# Homework 5 Solutions

CS161 Computer Security, Fall 2008
Assigned : 11/26/08
Due : 12/3/08

For your solutions you should submit a hard copy; either hand written pages stapled together or a print out of a typeset document[1].

1. *SQL injection* **[7 points]**

   SQL's prepared statements add the "?" syntax to the language: `select * from foo where bar=?`. `''?''` can then be replaced with a string using a separate function `setString ()`. This is more secure than building up queries by concatenating strings, because `setString()` understands enough of the SQL language to ensure that its arguments are properly interpreted by the database server. For example, if the " bar" column of a database contains strings, then `setString` ensures that its parameter is a string, and the server interprets it as raw string data, instead of interpreting it as a SQL language construct.

   *NOTE : For details of SQL prepared statements in Java, you should read :*
   `http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html`.

   Suppose that a credit card web site uses a form like the one below that asks the user to select a month by number. Assume that the form posted from the user is used to generate an SQL query using a prepared statement and string concatenation.

```
String mname = request.getParameter("month");
String uid = session.getCurrentUserId());
PreparedStatement pstmt = conn.prepareStatement
   ("SELECT username, startdate, transaction, amount
     FROM transaction_history
     WHERE user=" + uid + " AND month=" + mname);
pstmt.execute(); pstmt.close();
```

   (a) **[3 points]** Explain very briefly why this site is vulnerable to SQL injection, even though it attempts to use Java Preparestatements.

   **Solution.** The program still uses string concatenation, rather than bind arguments ("?"), which means that user supplied data (mname) can alter the query even before the prepared statement processes it.

   (b) **[4 points]** Rewrite the above code using Preparestatement placeholder (or bind arguments) parameters (using '?') to eliminate all SQL injection bugs. Explain what will your fixed code do when the attacker attempts the attack you outlined in part (a)?

   **Solution.**

[1] LaTeX is the most suitable tool for typesetting mathematical documents, but other use of other editors are perfectly acceptable

```
    String mname = request.getParameter("month");
    String uid = session.getCurrentUserId());
    PreparedStatement pstmt = conn.prepareStatement
       ("SELECT username, startdate, transaction, amount
         FROM transaction_history
         WHERE user=? AND month=?");
pstmt.setString(1, uid);
pstmt.setString(2, mname);
    pstmt.execute(); pstmt.close();
```

2. *DNS Cache Poisoning* **[5 points]**

   Consider a bank web site, hosted at `https://stockbank.com/`, that uses HTTPS (which is HTTP over SSL/TLS) for all communication. When viewed in a web browser, suppose the bank web page fetches and executes a JavaScript from `https://stockbank.com/scripts/` after 2 seconds of loading the web page.

   Recall that whenever a browser connects to a server using HTTPS, the server sends a certificate to the browser, and the browser checks to make sure that the certificate was issued for the domain the browser requested. Suppose your browser has a bug in the way it treats HTTPS connections: if there is one HTTPS connection open to a server, then while this session is still open, all new connections via HTTPS to the same site are assumed to be part of the existing SSL session and thus do not require a certificate check.

   Explain how a remote attacker could exploit this bug using DNS cache poisoning, to read the cookie of the banking web page when a victim user visits the bank web page. For this question, you may assume that the victim's browser does not cache the DNS bindings (IP addresses for a resolved host name), i.e, it performs a separate DNS lookup for each HTTP request.

   **Solution.** When the user visits the web page, the user connects normally to the bank web site. When the browser performs a second DNS lookup for fetching the scripts, the attacker launches a DNS cache poisoning attack, and resolves the name lookup request to its own IP address. From this point onwards, the browser treats this as another connection the bank website, simply sets up another SSL connection without verifying the certificate (which allows the attacker to present a self-signed certificate), and proceeds to execute scripts served by the attacker's IP address in Stockbank's context. It is easy for the script to read Stockbank's cookies and send them to the attacker's IP address, as it is permitted by the browser according to the same-origin policy.

3. *More DNS attacks* **[8 points]**

   Java web applets can be run off the web using the Java virtual machine (the Java interpreter) in the browser. The Java Virtual Machine (JVM) in the browser enforces the same origin policy just like the rest of the browser. Suppose the browser's JVM has the following implementation of the same origin policy: it allows a Java web applet, obtained from a site A, to communicate over network sockets with *all* those IP addresses that belong to domain A, and with *only* those that belong to site A. For instance, the JVM prevents applets from site A to setup a socket connection to IP addresses belonging to other host names. As expected, the JVM relies on the DNS resolution protocol to determine which IP addresses belong to which host name.

   Recall that in the DNS protocol, an DNS authoritative server can respond with multiple IP addresses for a queried host name. So, for instance, if a users visits the web page of `www.foo.com` containing

a Java applet, and if the DNS server binds `www.foo.com` to two IP addresses, say X and Y, then the JVM would allow the applet to read data from and send data to X and Y unrestricted.

(a) **[4 points]** Suppose Alice's company, FooBar Corp., hosts important confidential documents on an internal web server at `foobar.com/internal/`, allowing access to it for only its employees. FooBar's firewall only allows access to the internal web server from within the company's local intranet; direct connections to this web server from outside the company network are blocked. Show how an attacker (employee of a different company), owning a registered web domain `attacker.com`, can steal FooBar's confidential documents if he can entice Alice to visit `attacker.com` from her office machine. Your attack must exploit the above mentioned weakness in the DNS protocol and the way Alice's JVM handles multiple IP bindings for a hostname.

**Solution.** Alice visits `attacker.com`, which results in a DNS lookup to the authoritative server for `attacker.com` which is under attacker control. The server returns two IP addresses as the response, one is the attacker's server address and the other is the FooBar's internal web server address. The attacker also includes a Java applet, which Alice runs when viewing the web page. The applet makes connections, via Alice's browser, to the IP address of the internal web server, and forwards the read data to the attacker's IP addresses. It is relatively straight forward to have the applet crawl all pages hosted on the internal server this way. This is permitted by the JVM, as it allows applets to communicate with all IP addresses that it binds to `attacker.com`.

(b) **[4 points]** Recall that DNS-SEC is a data integrity and origin authentication protocol for DNS records. For the purpose of answering this question, assume that the functions of DNS-SEC are implemented by providing a signed certificate that allow a DNS module to verify that a DNS record comes from a DNS server that is registered as the authoritative DNS server for that domain. So for instance, if you visit the CS dept. homepage at `cs.berkeley.edu`, your machine will receive cryptographically signed messages that states that the IP address for `cs.berkeley.edu` is 128.32.139.48, and that the DNS server which provides this IP address is authorized to provide them for `berkeley.edu`

Suppose DNS-SEC was ubiquitously deployed on the Internet and FooBar's internal network. Would your attack in part (a) still work? If not, explain how DNS-SEC prevents the attack. If yes, explain why DNS-SEC does not prevent it and suggest a simple fix to prevent the attack.

**Solution.** . DNS-SEC does not prevent the attack. In the attack, the attacker is a valid authorative server for `attacker.com`. The attacker did not affect the data integrity of any request, nor did it claim to be the authorative server for another domain. The weakness is that the IP address data provided by the attacker's DNS server is overly trusted to be valid by the JVM.

One fix is to perform a reverse DNS lookup for the IP address,

4. *Firewalls* **[10 points]**

   Suppose your company has the network topology shown in Figure 1 .

   It has two firewalls: one for the DMZ[2] and another for the internal network.

   You want to ensure the following properties :

   (a) Unless otherwise specified, all traffic should be denied.

   (b) The satellite networks, except 12.0.0.0, should be able to communicate with any DMZ host over port HTTP (port 80).

[2] DMZ is an abbreviated term for a demarcation zone or perimeter network, i.e, is a physical or logical subnetwork that contains and exposes an organization's external services to the Internet. The purpose of a DMZ is to add an additional layer of security to an organization's local network; an external attacker only has access to equipment in the DMZ, rather than the whole of the network.

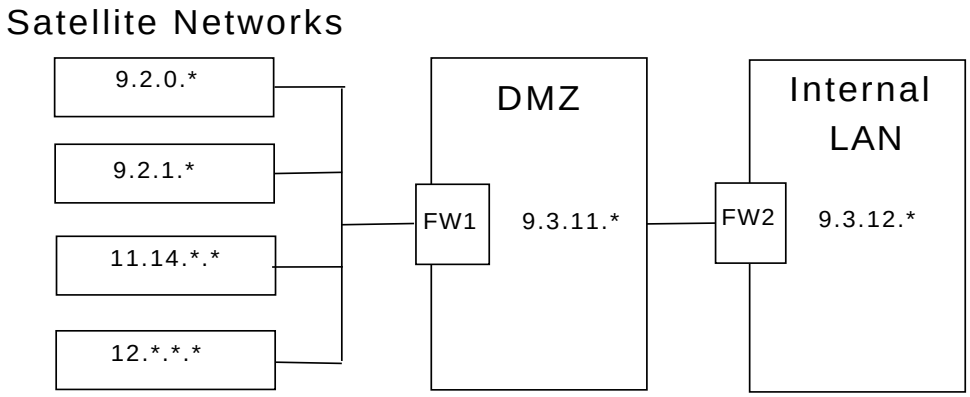


Figure 1: Network Topology of the organization.

(c) Satellite network 9.2.1.0 should be able to communicate with 9.3.11.4 over ssh (port 22)

(d) Nobody outside the DMZ should be able to contact the internal network.

(e) Any host in the internal network should be allowed to talk to the hosts in DMZ over vsftp (port 21)

(f) Any host in the internal network should be able to connect to HTTP servers (port 80) on the Internet

Create stateless firewall policies for the firewalls FW1 and FW2 by filling up tables in figure 2 and figure 3. Create only as many rules as you need in the tables below, keeping number of rules to a minimum. Rules are evaluated by the firewalls in top-to-bottom order, with rules higher up receiving priority over those lower down. For simplicity, you can assume that the firewall does not need to keep track of whether a packet is recieved on an in-bound link or outbound link, i.e, attacks that spoof the internal IP of the organization as the source IP in the packet (discussed in class) are not in scope for this question.

(a) **[6 points]** Show the firewall table entries for FW1 and FW2. The flags field can have values "ACK" and None. The value "ACK", if present in the flags field, implies that the rule will be satisfied iff the packet has the ACK bit set. You may assume that an initial open request packet in TCP does not have the ACK bit set in the header; all other TCP packets do.

**Solution.**

| SRC IP | SRC Port | DEST IP | DEST Port | flags (ACK) | Allow / Drop |
|---|---|---|---|---|---|
| 12.0.0.0 | * | 9.3.11.* | * | * | DROP (b) |
| 9.2.0.* | * | 9.3.11.* | 80 | * | ALLOW (b) |
| 9.2.1.* | * | 9.3.11.* | 80 | * | ALLOW (b) |
| 11.14.*.* | * | 9.3.11.* | 80 | * | ALLOW (b) |
| 12.*.*.* | * | 9.3.11.* | 80 | * | ALLOW (b) |
| 9.2.1.0 | * | 9.3.11.4 | 22 | * | ALLOW (c) |
| 9.3.12.* | * | *.*.*.* | 80 | * | ALLOW (f) |
| *.*.*.* | * | *.*.*.* | * | ACK | ALLOW (b,c,f) |
| *.*.*.* | * | *.*.*.* | * | * | DROP (a) |

Figure 2: FW 1 Table

(b) **[2 points]** Suppose the machine with IP address 9.2.1.1 suddenly starts generating immense traffic to a HTTP web server in the DMZ. How would you change your firewall rule to block the

| SRC IP | SRC Port | DEST IP | DEST Port | flags (ACK) | Allow / Drop |
|---|---|---|---|---|---|
| 9.3.11.* | * | 9.3.12.* | * | * | ALLOW (d) |
| 9.3.12.* | * | 9.3.11.* | 21 | * | ALLOW (e) |
| 9.3.12.* | * | *.*.*.* | 80 | * | ALLOW (f) |
| *.*.*.* | * | *.*.*.* | * | ACK | ALLOW (d,e,f) |
| *.*.*.* | * | *.*.*.* | * | * | DROP (a) |

Figure 3: FW 2 Table

offending machine from attacking the DMZ machine.

**Solution.** Add `9.2.1.1:* -> 9.3.11.*:* * DROP` to the top of FW1. Blocking port 80 specifically also works but the question does not specify which port, only that it is going to a web server in the DMZ.

(c) **[2 points]** Suppose your employees download lots of videos from the new SnailTube website at 1.2.3.4:80. How can you change your firewall rules to stop employees from accessing SnailTube website?

**Solution.** Add `*.*.*.*:* -> 1.2.3.4:80 * DROP` to the top of FW1 or FW2. Blocking traffic from `1.2.3.4:80 -> 9.3.12.*` will also work since it prevents the TCP handshake from completing though it's better to prevent the connection at the start for reasons such as bandwidth cost.