct password and mention (in one or two sentences) how a developer can ***best*** prevent what you think is the vulnerability here.

You may assume the *admin* user is the first user in the *users* table.

c) Consider a web application which stores its users' credentials in hashed form as follows:

```
store_db(md5($_POST['password'], $salt));
```

Explain why the above approach is insufficiently secure to reduce the impact of for example a database breach. Make sure to explicitly mention at least 2 reasons making it insufficiently secure and what kind of attack(s) an attacker can use against this approach. Also explain how these credentials should ideally be stored.

d) Consider the following piece of Java code which generates a random reset token whenever someone requests a password reset for a particular user account by specifying their user id:

```
public static String gen_random_token(long user_id, long token_length)
{
     String alphabet = "abcdefgh1234567890";
     String token = "";
     Random r = new Random(user_id * token_length);
     for (long i = 0; i < token_length; i++)
     {
          token += alphabet.charAt(r.nextInt(alphabet.length()));
     }
     return token;
}
```

Some clarification on the functions:

- `Random(x)`: Initiates random number generator with seed x (full name: java.util.Random)
- `r.nextInt(x)`: Generates the next integer in random number sequence, bounded by value x.
- `str.charAt(x)`: Takes character at position x from string str.

The above function is always called as *gen_random_token(user_id, 32)* where user_id is specified by the person requesting the reset. The resulting token is sent via a secure channel to the actual user corresponding to the userid.

Is the above function secure? If not, mention at least two reasons why not.

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 country, which contains one column 'xml' containing an element "country" with an attribute "runtime" with the runtime and an attribute "movie" with the title of the movie, and as contents the country.

```
SELECT m.name, r.runtime, r.country
FROM runtime AS r, movie AS m
WHERE r.mid = m.mid
```

An example result looks like:

xml
<country movie="Monty Python and the Holy Grail" runtime="89">UK</country>
<country movie="Monty Python and the Holy Grail" runtime="91">USA</country>
<country movie="Twelve Monkeys" runtime="130">Australia</country>
…

b) [SQL/XML] Give an SQL-query that uses the SQL/XML standard such that it produces a result as XML containing all actors with the roles they played in which movies. The result should have one row per actor, which contains one column 'xml' containing an element "actor" with an attribute "name" with the actor's name. The contents should be a list of movie elements with an attribute "role" with the role the actor played and as contents the name of the movie. An example result looks like:

xml
<actor name="Julian Arahanga"> <movie role="Apoc">Matrix, The</movie></actor>
<actor name="Brion James"> <movie role="Leon">Blade Runner</movie> <movie role="Joel Levison">Player, The</movie></actor>
<actor name="Patricia Hitchcock"> <movie role="Caroline">Psycho</movie> <movie role="Barbara Morton">Strangers on a Train</movie></actor>
…

c) [JSON construction] Given the SQL-query below. Describe as exactly as possible what the result looks like. Note the similarity with Question 1a. Describe the result analogously as the given example result for that question, i.e., as a table with a header row and three example rows.

```
SELECT jsonb_build_object('name', m.name, 'runtime', r.runtime,
'country', r.country) as json
FROM runtime AS r, movie AS m
WHERE r.mid = m.mid
```

d) [JSON indexing] Suppose the JSON objects of Question 1c are stored in attribute “json” of table “moviejson”. Given the CREATE INDEX statement and the SQL-query below. Does this index speed up the query? Explain your answer, i.e., describe the conditions and explain whether or not the query complies with them.

```
CREATE INDEX myidx ON moviejson USING gin (json)

SELECT json ->> 'name'
FROM moviejson
WHERE (json ->> 'runtime')::integer = 89
```

e) [JSON querying] The query of Question 1d uses the operator ‘->>’. There is also an operator ‘->’. What is the difference between the two and explain why the query needs to use ‘->>’ and not ‘->’.

f) [JSON querying] Rewrite the WHERE-clause of the query of Question 1d above so that it uses the ‘@>’ operator for the condition that the runtime should be 89.

g) [XML querying] A table ‘x’ contains an attribute ‘xml’ which is of type ‘xml’ and contains actor elements as in the illustration of question 1b.

   i. Write a SQL-query containing an XPath that produces all such ‘actor’ elements for actors who played in the movie “Psycho”.

   ii. Write a SQL-query containing an XPath that produces all actor names for actors who played in more than 1 movie.

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 name="Patricia Hitchcock">
   <movie role="Caroline">Psycho</movie>
   <movie role="Barbara Morton">Strangers on a Train</movie>
</actor>
```

b) Translate the XPath `//movie[@role='Caroline']/ancestor::actor` 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 actor-element, but obviously the query should also work on any XML- fragment that is produced by the query of 1b).

c) [Dewey numbering] Given an 'edge' table based on Dewey numbering. Do you need an attribute 'parent' in the 'edge' table to store the parent-child relationships between the nodes in the tree? 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