Paxson Spring 2011 — CS 161 Computer Security — Homework 2

Due: Friday March 4, at 5:00pm

Updated 28Feb11: In problem 2, we clarified that the terms "speak," "contact," and "talk" all refer to bidirectional *communication.. Also, the escape sequence in the Java code snippet in problem 4(c) has been fixed; it needed two more backslashes.*

Instructions. Submit your solution by Friday March 4, at 5:00pm, in the drop box labelled CS161 in 283 Soda Hall. Print your name, your class account name (e.g., `cs161-xy`), your TA's name, the discussion section time where you want to pick up your graded homework, and "HW2" prominently on the first page. Staple all pages together. Your solutions must be legible and the solution to each problem must be labelled clearly. You must work on your own on this homework.

Problem 1 ***Reconnaissance Attacks*** **(20 points)**

The IP packet header contains a 16-bit ID field that is used for assembling packet fragments.[1] The protocol standard states that the ID field should differ between different packets sent by a source to a given destination.[2] A common method that hosts use to implement the ID field is to maintain a single counter that the host increments by one for every packet it sends, regardless of to which destination it sends it. The host sets the ID field in each packet it sends to the current value of the counter. Since with this implementation the host uses a single counter for all of its connections, we say that the host implements a *global ID field*.

(a) Suppose a host $\mathcal{P}$ implements a global ID field. Suppose further that $\mathcal{P}$ responds to ICMP ping requests. You control some other host $\mathcal{A}$. How can you test if $\mathcal{P}$ sent a packet to anyone (other than $\mathcal{A}$) within a certain one minute window? You are allowed to send your own packets to $\mathcal{P}$.

(b) Your goal now is to test whether a victim host $\mathcal{V}$ is running a server that accepts connection to port n (that is, test if $\mathcal{V}$ is listening on port n). You wish to hide the identity of your machine $\mathcal{A}$. Hence, $\mathcal{A}$ cannot directly send a packet to $\mathcal{V}$, unless that packet contains a spoofed source IP address. Explain how to use $\mathcal{P}$ to do this.

HINT: recall the following facts about TCP.

- A host that receives a SYN packet to an open port n sends back a SYN/ACK response to the source address.

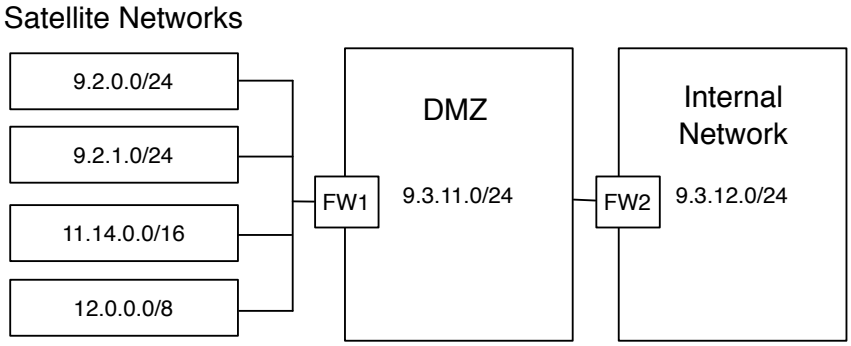
[1] For now, don't worry about how such fragments work. For this problem, we only care that this field exists and how hosts set it in the packets they send.

[2] Clearly, if a host sends more than 2^{16} packets, the field will necessarily repeat. That consideration doesn't matter for this problem, either!

- A host that receives a SYN packet to a closed port n sends back a RST packet to the source address.
- A host that receives a SYN/ACK packet that it is not expecting sends back a RST packet to the source address.
- A host that receives a RST packet sends back no response.

(c) How would you change $\mathcal{P}$ to avoid this problem? You are not allowed to modify the TCP/IP protocol or the services running on $\mathcal{P}$. You may only modify the *implementation* of TCP/IP on $\mathcal{P}$.

Problem 2 ***Firewalls*** **(15 points)**



The diagram above shows a private network used by a company. It includes a number of remote offices connected by satellite links. Like many enterprises, the main internal network is isolated from external networks using a *demilitarized zone*, or **DMZ**. (You can read about DMZs at http://en.wikipedia.org/wiki/DMZ_(computing).) You have been tasked with securing this network by ensuring that all of its network activity conforms to the following policy:

- Unless otherwise specified, all traffic should be denied.
- The satellite networks, except 12.0.0.0/8, should be able to communicate with any DMZ host over DNS (UDP port 53) and HTTP (TCP port 80).
- Satellite network 9.2.1.0/24 should be able to speak with 9.3.11.4 over SSH (TCP port 22).
- Nobody outside the DMZ should be able to contact the internal network.
- Any host in the internal network should be allowed to talk to hosts within the DMZ to the FTP service (TCP port 21).

(Updated 28Feb11: for all of these policy requirements, what matters is whether or not

bidirectional communication can occur. For example, the 4th requirement means that no host outside the DMZ can both send traffic to the internal network and receive traffic from it. To block such communication, it suffices to prevent *either* inbound or outbound communication. To enable it, you must allow *both* directions to work.)

You need to implement the policy by defining rules for both firewall FW1 and firewall FW2. For each firewall, list the rules you would use. You should aim to keep the rules as simple (minimal) as possible. Keep in mind the order in which rules are processed.

Problem 3 ***Exploring Network Access Properties*** **(10 points)**

"Netalyzr" is a Java applet that one can run from any browser to measure the properties of the Internet connectivity available to the browser.

Using any browser, 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.) At the end of its execution, Netalyzr provides a report summarizing different properties of the network access available to the browser from which you ran it. The report includes sections that you can expand by clicking on "+" or collapse by clicking on "−".

Inspect the Netalyzr 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 whether your browser's DNS resolver randomizes its source ports? Is the resolver vulnerable to the Kaminsky attack discussed in lecture?

Note: the example report that the Netalyzr home page links to includes output for a DNS resolver that does not randomize its source ports, highlighted in red. Also, do not confuse Netalyzr's reporting of the "ephemeral ports" selected by a NAT with the DNS resolver source ports.

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

(d) How can this report be useful to an attacker?

(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. (If you would rather not send us measurements from your home system, feel free to instead run from an instructional machine or AirBears or some other hot-spot.)

Problem 4 ***SQL Injection*** **(20 points)**

You join as a software developer at *Bittr*, a fancy new Web 2.0 startup that aims to gastronomically empower communities by enabling disintermediated folksonomies of vegetables.[3] Freshly equipped with your CS 161 knowledge, you are horrified to find the following code snippet in the Java codebase:

```
/*
Function to search for the username that we got from the URL parameters.

Author: Arthur Dent
dent@zz9p.za

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

(a) This code uses an API that's more feature-rich than necessary. Identify the API and describe why it is more feature rich than necessary. You can find documentation on `executeQuery` at: http://bit.ly/executequery

(b) You quickly send a email pointing out the issue to Arthur, telling him that the code is vulnerable to SQL injection. Arthur doesn't agree with your assessment. You decide that the only thing to do is to demonstrate a working SQL injection attack. Craft a value for `username` that will result in the table `Users` being dropped (deleted).

(c) On seeing your demonstration, Arthur apologizes and agrees that the issue is serious. Since he did not take CS 161, he decides to fix the code by backslash-escaping all single quotes in `username`.

```
username = username.replaceAll("'","\\\\'");
```

The code is still vulnerable to SQL injection. Make appropriate modifications to your answer above, so that the table `Users` will still be deleted. Explain why you made the modifications.

[3] No, the Bittr folks don't quite know what that buzz-phrase means, either. But they're pretty sure that people want it!

(Updated 28Feb11: the code in the example above actually needs four backslash-escapes to have the intended functionality. What's now listed as `'\\\\'` used to read `'\\'`.)

(d) Finally, Arthur sees the light and requests that you teach him a less feature-rich API. Explain why prepared statements match this description. For Java, these are documented at: http://bit.ly/preparedstatement

(e) To get Arthur started with prepared statements, you decide to demonstrate the concept with an example. Rewrite the `getProfile` function using prepared statements. Explain why your rewritten function is safe from SQL injection.

(f) While you go skiing in Tahoe, Arthur sits and fixes the code. Unfortunately, when you return you find Arthur—still in his bathrobe—waiting for you. He explains with dismay that he has found an example where prepared statements can't be used, as follows.

A Bittr user can search for vegetables based on their bitterness rating, and then order the results by either color or calories. For example, they might make requests that lead to queries such as the following:

```
SELECT * FROM vegetables WHERE bitterness >= 20 ORDER BY calories;
SELECT * FROM vegetables WHERE bitterness < 50 ORDER BY color;
```

To generate the second query, the user will have typed "< 50" in the text input box for bitterness, and chosen "color" in an associated drop down box. Both these strings are then sent as arguments to a function, `getVegetables`, that generates the SQL queries. In its original implementation, this function looks like:

```
ResultSet getVegetables(Connection conn, String bitterness, String orderBy)
        throws Exception {
    String q = "SELECT * FROM vegetables WHERE bitterness ";
    q += bitterness + " ORDER BY ";
    q += orderBy + ";"
    return conn.createStatement().executeQuery(q);
}
```

Arthur can't see how to rewrite this code using a prepared statement. What is the limitation of the prepared statement API that Arthur has hit? Why does the API impose this limitation?

(g) Rewrite the `getVegetables` to be safe by using prepared statements. You can modify it however you see fit, as long as you preserve the ability of users to make queries like those sketched above. Clearly explain the nature of your modifications.

Problem 5 ***Cross-Site Scripting (XSS)*** **(15 points)**

Prof. Hinkley comes up with what he believes is a simple solution for mitigating HTML "content injection" attacks such as cross-site scripting. He proposes introducing a new HTML tag `<NOXSS>`. In between `<NOXSS>` and `</NOXSS>`, JavaScript and plugin content (such as Flash or Java) is disabled: browsers will ignore (and in particular, not execute) any such content that appears between a `<NOXSS>` tag and its matching `</NOXSS>` tag. Prof. Hinkley suggests that web developers can use this to avoid cross-site scripting attacks: they should surround every place in their HTML page where they are including untrusted content with a `<NOXSS>` tag.

Additionally, developers have to include a random token in the `NOXSS` tag too.

For instance, consider the following vulnerable code:

```
w.write("Hello, " + name + "! Welcome back.\n");
```

Because `name` comes from user input, the above code has a XSS vulnerability. Prof. Hinkley proposes that instead of writing code like the above, the web developer should use:

```
Random rng = new Random();
int random_int = rng.nextInt();
w.write("Hello, <NOXSS" + random_int + ">" + name
    + "</NOXSS" + random_int + ">! Welcome back.\n");
```

Similarly, instead of writing

```
w.write("Today's most popular link is: "
    + "<A HREF=\"" + url + "\">" + url + "</A>\n");
```

which may be vulnerable, since `url` comes from user input, Prof. Hinkley proposes the web developer should write

```
int another_random_int = rng.nextInt();
w.write("Today's most popular link is: "
    + "<NOXSS" + another_random_int + "><A HREF=\"" + url
    + "\">" + url + "</A></NOXSS" + another_random_int + ">\n");
```

(a) What is the need for the random token (`random_int`) ?

(b) Should browsers process `img` tags inside the `NOXSS` environment? Why or why not?

(c) One of Prof. Hinkley's graduate students proposes an optimization: instead of generating random numbers for each untrusted content block, the whole page can share the same random number. Does this affect the security of the scheme? Why or Why not?

Problem 6 ***Web Security*** **(20 points)**

(a) Email services like GMail, Hotmail, etc., often change external links to first take you to another internal page and then forward you to the external page. For example, a link in an email message to `www.example.net` might be rewritten to `www.emailservice.net/redirect?www.example.net`. The latter might show a message or just redirect to `www.example.net`. State two reasons that a service might do this.

(b) We can view clickjacking as one example of the general issue of *UI hijacking*. The fundamental issue is that the UI's of two mutually untrusting components can overlap without the user's knowledge. Attackers can then exploit the user's unawareness by misleading the user regarding with which UI they are actually interacting.

In clickjacking, for example, the overlapping UI is the `iframe` of a web page. Identify another instance (not discussed in class) of UI's from untrusted sources overlapping. Can an attacker exploit the overlap? If so, sketch how; if not, explain why it does not appear possible.

(c) From the list below, pick any four sites. What are the values for the `X-XSS-Protection` and `X-FRAME-OPTIONS` response headers when a request is made to these sites (write N.A. if absent)?

Tip: A simple way to look at the HTTP Response headers is to use a command-line tool such as `wget` or `curl`. For `wget`, you can use `wget -S` *site*, and the tool will print the HTTP response headers in the form "<Header Name> : <Header Value>". For `curl`, you can get similar behavior using `curl -v`.

List of sites: `http://www.twitter.com`, `http://www.google.com`, `http://www.facebook.com`, `http://www.amazon.com`, `http://en.wikipedia.org`, `http://www.youtube.com`, `http://www.bing.com`, `http://www.microsoft.com`, `http://www.yahoo.com`

(d) As discussed in Section on Feb 23, a solution to CSRF is to use a *nonce* to bind requests to the form that the user is supposed to use to enter them. The solution posted on the website requires a server to keep state—the server has to remember the token it set on the `askpermission` page for the `withdraw` page. Modify the nonce solution so that the server doesn't have to store extra state at the server side, while still being safe from CSRF attacks.