CSci 5271: Introduction to Computer Security

Exercise Set 4 **due: Wednesday, November 20th, 2019**

Ground Rules. You may choose to complete these exercises in a group of up to three students. Each group should turn in **one** copy with the names of all group members on it; use the Canvas groups mechanism to indicate your group members if there is more than one. You may use any source you can find to help with this assignment but you **must** explicitly reference any source you use besides the lecture notes or textbook. An electronic PDF copy of your solution should be submitted on the Canvas by 11:59pm on Wednesday, November 20th.

0. Following directions. (1 point) Specify all the members of your group, both inside your PDF and using a Canvas assignment group.

1. Random numbers with limited entropy. (32 pts) Alice, Bob, and Carol are employees of a company (in a small island nation) setting up an online casino website based on card games like blackjack. They realize that if users could predict the sequence of pseudorandom numbers used to deal cards, they could win reliably and hurt the company's bottom line. They've found a good cryptographically-strong pseudorandom number generation algorithm to use in the shuffling process, but they're having trouble deciding what to use as the seed when they initialize the generator at the start of each user's session.

(Following the usual good security design principles, they don't want the security of the games to depend on the choice of the pseudorandom generator or the shuffling algorithm being secret; they might also want to franchise their casino out in the future. But practically speaking, reverse-engineering those algorithms would be a significant effort, so attacks that worked without the attacker needing to do so would be particularly damaging.)

(a) Alice suggests seeding the PRNG with the time: specifically the date and time as returned by the Unix `time` system call, equal to the number of seconds since midnight, January 1st 1970 UTC. Explain why this is a bad idea by describing an easy attack.

(b) Bob suggests seeding the PRNG with the process ID of the login CGI script. Assuming this script runs once each time a player logs in, and process ID numbers are assigned sequentially in the range of 2 to 65535, describe an attack against this scheme.

(c) Carol suggests combining Alice and Bob's ideas by taking the time and the PID and XORing them together. But Alice points out a problem with this scheme that involves a user logging in once every second. Explain the details of her attack and why it's a problem.

(d) After the problems with their previous schemes, Alice, Bob, and Carol have called you in as a consultant. Suppose that because of the architecture of the system, the seed is required to be a deterministic function of the time in seconds and the PID. Propose a better combining function that takes these two pieces of information as input and produces a bit string (of any length) than can be used as a seed. Would it help if the function could also take another input that was like a key, fixed per-site but secret? Evaluate the security of your approach.

2. Cross-site scripting variations. (17 pts) There are a lot of different kinds of cross-site scripting vulnerabilities, but for space reasons we only covered one of them in hands-on assignment 2. This question covers another. Here's an excerpt from some Java code in the 2014 implementation of question 6 from hands-on assignment 2:

```
public class MACCookieServlet extends GroupServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, IOException {
        String username = req.getParameter("username");
        if (username == null)
             username = "";
        String digest_hex = ...;
        resp.setStatus(HttpServletResponse.SC_OK);
        resp.setContentType("text/html;charset=utf-8");
        resp.getWriter().print("User \"");
        resp.getWriter().print(username);
        resp.getWriter().print("\" is identified with the MAC ");
        resp.getWriter().print(digest_hex);
        resp.getWriter().println("\".");
    }
}
```

This code suffers a reflected XSS vulnerability: the `username` parameter is under the control of the untrusted user, and it is copied directly into the HTML output. So if it contained JavaScript, that code would run with the site's permissions. There's no similar problem with `digest_hex`, because the omitted code ensures that it contains only hexadecimal digits.

One way to fix this vulnerability would be to sanitize the contents of the `username` string using HTML entities; for instance, translating each "<" into "`&lt;`". This is what we did for the newer version of the question (in PHP, we used `htmlspecialchars`). But suppose the programmer didn't know what library would contain a good implementation of that translation or was too lazy to implement it him or herself. What other simple change could you make to this code to avoid the cross-site-scripting danger?

3. TCP-Unfriendly. (30 pts) TCP's "congestion control" mechanism relies on *end-hosts* (i.e., users) to respond appropriately to network congestion by backing off their sending rate. One potential problem with this mechanism is what's called by economists the "tragedy of the commons." Suppose Alice knows that everyone else obeys TCP's congestion control mechanism. Then if she continues sending at the same rate, everyone else will slow down a little bit more and she will get better service from the network. So Alice has no motivation to obey TCP congestion control (other than the fact that not doing so involves finding or writing her own TCP stack—details, details) and in fact neither does anyone else. But if no one obeys the mechanism, the network (commons) becomes useless, which is the tragedy.

(a) Bob the Network Builder has an idea about how to solve this problem. He reasons that congested routers can see the *exact* state of a TCP connection. So if a particular connection does not slow down in response to dropped packets, the router can send a `RST` packet to each end of the connection. This will cause both ends of the connection to drop the connection, much more painful than just dropping an odd packet or two. From a *security* standpoint, what's the problem with Bob's idea—that is, if I'm an unscrupulous user intent on communicating at a high rate, can I circumvent this mechanism?

(b) When Bob realizes that reset packets aren't sufficient, he proposes a more direct approach: *blacklisting*. Under this idea, routers that notice TCP senders that don't respond to dropped packets appropriately will just stop routing packets for that sender. List several ways in which this is both ineffective against adversaries and a generally bad idea if adversaries got wind of it.

4. Firewall Schmirewall. (20 pts) Sarah is installing a network firewall for her company. Being familiar with the principle of fail-safe defaults, she has configured the firewall to DENY all packets by default. Now she needs to identify the minimal access rules that will allow her organization to use its Internet connection. For example, her organization will need to be able to send and receive email through the firewall, and uses a central mail server at IP address 10.1.100.100. So she has added rules to the firewall that look like this:

SRC ADDR	DEST ADDR	SRC PORT	DST PORT	PROTOCOL	ACTION
10.1.100.100	*	*	25 (SMTP)	TCP	ALLOW
*	10.1.100.100	*	25 (SMTP)	TCP	ALLOW

The organization has determined that it will also require the following kinds of Internet access:

- Incoming SSH access to a VPN server, at 10.1.100.200.
- Access to the web, through a proxy that whitelists approved sites. The proxy's address is 10.1.200.200.
- Outgoing SSH access to three client sites: 0.1.2.3, 42.42.42.42, and 3.14.15.9.

List the minimal set of firewall rules necessary to allow these connections. List some potential vulnerabilities associated with this ruleset. Can the firewall and proxy servers defend against these vulnerabilities?