CS 161 Computer Security
Spring 2010 Paxson/Wagner

HW 2

Due Thursday February 25, 5:00pm

Instructions: Submit your solution by Thursday, February 25, 5:00pm *electronically*. Write up your answers in either PDF format or plain text (7-bit ASCII), in a separate file for each question. Your files should be named `q1.txt`, `q2.txt`, `q3.txt`, ..., `q6.txt` (for plain text), or `q1.pdf`, ..., `q6.pdf` (for PDF). You may mix-and-match text and PDF if you like. Also, include a file called `collaborators.txt`, containing a list of everyone you collaborated with (see below). Then run `submit hw2`.

You may collaborate with at most 3 other CS161 students on this homework. Collaboration may involve discussing the problems with other students, but must *not* involve sharing solutions or having access to anyone else's solutions *at any time*. You must develop and write up your solutions *entirely on your own*. You must never read or copy the solutions of other students, and you must not share your own solutions with other students.

At the beginning of each file, include your name, class account (e.g., cs161-xy), your TA's name, and the discussion section time where you want to pick up your graded homework. (You don't need to include this for `q6` or `collaborators.txt`.)

1. **(20 pts.) Using prepared statements**
"Prepared statements" are a way of issuing SQL queries by initially compiling a query down to a *query plan*. (A query plan is a template that indicates which tables and indexes to use to execute the query.) When you need to run the query, your program fills in the "holes" in the query with data. Prepared statements can help defend against SQL injection attacks because the data used to fill in the query plan gets treated as a single SQL element, rather than possibly including SQL operators or syntax.

   For a tutorial on using prepared statements in Java see `http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html`. You can find more information about the `java.sql` API at `http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html`. For this problem, you will probably find most useful to examine the API documentation for `Connection`, `Statement`, `PreparedStatement`, and `ResultSet`.

   In this question, imagine you are the security guru at a web startup. You've been called in to fix some insecure code that was written by another programmer who wasn't very knowledgeable about security. (Probably graduated from Stanford.)

   Parts (a)–(c) show three snippets of *insecure* code for querying a SQL database. For each snippet, your job is to rewrite the code to use prepared statements instead. Your code *must* use prepared statements. Your code must fetch the same data from the database when the system is not under attack, but must be secure against SQL injection attacks. If prepared statements cannot be used for the given type of query, explain why.

   (a) This snippet is intended to retrieve the profile of a particular user of your web site. External input to this snippet provides the user id (`uid`) that identifies the user whose profile should be retrieved. The *insecure* code you've got to fix is

```
ResultSet getProfile(Connection conn, int uid) throws SQLException {
    String query = "SELECT profile FROM Users WHERE uid = " + uid + ";";
    Statement s = conn.createStatement();
    return s.executeQuery(query);
}
```

This code was intended to produce SQL queries such as:

```
SELECT profile FROM Users WHERE uid = 555;
SELECT profile FROM Users WHERE uid = 80;
```

(where `uid` was specified by the external input as `555` in the first example and as `80` in the second). Rewrite the body of the `getProfile()` method using prepared statements, as specified above, or explain why prepared statements cannot be used for this purpose.

(b) To jump on the social networking bandwagon, your web site includes a feature that lets a user list the names of all "friends" who are also relatives of the user. The insecure code that queries the database is

```
ResultSet getFriends(Connection conn, int uid) throws SQLException {
    String query = "SELECT name FROM Friends WHERE uid = " +
      uid + " AND friend_uid IN " +
      "(SELECT relative_uid FROM Relatives WHERE uid = " +
      uid + ");";
    Statement s = conn.createStatement();
    return s.executeQuery(query);
}
```

This was intended to generate queries such as the following:

```
SELECT name FROM Friends WHERE uid = 555 AND friend_uid IN
  (SELECT relative_uid FROM Relatives WHERE uid = 555);
SELECT name FROM Friends WHERE uid = 80  AND friend_uid IN
  (SELECT relative_uid FROM Relatives WHERE uid = 80);
```

Rewrite this insecure code, following the instructions above, or explain why that's not possible.

(c) Your site includes a forum. A user of the forum can search for a post by title and/or by author. There can be an arbitrary number of "OR"s for either of these fields. Thus, the code needs to generate SQL queries, such as the following:

```
SELECT * FROM posts WHERE (author='kitkat');
SELECT * FROM posts WHERE (title='foo') AND (author='alf');
SELECT * FROM posts WHERE (title='bar' OR title='snickers') AND
    (author='ziggy' OR author='yoda' OR author='xavier');
```

The *insecure* code that generates these SQL queries is

```
private String join(String[] a, String field) {
    if (a.length == 0)
        return "";
    String s = "(";
    for (int i=0; i<a.length; i++)
        s += (i>0 ? " OR " : "") + field + "='" + a[i] + "'";
    return s + ")";
}
ResultSet getPost(Connection conn, String[] authors, String[] titles)
        throws SQLException {
    String q = "SELECT * FROM posts WHERE ";
    q += join(authors, "author");
    if (authors.length > 0 && titles.length > 0)
        q += " AND ";
    q += join(titles, "title");
    q += ";";
    return conn.createStatement().executeQuery(q);
}
```

Rewrite this code, as specified above, or explain why that's not possible.

(d) During a code audit, you discover the following method, which checks the password a user has entered against the password stored for the user in the database. Give an example of a user name and password combination that an attacker could use to exploit this method in order to authenticate without knowing a user's password.

```
public static boolean
checkPassword(Connection conn, String userName, String enteredPassword)
        throws SQLException {
    String query = "SELECT * FROM Users WHERE userName = '" + userName +
        "' AND password = '" + enteredPassword + "';";
    Statement s = conn.createStatement();
    ResultSet rs = s.executeQuery(query);
    if (rs.isAfterLast()) // if no results in result set
        return false;
    return true;
}
```

(e) Rewrite the vulnerable method in part (d) to use prepared statements to eliminate the vulnerability.

2. (20 pts.) XSS

(a) What is the basic difference between a reflected XSS attack and a stored (or persistent) XSS attack?

(b) Consider a web page located at `http://vulnerable.com/test.html` (not a real page) consisting of the following HTML:

```
<html>
<div id="theDiv">Hi </div>
<script type="text/javascript">
var pos=document.URL.indexOf("name=");
var name = "John Doe";
if (pos !== -1)
    name = unescape(document.URL.substring(pos+5,document.URL.length));
document.getElementById("theDiv").innerHTML += name;
</script>
</html>
```

Give an example of a malicious URL an attacker could send out to mount a XSS attack against a user of this web site. Your script can simply execute `alert(1)` if the user clicks on the URL.

Note: It may be helpful to save the HTML to a file and point your browser to the file in order to test your exploit. For example, if the file is saved in `/Users/me/test.html` then you should point your browser to `file:///Users/me/test.html`. You can then try URLs like `file:///Users/me/test.html?name=foo` to invoke the script with the variable `name` bound to the string `"foo"`.

(c) Modify your attack URL so that the injected script steals the cookies from the `vulnerable.com` web site and sends them to an attacker who owns `badguy.com`. The following hints might be helpful:

- JavaScript code can access the cookies associated with a site via the variable `document.cookie`. You might try, for example, going to any site that you think might have set one or more cookies, and typing the following in your URL bar: `javascript:alert(document.cookie);`. (If you're curious about accessing cookies in JavaScript, check out `http://www.quirksmode.org/js/cookies.html`, but you shouldn't need to do so to answer the problem.)
- Browsers provide something called the DOM (Document Object Model) so that JavaScript code can reference the structure of the HTML page and dynamically change the content of the page. For example, the page in this problem contains a `div` element (which defines a division or section of a page) that has an `id` attribute set to the value `theDiv`. You can use `document.getElementById("theDiv")` to get an object representing the `div` element. You can then set its `innerHTML` property to change the contents of the `div`. To help understand how this works, you might try typing the following in your browser's URL bar after loading the page in your browser: `javascript:document.getElementById("theDiv").innerHTML = "foobar"`.

3. (20 pts.) Netalyzr

In lecture (either Friday 2/19 or Monday 2/22) we briefly discussed the "Netalyzr" tool, a Java applet you can run from any browser to measure the properties of the Internet connectivity available to the browser.

Using whatever browser you wish, load the applet from `http://netalyzr.icsi.berkeley.edu` and run its analysis. (You might need to confirm to the browser to trust the applet. Note, it will take several minutes to run.) Inspect the output and briefly summarize:

(a) What browser did you use (for example, "my laptop over AirBears" or "my desktop at home")?

(b) What does Netalyzr indicate about which TCP and UDP ports, if any, are blocked or in some way controlled by the network?

(c) What was your brower's public address? Does Netalyzr indicate that you ran the browser behind a NAT? If so, what was its private address?

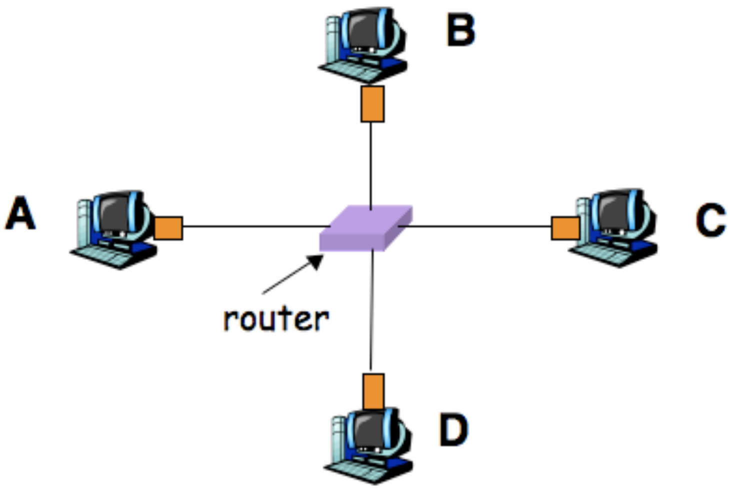
(d) What does Netalyzr indicate about whether your browser's DNS resolver randomizes its source ports? Is the resolver vulnerable to the Kaminsky attack discussed in lecture?

(e) Near the top of the results page there's a link labeled "Permalink". Follow this link and record the resulting URL. This allows you (and us) to later fetch a copy of the results of the measurement run.

4. (20 pts.) Manipulating the network into letting you eavesdrop

This problem concerns how an attacker can use specially crafted packets to manipulate a router into allowing the attacker see traffic that normally the attacker couldn't see. The version of the problem presented here is simplified compared to how the attack actually works in practice, but the principle it illustrates is the same. (The actual attack is against "switched Ethernet" networks.)

The following figure shows a router that has four nodes – A, B, C, and D – directly connected to it:



The router functions as follows. When it receives a packet from one of the nodes, the router by default will *broadcast* the packet to all of the other nodes. However, any time a node *sends* traffic to the router, the router *remembers* the link that connects to that node, so in the future for any packet sent *to* the node, the router can send the packet directly, rather than broadcasting it.

For example, if A sends a packet destined for B, then (1) the router broadcasts the packet across the links to each of B, C, and D, and (2) the router learns the location of node A (*not* B!) since the router observes from where it received the packet, and the packet has a source address of A.

Thus, if A sends another packet, the same process repeats (the router broadcasts the packet to each of B, C, and D); but if B sends a packet in reply, the router forwards the packet directly to A (and does *not* broadcast it to C and D) because it previously learned the link that connects it to A. In addition, due to B sending this packet, at this point the router also learns the path to B, so any further packets sent to B (whether from A, or from C or D) get sent directly to B and are not broadcast.

The router uses a *forwarding table* to remember the links associated with different nodes to which it is attached. After A sends a packet to B, the forwarding table has an entry for A. After B replies to A, the forwarding table has entries for both A and B.

Thus, this type of network resists eavesdropping: other than for packets sent at the start of communication, nodes not involved in the communication do not receive copies of packets.

However, the forwarding table that the router uses has a limited capacity. If the router sees a packet sent from a new node X and its forwarding table is full, the router will eject from the table the entry least recently used and assign that slot for the new entry for X.

Note that the router has no way to verify which nodes might be connected to it over a given link, nor how many of them. As far as the router is concerned, it might receive packets from distinct nodes A, A', A'', etc., all arriving via the link that connects the router with A.

Assume that (1) node *C* has been compromised, and (2) the router's table can hold 10 entries. In addition, assume that *A* sends 10 packets to *B* every second at a steady rate, and whenever *B* receives two of these packets, after a 50 msec delay it sends a single packet in response (so it sends 5 packets every second, at a steady rate).

(a) If *C* can spoof whatever packets the attacker wishes, how can the attacker manipulate the router into allowing *C* to see *all* of the communication between *A* and *B*?

(b) If the attacker at *C* instead wishes to disrupt the communication between *A* and *B*, how can they do so just by manipulating the router? (That is, without resorting to techniques like injecting a TCP RST packet.)

5. (20 pts.) Home router

Econorouter ships a wireless DSL router to customers. It has an administrative interface that lets you change lots of configuration options by accessing its web server (which is open to the world):

URL	**Purpose**
`http://yourrouter/login?u=daw&p=mypass`	to login
`http://yourrouter/set?ssid=SkyNet`	set the name of the wireless network
`http://yourrouter/set?wifichannel=3`	to set the WiFi channel
`http://yourrouter/set?time=11:36AM`	set the date/time
`http://yourrouter/set?dns=1.2.3.4`	set the primary DNS server
`http://yourrouter/set?speed=1.5Mbps`	set the link speed
`http://yourrouter/set?dhcp=on`	enable DHCP
`http://yourrouter/set?logging=on`	to enable logging
`http://yourrouter/set?report=24hr`	set how often the router reports status

You have to log in using the correct username and password for that router before setting any configuration option; logging in sets a session cookie on your browser, and then subsequent requests to the router are allowed to set config options. Unfortunately, the default username and password is `admin`/`password`, and many users do not change the default.

(a) Explain how an attacker anywhere on the Internet can attack Econorouter users who haven't changed their default password, to steal all their subsequent search queries to Google and redirect them to the `HackrzSrch.com` search engine (thus getting the ad revenue for themselves). Your scheme should require only a one-time attack on the router, and should not assume the existence of any implementation bugs in the router's software.

(b) Econorouter hears about this flaw, and they decide to modify their routers to prevent this attack. On the new routers, the web server providing the administrative interface will now respond only to connections from the internal home network (e.g., from machines on its local wireless network or local machines connected via Ethernet to the router), at the IP address 192.168.0.1. The router will not respond to connections coming in over the Internet connection (coming in over DSL/cable) to its administrative interface. By default, the router ships with its wireless connection enabled and configured for open wireless, with no password or access control. Explain how an attacker who drives by the house of someone who has bought one of these new Econorouter's and is using it without changing any default setting, can mount the attacks you found in (a).

(c) Econorouter decides that the new default will be to leave wireless disabled. Imagine that Joe is using their newest router, with all the defaults left intact, and he has several home computers hooked up to his Econorouter. He allows of friend of his to connect her laptop to his home network; unfortunately, it's infected with some malware. Explain how that malware could exploit features in the Econorouter to steal all search engine traffic coming from all of Joe's home computers.

(d) Sam is using Econorouter's newest router, with all the defaults. Sam often visits random third-party websites. Suppose the attacker controls a website (dancingbears.com) that Sam happens to visit. Explain how the attacker can exploit features on Sam's Econorouter to steal all of Sam's subsequent search engine traffic subsequently coming from Sam's computer.

6. (0 pts.) Optional: any feedback?

Optionally, feel free to include feedback. What's the single thing we could to make the class better? Or, what did you find most difficult or confusing from lectures or the rest of class, and what would you like to see explained better?

Your answers will not affect your grade. Feel free to be frank: we appreciate all feedback, even (especially) critical feedback.