201300180 Data & Information – Test 4 (1.5 hours)

17 June 2016, 13:45 – 15:15

Please note:

- ***Please answer questions 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 person 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 following fragment of PHP code:

```
echo $header . "<img src = ' " . $_GET['image'] . " '>" . $footer;
```

which is called as part of a script accessed via a URL that looks as follows:

```
http://website.com/script.php?image=picture.jpeg
```

a) *[XSS] Is this code insecure and if so, why? Describe (in a few sentences, an example is not required) how an attacker could exploit the above code to hijack a victim's session.*

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

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

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

b) *[SQL injection] 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.*

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

```
store_db(hash($_POST['password']));
```

c) *[Credential storage] Where hash is a cryptographically secure hashing function such as SHA256. Explain why this is insufficiently secure to reduce the impact of for example a database breach, make sure to mention what kind of attack an attacker can use against this approach. Also explain how these credentials should ideally be stored?*

Consider the following piece of Java code which generates a session token stored in a user's cookie upon successful login:

```
public static String gen_session_token(long user_id, long timestamp)
{
      String alphabet = "abcdef1234567890";
      String token = "";
      Random r = new Random(user_id + timestamp);
      for (int i = 0; i < 32; i++)
      {
            token += alphabet.charAt(r.nextInt(alphabet.length()));
      }
      return token;
}
```

Some clarification on the functions:

- `Random(x)`: Initiates random number generator with seed x.
- `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 `get_session _token(user_id, time())` where `user_id` is the user ID of the person logging in and `time()` will produce a UNIX-style timestamp of the current time on the server (at the time of login) accurate in the order of seconds. The token is stored in a cookie like this:

```
Cookie: SESSID=098f6bcd4621d373cade4e832627b4f6
```

You may assume sessions last at most 24 hours and are automatically removed from the database after exceeding that age for security and performance reasons.

d) *[Random number generation] Is the above function secure? If not, mention at least two reasons why not.*

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.

The SQL-query below determines for each language, how many movies there are in that language.

```
SELECT l.language, COUNT(*)
FROM language AS l, movie AS m
WHERE l.mid = m.mid
GROUP BY l.language
```

a) *[SQL/XML] Given the SQL-query above, adapt it using the SQL/XML standard such that it produces the result as XML. The result should have one row per language, which contains one column 'xml' containing an element "language" with an attribute "name" with the language and an attribute "count" with the number of movies in that language. An example result looks like:*

xml
<language name="English" count="219"/>
<language name="German" count="16"/>
<language name="Spanish" count="2"/>
…

b) *[SQL/XML] Adapt the above query, such that the names and the years of the movies are child elements of 'language' with element name 'movie', attribute 'year' and the title of the movie between the opening and closing tags of the element movie. An example of the result looks like:*

xml
<language name="English" count="219"><movie year="1972">Godfather, The</movie><movie year="1994">Shawshank Redemption, The</movie>…</language>
<language name="German" count="16"><movie year="1942">Casablanca</movie><movie year="1998">Saving Private Ryan</movie>…</language>
<language name="Spanish" count="2"><movie year="1999">Sixth Sense, The</movie><movie year="1999">Todo sobre mi madre</movie></language>
…

c) *[XML-to-relations; schema-based] Give the CREATE TABLE statements that result from applying the shared inlining approach to the DTD below*

```
<!DOCTYPE languages [
<!ELEMENT languages (language)*>
<!ELEMENT language (movie*)>
<!ATTLIST language name CDATA #REQUIRED>
<!ATTLIST language count CDATA #REQUIRED>
<!ELEMENT movie #PCDATA>
<!ATTLIST movie year CDATA #REQUIRED>
]>
```

d) *[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.*

```
<languages>
   <language name="Spanish" count="2">
      <movie year="1999">Sixth Sense, The</movie>
      <movie year="1999">Todo sobre mi madre</movie>
   </language>
</languages>
```

e) *[XML-to-relations; schema-less] Translate the XPath query*
`//movie[@year="1999"]/parent::language`
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 language-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] The XQuery below produces a 'result' element for each text match. Does it produce a match at all given the document of Question 2(d)? Explain your answer by mentioning all relevant aspects that full-text search takes into account that make sure that a match is made (in case of "yes") or blocked (in case of "No").*

```
for $v score $s in //language[. contains text "sixth"]
order by $s descending
return <result score="{$s}">{$v}</result>
```

b) *[IR; tf.idf] Given the two plot_outlines below, calculate the tf.idf scores for the query 'finds only'. Explain your answer by giving the full calculation. The formulas 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
776	Groundhog Day	1993	Phil, a sarcastic weather man, finds himself trapped in the identical February 2, but he's the only person that realizes it.	7.7
586	2001: A Space Odyssey	1968	Mankind finds a mysterious, obviously artificial, artifact buried on the moon and, with the intelligent computer HAL, sets off on a quest.	8.3

$$Idf(t) = \log (N/df)$$

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

c) *[IR; tf.idf] Suppose we would like improve our movie search engine, which works based on the tf.idf approach, such that it ranks movies with higher rating higher. Adapt the formulas above to accomplish this; explain your answer.*

d) *[IR; language models] Calculate P(finds|D) where D is the plot_outline of "2001: A Space Odyssey".*

e) *[IR; language models] Give a ranking for both plot_outlines according to the language modeling approach for the query 'finds only'. Explain your answer by giving the full calculation.*

f) *[IR; language models] Suppose again we'd like to improve our search engine such that it ranks movies with a higher rating higher, but now for the language modeling approach. In which of the probabilities would you incorporate the rating for this purpose? P(D|T), P(T|D), P(D), or P(T)? Explain your answer.*

Appendix 1: 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

Appendix 2: 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, ...)
...