1.) A Time-of-check Time-of-use bug is a race condition in which the value you get when checking the state of a program is no longer valid by the time you use it. For example, consider a C program that writes some data to a file. When the program accesses that file, it is essentially "checking" the access to that file. However, a malicious user can use a well timed attack to change the file that is being accessed, say changing it to `/etc/passwd`, thus when the program actually writes to the file, the access that was checked previously is no longer valid when it is being used, thus the malicious user can overwrite the user password and cause damage.

2.) In order to take advantage of this vulnerability, a malicious user might provide the following inputs for "id" and "ssn":

id = "`12345' OR 1=1); --`"

ssn = "`123-45-6789`"

When the SQL instruction is executed, the inputs are simply passed into the instruction without being sanitized, resulting in the following:

```
select name from students where (id = '12345' OR 1=1);

-- ' and ssn='123-45-6789');
```

This always evaluates to true, so a malicious user could easily insert and run destructive commands by chaining them together with semicolons.

3.) Let `prog` be our root-owned Set-UID program. When `prog < /etc/shadow` is run, the `prog` command is run with root privileges, but the `< /etc/shadow` portion is attemping to redirect the contents of `/etc/shadow` with the regular user's permissions, which the OS doesn't allow, thus even if `prog` is a root-owned Set-UID program, we cannot view the contents of a restricted file without having root privileges ourselves. This is demonstrated by the fact that `sudo cat < /etc/shadow` does not print `/etc/shadow` even if the user has sudo privileges.

4.) a.) Reflected cross-site scripting is when a malicious site or user tricks you into traveling to a URL that points to a legitimate site but also contains malicious code. When the request is returned to the user, the browser executes the malicious code. An example of this would be if one were to visit an attacker's blog site, there might be a script attached to a button or link that makes a request to the user's bank, and thus the attacker would easily be able to run scripts with the privileges of the user's bank, possibly causing considerable damage.

b.) Cross-site request forgery is when an attacker uses a request made by a legitimate user to a legitimate website to perform unauthorized actions. For example, if a user is logged in to their bank website, that session might be stored in the browser's cookies. If said user were to visit a malicious blog, the attacker could take advantage of them by creating a request that uses the user's session cookies in order to appear legitimate, thus allowing the attacker to perform unauthorized actions.

c.) Click-jacking is when an attacker uses a transparent frame in order to trick the user into clicking a button or link that actually performs some other malicious action. For example, a malicious blog site could use a transparent button to trick the user into performing an action they did not intend to, such as sending the attacker money or giving them access to their bank account.

d.) Using seperate browsers prevents CSRF attacks, as these attacks rely on important information being stored in cookies. Say a user uses firefox for important websites and google chrome for recreational purposes. Since these cookies are not shared between the browsers, as long as the user is diligent in not visiting non-trusted websites on firefox, they are not at risk of a CSRF attack.