Homework #2 Solution

Question 1.
1)  
   a.
   
<table>
<thead>
<tr>
<th></th>
<th>alicerc</th>
<th>bobrc</th>
<th>cyndyrc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>own, execute</td>
<td>read</td>
<td>-</td>
</tr>
<tr>
<td>Bob</td>
<td>read</td>
<td>own, execute</td>
<td>-</td>
</tr>
<tr>
<td>Cyndy</td>
<td>read</td>
<td>read/write</td>
<td>read, write, own, execute</td>
</tr>
</tbody>
</table>

   b.
   
<table>
<thead>
<tr>
<th></th>
<th>alicerc</th>
<th>bobrc</th>
<th>cyndyrc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>own, execute</td>
<td>read</td>
<td>read</td>
</tr>
<tr>
<td>Bob</td>
<td>-</td>
<td>own, execute</td>
<td>-</td>
</tr>
<tr>
<td>Cyndy</td>
<td>read</td>
<td>read/write</td>
<td>read, write, own, execute</td>
</tr>
</tbody>
</table>

3) It is generally infeasible to apply query-set-overlap because of the need to keep track of query history. The larger the history, the longer the query response time will be.

4)  
   a. command delete_all_rights(p, q, s)
      delete r in a[q, s]
      delete w in a[q, s]
      delete e in a[q, s]
      delete a in a[q, s]
      delete l in a[q, s]
      delete m in a[q, s]
      delete o in a[q, s]
      end

   b. command delete_all_rights(p, q, s)
      if m in a[p, s]
         then
            delete r in a[q, s]
            delete w in a[q, s]
            delete e in a[q, s]
            delete a in a[q, s]
            delete l in a[q, s]
            delete m in a[q, s]
            delete o in a[q, s]
      end
c. command delete_all_rights(p, q, s)
   if m in a[p, s] and not o in a[q, s]
   then
      delete r in a[q, s]
      delete w in a[q, s]
      delete e in a[q, s]
      delete a in a[q, s]
      delete l in a[q, s]
      delete m in a[q, s]
      delete own in a[q, s]
   end

5) a. command copy_all_rights(p, q, s)
   if r in a[p, s] then
      enter r in a[q, s]
   if w in a[p, s] then
      enter w in a[q, s]
   if e in a[p, s] then
      enter e in a[q, s]
   if a in a[p, s] then
      enter a in a[q, s]
   if l in a[p, s] then
      enter l in a[q, s]
   if m in a[p, s] then
      enter m in a[q, s]
   if o in a[p, s] then
      enter o in a[q, s]
   end

b. command copy_all_rights(p, q, s)
   if r in a[p, s] and rc in a[p, s] then
      enter r in a[q, s]
   if w in a[p, s] and wc in a[p, s] then
      enter w in a[q, s]
   if e in a[p, s] and ec in a[p, s] then
      enter e in a[q, s]
   if a in a[p, s] and ac in a[p, s] then
      enter a in a[q, s]
   if l in a[p, s] and lc in a[p, s] then
      enter l in a[q, s]
   if m in a[p, s] and mc in a[p, s] then
      enter m in a[q, s]
   if o in a[p, s] and oc in a[p, s] then
enter o in a[q, s]
end

c. The subject receiving the copy flag along with the right would be able to grant that right to other subjects.

Question 2.

3.9.1) [An old Solution]
The goal is to show that multiple create commands in a sequence of commands that leak a right \( r \) are equivalent and can be replaced by a single create for safety analysis purposes. Also note that in the proof of the theorem, we are interested in the minimum sequence of commands \( c_1, c_2, \ldots, c_k \) needed to leak a right.

A mono-operational command in the Access Control Matrix model can test for right(s) in access matrix entries \( A[s_1, o_1] \) and \( A[s_2, o_2] \) and based on the outcome can perform a primitive command that leaks a right \( r \). If multiple creates are equivalent then the same test can be performed with entries \( A[s_1, o_1] \) and \( A[s_1, o_2] \), where \( A[s_1, o_2] \) is now union of \( A[s_2, o_2] \) and \( A[s_1, o_2] \) \((A[s_2, o_2] \cup A[s_1, o_2])\), and get the same outcome for the test as before.

To justify the statement, consider the nature of the tests in the model. Note that, in ACM model, we only check for presence of right(s) in a matrix entry (we do not test for the absence of right(s)). A test, hence, comprises of a conjunction and/or disjunction of simple primitive tests, each checking for the presence of a right in an access control matrix entry.

Now, consider entries \( A[s_1, o_1] \) and \( A[s_2, o_2] \). Some tests check for presence of a right in \( A[s_1, o_1] \) and others check for a right in \( A[s_2, o_2] \). When we consider the same test for \( A[s_1, o_1] \) and \( A[s_1, o_2] = A[s_1, o_2] \cup A[s_2, o_2] \), all the previous tests on \( A[s_2, o_2] \) will now be carried out on \( A[s_1, o_2] \). The outcome of the primitive tests on \( A[s_1, o_1] \) will remain the same, since \( A[s_1, o_1] \) hasn’t changed. \( A[s_1, o_2] \) is now a superset of \( A[s_2, o_2] \) since we have added all the rights in \( A[s_2, o_2] \) to \( A[s_1, o_2] \) that were not there already (the enter command doesn’t fail if a right added already exists). Hence the test on \( A[s_1, o_2] \) will also give the same output as before, since we are always checking for presence of a right. The overall output of the test will not change, since the output of constituent primitive test hasn’t changed.

To clarify the argument, consider the following example.

Let \( A[s_1, o_1] = \{r\} \), \( A[s_2, o_2] = \{w\} \) and \( A[s_1, o_2] = \{x\} \), where \( r \), \( w \), and \( x \) are generic rights. Let the test check for presence of \( r \) in \( A[s_1, o_1] \) and \( w \) in \( A[s_2, o_2] \). The test succeeds when we consider \( A[s_1, o_1] \) and \( A[s_2, o_2] \).

Now \( A[s_1, o_2] \) is set to \( \{w, x\} \) (by the union operation). The test for \( w \) in \( A[s_1, o_2] \) succeeds since \( A[s_1, o_2] \) is now a superset of \( A[s_2, o_2] \).
Would it be true if we check for absence of rights?

No. This will not work if we allow checking for absence of a right r in A[s2, o2] and the right is not there. However, when we take the union of A[s1, o2] and A[s2, o2] and r is present in A[s1, o2] initially, the test will succeed on the modified A[s1, o2].

To continue with earlier example, if we test for absence of right x in A[s2, o2]. The test will succeed on A[s2, o2] but the test will fail for A[s1, o2] (as A[s1, o2] = {w, x}).

3.9.4) [An old Solution]
**Claim**: If object x and subject y are connected by a tg-path of length 1, they can share rights.

The claim is **false**.

**Proof**:
Assume that there exists a sequence of take-grant rule applications that creates the edge from x to y, i.e., x gets a right over y. The last step in the sequence would be the actual creation of that edge, for which there are two possible cases:

1. x takes (a to y) from another subject or object v.

This scenario implies that x must have created v in a previous step. However, the create rule specifies that only subjects can create new vertices. x being an object cannot create the new vertex v. So x cannot take (a to y).

2. Another subject or object v grants (a to y) to x.

This is sketched in the following picture:
This scenario implies that v must have been created in some previous steps and that v granted a permission g to x. In this case, only y or z can create the new vertex v. Suppose y creates v, then there are no rules that allow v to grant/take permissions over other objects or subjects of the graph. Now suppose z creates v. In that case, z can grant (t to x) to v, by using the grant rule:

However, it is not possible in any way for v to grant a permission g to x using the take-grant rules. Even if v is a subject and in turn creates a new subject v', there is no sequence of rules that would make v' (or possibly some child vertices) grant a permission g to x. So v cannot grant (a to y) to x.

**Hence:** If object x and subject y are connected by a tg-path of length 1, they cannot always share rights.

3.9.5)
Conditions (a), (b), and (c) are same as those in Theorem 3-10. So let’s focus on condition (d).

Note the following:
- all vertices x1, x2, ..., xn are subjects.
- between any pair of vertices there are four possible ways that they can be connected by a t or g edge, which are captured by the take rule, grant rule, lemma 3.1, and lemma 3.2.
  - so any pair of vertices connected by a t or g edge can share the rights.
- If a pair of vertices is connected by a bridge then they can share rights.

Since between any consecutive pair of vertices we have a t-edge, g-edge, or a bridge, it follows that the sharing occurs between any pair of vertices. Hence the result holds.

**Question 3.**
Prove, by using induction, that any right possessed by any vertex in the island can be shared with any other vertex in the island. Here you can refer to the cases related to 1 length tg-path discussed in section 3.3.1.

**Solution (1) of Kim Cingota:**

**Base case:**
This is true according to Lemma 3-1
(any two subjects with a tg-path of length 1 can share rights)

Hypothesis:
Assume that “any right possessed by any vertex in the island can be shared with any other vertex in
the island” is true for k tg-paths.

Prove:
If the rights possessed by any vertex in an island can be shared with any other vertex in the island
when there are k tg-paths, then the hypothesis should still be true when you add one more tg-path
(make an island of k+1 tg-paths).

This holds true based on Lemma 3-1
This is true based on the Take Rule

This is true based on the Grant Rule
This is true based on Lemma 3-2

Because all of the possible tg-paths of length 1 that can be added to the island allow the new vertex to
be share rights with vertices z any y, and since z and y can share their rights with any other vertex in the
island, it follows that x can also share its rights with any other vertex in the island.