Austin Strobel

1.

1. A website that allows customers to order products with their credit/debit card

a) Who might want to attack the system?
   - Attackers who are looking to steal people's credit card information might want to attack the system.

b) What types of harm might they want to cause?
   - They might want to use other people's credit card information, i.e. their money, to purchase items on a different website.

c) What kinds of vulnerabilities might they exploit to cause harm?
   - There would have to be a database of all the customers' credit card information, so the attacker could choose any account he or she so desires.

2. A social media website that allows anyone to sign up and post content.

a) Who might want to attack the system?
   - Someone looking to ultimately post misleading information may want to attack the system.

b) What types of harm might they want to cause?
   - They may want to post information about free bitcoin that leads to a phishing website, to give an example.

c) What kinds of vulnerabilities might they exploit to cause harm?
   - Unlike a social media platform like Instagram, a user could successfully set up a fake account using invalid credentials and spread misleading information.

3. An internet-connected thermostat that allows electric utility operators to adjust temperature to regulate power supply based on demand.

   a) Who might want to attack the system?
   - Someone looking to adjust the temperature to an unreasonable extreme or upcharge someone's utility bill might want to attack this system.

   b) What types of harm might they want to cause?
   - They might want to upcharge someone's utility bill and charge them for overheating or overcooling their house/apartment.

   c) What kinds of vulnerabilities might they exploit to cause harm?
   - The attacker could find a hardware vulnerability, such as with an FPGA, to manipulate the temperature.

2.

Security goals:

A general description would work, but CIA also satisfies this.

Confidentiality – The course instructor and related authorities should be the only individuals who are able to access the database.

Integrity – The only people who should be able to modify the database are the course instructor and related administrators.

Availability – Only the course instructor and related administrators should be able to access the database at any given time.

Reasonable attacks:

Someone could steal the laptop, a SQL injection could be used to get a list of several entries in the database, or the instructor's password could be hacked if the password is easy to guess.

Potential attackers:

Teaching assistants looking to expose the students' grades or students who wish to modify their own grades (or lower others' grades) may wish to hack into this database.

Threats we can exclude:
Attacks unrelated to hacking into a database would not need to be considered.

Two security mechanisms:

1. A safe to store the laptop

- Annual rate of occurrence: 0.2

- Single loss expectancy: $15,000

-> 0.2 * $15,000 = ALE = $3,000

-> Net risk reduction = $3,000 when a safe is used to store the laptop

2. Using MSSQL to more securely store the data in the database

- Annual rate of occurrence: 1.5

- Single loss expectancy: $20,000

-> 1.5 * $20,000 = $30,000

-> Net risk reduction = $30,000 when MSSQL is used to store the data in the database

3.

a) Vulnerability: occurs on the following line: $result = 'last -1000 | grep $username_to_look_for'

The username entered by the user is not validated for when it is used later in the Perl script.

- How to exploit it: An attacker could insert a command, such as “id” for the username, then the attacker could get an entire list of usernames.

To fix this, one should validate and verify user input before using it in commands to avoid this potential attack. A variable should be stored after the user gives their input and before it is used in a function like grep.

b) We would delete the last byte of the path “what/ever” by doing the following:

The function calls “stat” to see if “pathname” and “what/ever” are the same file. “Pathname” now has a link to itself created in “what/ever” with the link pointing to “what/ever.” We then read from “pathname.” Stat is then ran to observe if “pathname” now points to “what/ever.” This is where the race condition is used. The link also points back to “pathname.”

This silly_function should be executed with the pathname pointing to the file “what/ever.” A sym link, before the first stat call - stat(pathname, &f); - is reached, should be made under the name “what/ever.” Both the sym link and “what/ever” will point to the same file after the first stat call. Our new “pathname” and “what/ever” point to the same file “what/ever” because of the sym link we created. This new “pathname” will be opened, deleting the last byte of the “what/ever.” Essentially, silly_function only checks to see what file is being worked with at the start of the function rather than also checking throughout.