CS 161 – Homework#2

Student: Ninh DO
SID: 25949105

Problem 1 – Web Security Warm-Up

(a) My friend is NOT right. The SOP and DOM protect different websites from each other. Since the site's owner does not have origin of the other sites, he cannot use scripts in his site to access contents of other sites, like cookies. Beside, DOM encapsulates data in objects, which hides data, protecting it from external manipulation.

(b) A separate website is better because of SOP. If user sites share the same origin with google, they can launch scripts to access/alter the google sites other than their own. They also can name their site overlapping google's existing site, say, google has site google.com/sites/abc/def, and the attacker can name abc/def or launching some scripts to take control of the site.

(c) The tinyurl redirects to:
https://www.bramspam.com/share?text=Caleb%20is%20the%20best%20brother%20ever&theme=howboudah

(d) We have a number of ways:
- Control information, allow what to be posted.
- Do not allow the users to inject command into the url.
- Use cookies and token: the users log in their account, they have cookies and token, and only allow posts from browsers having those cookies and token. The url sent around does not, so it should not be allowed to post.

Problem 2 – XSS: The Game

Level 1: Vulnerability: the site does not check/escape text input. Attack: inject javascript into textbox.

```
<script> alert("I'm in :P") </script>
```

Level 2: Vulnerability: the post contain HTML, i.e. template does not escape the contents of status message. Attack: sneak in code to execute alert()

```
<img src="image.gif" onerror=alert("My portrait")>
```

Level 3: Vulnerability: look at the source code, it executes the query from URL bar `html += "<img src='/static/level3/cloud" + num + ".jpg' />";` . Attack: insert code to variable num
https://xss-game.appspot.com/level3/frame#' onerror="alert(“It’s me again!)"

Level 4: Vulnerability: look at the source code, we see the function startTimer handles all input, and the function call `onload="startTimer('{{ timer }}');"` is vulnerable if we insert some quotes and another function to be executed, like 300') || alert ('Done.
Attack:
https://xss-game.appspot.com/level4/frame?timer=300') || alert ('Done

Problem 3 – SQL Injection

(a) username = “devil’; DROP Customers; --“
i.e. the string devil’; DROP TABLE Customers; --

(b) username = “Admin’; --“
i.e. the string Admin’; --

(c) username = “aaa\’ OR username =\”Admin\”; --“
i.e. the string aaa\’ OR username =\”Admin\”; --

(d)
```
ResultSet checkPassword(Connection conn, String arg_usr, String
arg_pw)throws SQLException {
     String query = "SELECT user_id FROM Customers WHERE
username = ? AND password = SHA256(’?’);";
     PreparedStatement p = conn.prepareStatement(query);
     p.setString(1, arg_usr);
     p.setString(2, arg_pw);
     return p.executeQuery(query);
}
```

The query does not take input directly from the user but the query is precompiled and stored in the PreparedStatement object, thus it is independent from the user.
Since the query is independent, the user cannot make use of it to launch a SQL injection attacks.

(e)
```
ResultSet searchTransactions(Connection conn, BigDecimal amt,
String arg_oC, String arg_oD) throws Exception {
     String q = "SELECT * FROM Transactions WHERE amount >= ?
ORDER BY ? ?;";
     PreparedStatement p = conn.prepareStatement(q);
```

```
        p.setBigDecimal(1, amt);
        p.setString(2, arg_oC);
        p.setString(3, arg_oD);
        return p.executeQuery(q);
}
```

The query involves different data types and operation. It is also a long query so not easy to write a Java prepared statements.

Problem 4 – Reasoning About Memory Safety

(a) Precondition: require hex != NULL

(b) Invariant: tmp != NULL && 0 <= j && j < 8
Explain: tmp is indexed, so it cannot be NULL, tmp has size 9, so the index j must be non-negative and less than 8 because the index j is increase by 1.

(c) Invariant: digit must be char, i.e. 0<= digit && digit < 256
Explain: because digit is assigned to tmp array of type char.

(d) Invariant: tmp != NULL && 1 <=j && j < 8
Explain: tmp is indexed, so it cannot be NULL, j is index of hex so j is non-negative, but j is subtracted 1 so j must be positive, tmp has size(tmp) element so j < size(tmp) = 9, combined with invariant (b), we have j < 8, so j must be less than 8.

(e) Invariant: hex != NULL && 0 <= k && k < size(hex)-1
Explain: hex is indexed, so it cannot be NULL, k is index of hex so k is non-negative, hex has size(hex) element so k < size(hex), but k is added by 1 so k < size(hex)-1

(f) Invariant: hex != NULL && 0 <= k && k < size(hex) - 1
Explain: hex is indexed, so it cannot be NULL, k is index of hex so k is non-negative, hex has size(hex) element so k < size(hex), combined with invariant (e), we have k < size(hex)-1

Feedback

Please be better in timing. Proj1 and HW2 overlap much in time creating difficulties for students.