CSci 5271: Introduction to Computer Security

Exercise Set 4 **due: December 8th, 2021**

Ground Rules. You may complete these exercises in a group of up to three students. Each group should turn in **one** set of answers, designating all group members using the group submission feature of Gradescope. You may use any source you can find to help with this assignment but you **must** reference any source you use besides the lecture notes or assigned readings. Submit an electronic copy of your solution online by 11:59pm on Wednesday, December 8th.

For this assignment, we will use Gradescope's template-based submission style. We will supply a template document with spaces to insert your answers in, in your choice of LaTeX or Google Docs formats. Prepare your answers by filling in this document and converting it to a PDF in a way that preserves the template formatting, and submit this to Gradescope. Note that we still ask that you type your written answers, rather than hand-writing and scanning them, since it is best for both you and us if your answers are easy to read.

1. Random numbers with limited entropy. (36 pts) Alice, Bob, and Carol are employees of a company 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. (18 pts) There are a lot of different kinds of cross-site scripting vulnerabilities, but for space reasons we only covered one of them in the hands-on assignment. 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. Denial of Service Denial. (22 pts) Sly is concerned about the possibility of DoS attacks against his web server program.

Sly has developed a new module for his web server that he claims will prevent DoS attacks by slowing them down. In Sly's module, every incoming HTTP request is put into a queue, with a timestamp and a "delayed" bit marked as false. When it is ready to serve a request, the web server takes the first request in the queue. If the "delayed" bit is false and there are no other requests from the same IP address in the queue, it serves the request immediately. If the "delayed" bit is false and there is at least one other request from the same IP address in the queue, the "delayed" bit is set to true and the request is re-inserted at the end of the queue. If the delayed bit is set to "true," then the request is served **if** the current time is at least 1 second greater than the request timestamp, and **otherwise** the request is sent to the end of the queue again. Sly's idea is that this will allow the site to deal with requests from legitimate users in preference to DoS attack requests.

Will Sly's scheme work to prevent a DoS attack from making his web server unusable by normal users? Give a detailed explanation.

4. Virus Virii. (24 pts) Sam has invented a brand-new virus detector, ViruSniff, and he claims it is "100% effective" — if executable file F is a virus, then ViruSniff(F) will output "`VIRUS!!!`".

(a) Does ViruSniff's claim conflict with the undecidability of the halting problem? Why or why not? (Hints: Read the claim literally. Is there another term besides "effectiveness" that describes that statistic that Sam claims is 100%? Is there a simple program that can do exactly what Sam says ViruSniff can do?)

(b) Unlike the hypothetical considered in (a), in fact some hackers reverse engineer ViruSniff and post its algorithm online. It turns out that ViruSniff does processor emulation of the first 10000 instructions of an executable, and then applies a neural-network based signature matching algorithm (that no one seems to understand) to the sequence of instructions and memory changes to decide if the program is a virus or not. Explain how to change any program that runs for at least 10001 instructions, and does not trigger the `VIRUS!!!` alert, to propagate a virus such that the altered program will also fail to trigger the alert. What does your strategy say about Sam's claim?