CSci 4271: Introduction to Computer Security

Problem Set 2 **due: Wednesday April 27th, 2022**

Ground Rules. This is an individual assignment. You may discuss the concepts behind these questions with other students, but you should formulate your answers individually and your answers must be entirely your own writing. You may use any paper or written online source that you find relevant to the questions but you **must** explicitly reference any source you use besides the lectures, labs, and readings. An electronic PDF copy of your solution should be submitted on Canvas by 11:59pm on Wednesday, April 27th.

1. OS interaction problems. (30 pts) The following C code appears in a program written to run on a Unix system. The intent of this code is to write the message contained in the string `secret_string` into a file named `secret-file.txt`. As indicated by the name of the file and the use of the `chmod` system call to change the permissions on the file, the key security concern of this code is to protect the confidentiality of the contents of the file. Other users on the system should not be able to figure out the secret. Unfortunately, some features of this code prevent it from reliably guaranteeing this. (If it matters for your answers, you can assume that the program is running in a public directory with a umask of 0000.)

```
int res;
FILE *fh = fopen("secret-file.txt", "w");
if (!fh) {
    fprintf(stderr, "Failed to open file for writing: %s\n",
            strerror(errno));
    exit(1);
}
res = fputs(secret_string, fh);
if (res == EOF) {
    fprintf(stderr, "Write failed: %s\n", strerror(errno));
    exit(1);
}
res = fclose(fh);
if (res == EOF) {
    fprintf(stderr, "Close failed: %s\n", strerror(errno));
    exit(1);
}
res = chmod("secret-file.txt", 0600);
if (res) {
    fprintf(stderr, "Chmod failed: %s\n", strerror(errno));
    exit(1);
}
```

(a) You can see that the author of this code was careful to check the return value of each standard-IO routine or system call that might fail. In case of any failure, the program exits with an informative error message. However, this choice of error handling behavior is not the best to protect confidentiality. Describe a scenario in which one of the function calls could fail that would be a problem for confidentiality.

(b) Even if every system call is successful, this code also has a race condition problem. Describe a sequence of events that includes both the steps taken by this code and actions taken by an attacker running at the same time, which could lead to the secret information being disclosed.

2. Another XSS defense. (20 pts)

In the days of the Internet before the dominance of the web, the `finger` program and protocol were a way of retrieving plain-text directory and status information (such a person's email address and the last time they logged in) across the network. During the transition to the web, it made sense for a while to have a web page that served as a *finger gateway*, to provide the same information from a `finger` request as a web page. Below is some Java code that might have been used to implement such a gateway:

```
public class FingerServlet extends MyServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, IOException {
        String username = req.getParameter("username");
        if (username == null)
             username = "";
        String info = getFingerInfo(username);
        resp.setStatus(HttpServletResponse.SC_OK);
        resp.setContentType("text/html;charset=utf-8");
        resp.getWriter().print("Finger information about the user \"");
        resp.getWriter().print(username);
        resp.getWriter().print("\":\n");
        resp.getWriter().print(info);
        resp.getWriter().println(".");
    }
}
```

Unfortunately, you may notice that this code has a reflected cross-site scripting vulnerability: the `username` variable is supplied via a GET parameter, and is copied directly into the HTML output. If it contained JavaScript, that code would run with the site's authority.

We have seen that one way to fix this vulnerability would be to sanitize the characters being copied into the response, such as replacing each "`<`" with "`&lt;`". However, since no part of this output needs to be interpreted as HTML, there is a different single line of this code could be changed to cause a web browser to interpret it the way we want: which is it?

3. Security definition for a MAC (20 pts)

We mentioned briefly in lecture that the standard definition of security for a MAC requires that an attacker with chosen-message access to the MAC can't forge MACs for any messages. Given that an attacker would normally achieve the ability to forge MACs by recovering the key, one might ask why we don't define security for a MAC more directly in terms of recovering the key. For instance we might say that a MAC is secure if an adversary with chosen-message access to a MAC can't recover the key (in polynomial time with a significantly better than random-guessing success probability).

Explain why this would not be a suitable definition. Recall the MAC from lab 11, where you found that with polynomially many chosen-message MACs you could compute $H(K)$. Should that MAC be considered secure?

4. Signing and hash collisions (30 pts)

We discussed in lecture how the usual application of public key encryption is *hybrid encryption* in which we first encrypt the data with a symmetric key, then encrypt the symmetric key with a public encryption key. For similar reasons, most uses of public key signatures begin by hashing a document with a cryptographic hash function, and then signing the hash with the signing key. This question concerns an attack that shows why it is important for the hash function used in this context to be collision resistant.

Suppose our potential victim Alice is using hash-then-sign on a receipt to agree to a purchase from an online store. Message class A consists of PDF files which are receipts for purchasing an item Alice wants to buy for a competitive price: for instance, a copy of the latest edition of Anderson's Security Engineering textbook for \$25. Message class B consists of PDF files which are receipts showing Alice paying an exorbitant price for an item she doesn't actually want: for instance, \$50,000 for a goat. Observe that we can easily create very large numbers of distinct messages of either class, such as by making combinations of small changes to the placement of lines and the spacing between words.

Suppose the hash function being used is one with a small enough output size, say 100 bits, that a birthday attack to find free collisions is feasible, but a preimage attack is infeasible. Explain how an attacker could defraud Alice by finding two messages with the same hash, using not much more computational effort than a standard birthday attack. Specifically, should the attacker try hashing messages just of class A, just of class B, or about equal numbers of both? Describe the complete attack and give a power-of-two estimate of how many hash function computations will be required.