

201300180 Data & Information – Test 4 (1.5 hours)

19 June 2015, 13:45 – 15:15

Please note:

- ***Please answer question 1 on a different sheet of paper than questions 2 and 3***
(Not on the back side of the previous question, the questions will be distributed to different persons for grading).
- You can give your answers in Dutch or English.
- Reference materials are given in the appendices, therefore you are not allowed to bring any study materials to the test
- **WAP-students need to answer all questions.**

Grade = #points/10

Question 1 (Security) (40 points)

Consider the PHP code fragment below:

```
function display_error($message)
{
    echo $message;
    return;
}
$s = $_GET['search'];
$result = apply_search_term($s);
if(!$result)
{
    $message = "Search term $s yielded no results!";
    display_error($message);
    exit();
}
```

- a) *[XSS] Why is this code insecure? Describe how an attacker could exploit the above code and what the consequences could be.*

Consider the following cookies with session tokens:

Cookie: SESSION=dXNlcmkPTE7

Cookie: SESSION=dXNlcmkPTI7

Where dXNlcmkPTE7 is base64("userid=1;") and dXNlcmkPTI7 is base64("userid=2;")

- b) *[Session management] Are the session tokens generated by the web application secure? If not, argue why not and describe how an attacker might exploit this fact.*
- c) *[Session management] Name the two desirable properties of session tokens.*

Consider the PHP code fragment below:

```
$i = $_GET['id'];
$query = "SELECT id, name, price FROM item WHERE id = $i";
$result = mysql_query($query);
```

- d) [SQL injection] Is this code secure? If not, give an input (assuming all id fields in the item table are >= 0) that would allow an attacker to make the query return the tuple (1, 2, 3).
- e) [SQL injection] What technique would you use to secure the above code (you don't have to give pseudocode, just mention your approach)?

Consider the following fragment of an exported SQL database dump:

```
INSERT INTO `user` (`user_id`, `username`, `password`, `email`)
VALUES
(1, 'admin', 'password1234', 'admin@server'),
(2, 'demo', 'demo', 'demo@server'),
(3, 'nobody', 'changeme', 'nobody@server'),
```

- f) [Hashing] What is insecure about the storage of this sensitive information? Describe briefly how this should be stored instead

Question 2 (From and to XML) (30 points)

We base ourselves again on the movie database (see Appendix 1 for the schema of the movie database). An informal syntax of SQL (including the SQL/XML functions) is given in appendix 2.

- a) [SQL/XML] Give an SQL-query that produces all directors with their name and pid in the form of a result with one row per director ordered by name, which contains one column 'xml' which contains an element "director" with an attribute "id" and a child text node with the name. An example result looks like:

| xml |
|---|
| <director id="6669">Akira Kurosawa</director> |
| <director id="9243">Alan J. Pakula</director> |
| <director id="6719">Alfred Hitchcock</director> |
| ... |

- b) [SQL/XML] Adapt the above query, such that the name of the director is an attribute, and the movies (s)he directs are child elements with tag name 'movie'. An example result looks like:

| xml |
|--|
| <director id="6669" name="Akira Kurosawa"> <movie>Ran</movie><movie>Rashomon</movie>... </director> |
| <director id="9243" name="Alan J. Pakula"> <movie>All the President's Men</movie> </director> |
| <director id="6719" name="Alfred Hitchcock"> <movie>Notorious</movie><movie>Birds, The</movie><movie>North by Northwest</movie>...< </director> |
| ... |

- c) [XML-to-relations; schema-based] Give the simplification according to the shared inlining approach, of the DTD below.

```

<!DOCTYPE movies [
  <!ELEMENT movies (movie)*>
  <!ELEMENT movie (name, year, (director|actor)*)>
  <!ELEMENT actor (name, role?)>
  <!ELEMENT director (name)>
  ...
]>

```

All elements not mentioned can be assumed to contain only #PCDATA.

- d) [XML-to-relations; schema-based] *The element "actor" will get a separate table. Give its definition in the form of a CREATE TABLE statement which defines all attributes with their types including any declarations such as PRIMARY KEY, REFERENCES, and NOT NULL (see Appendix 2 for an informal syntax of SQL CREATE TABLE statements).*
- e) [XML-to-relations; schema-less] *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.*

```

<movies>
  <movie>
    <name>Ran</name>
    <year>1985</year>
    <director>Akira Kurosawa</director>
  </movie>
</movies>

```

- f) [XML-to-relations; schema-less] *Give the Dewey number of the 'director' element in the above XML-document of question 2(e).*
- g) [XML-to-relations; schema-less] *Translate the XPath query '//movie[./year=1985]' 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 movie-element, but obviously the query should also work on any XML-document that is valid according to the DTD of question 2(c).*

Question 3 (Information Retrieval & Full-Text search) (30 points)

- a) [Full-Text search] *Given the XQuery below, does it produce a result for a voyage element with in its 'particulars' element the text "The Cape, you know, of Good Hope"? Explain your answer by mentioning what matters and what does not matter for text matching.*

```

for $v score $s in //voyage[. contains text "cape"
                        not in "cape of good hope"]
order by $s descending
return <result score="{ $s }">{ $v }</result>

```

- b) [IR] *Why are both very frequent terms and very infrequent terms not good search terms?*

- c) [IR; tf.idf] *Given the two plot_outlines below, calculate the tf.idf scores for the query ‘accused of’. The formula’s for ranking and idf are given below the table. If you don’t have a calculator for computing log, just invent a number between 0 and 1 (explicitly mention that you did this!)*

| mid | name | year | plot_outline | rating |
|-----|-----------------|------|--|--------|
| 588 | Green Mile, The | 1999 | Drama about the lives of guards on death row leading up to the execution of a wrongly accused man. | 8.4 |
| 714 | Fugitive, The | 1993 | Dr. Richard Kimble, unjustly accused of killing his wife, must find the real one-armed killer while avoiding Marshal Sam Gerard. | 7.8 |

$$\text{idf}(t) = \log (N/\text{df})$$

$$\text{Rank}(d,q) = \sum_{t \in q} \text{tf}(d,t) \text{idf}(t)$$

- d) [IR; language models] *Calculate $P(\text{accused} | D)$ where D is the plot_outline of “The Green Mile”.*
- e) [IR; language models] *A 2-gram language model does not make the assumption that, for example, $P(\text{“New York”} | D) = P(\text{“New”} | D) P(\text{“York”} | D)$. Why does this assumption not hold? Use terminology from probability theory and/or statistics in your answer.*

Appendix 1: Databaseschema movie-database

- **Movie:** mid integer PRIMARY KEY, name text, year numeric(4,0), plot_outline tekst, rating numeric(2,1)
- **Person:** pid integer PRIMARY KEY, name tekst
- **Acts:** mid integer, pid integer, role tekst
- **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

Appendix 2: Informal syntax for 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

sqlquery: 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 | '*'

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, ...)

...