Question 1

a)

This is a violation of confidentiality and privacy. The violation of confidentiality comes from the fact that access to data (your medical information) was given to a party that was not authorized properly. The violation of privacy comes from the fact that information about you was given away to people without your consent. Since drug companies are not authorized to view your personal data and you did not give consent for them to do so this is a violation of confidentiality and privacy.

b)

This is a violation of integrity since the data that you are receiving (the malware) is most definitely not the information that you requested (the website).

c)

This is a violation of confidentiality since the hacker was surely not an authorized party, but they still gained access to your information.

d)

This is a violation of privacy because your information is being moved around and distributed without your consent and you no longer have control over who has access to it. The child can now do whatever they want with your information without your consent.

e)

This is a violation of confidentiality because data (your password) is being provided to parties that should not have access to it.

f)

This is a violation of availability since now you cannot access data when you want to.

Question 2

Threat 1 A person at the convention could change the label on one of the pegs to reference a different server with nefarious purposes. This would be classified as a modification threat because someone has modified the asset that you received.

Threat 2 If a person at the convention were to create a peg of their own and add it to the clothes line so that you might take it thinking that it was a legitimate peg it would be classified as a fabrication threat.

Thread 3 Someone at the convention could just throw out a bunch of pegs that would otherwise have been assets to other people at the convention, making it a interruption threat.

Threat 4 Someone at the convention could walk over and copy down the addresses on the currently available pegs allowing them access to the assets that you could be using. This would be a interception threat.

Question 3

a)

A digital lock is a bit of software that prevents the user from duplicating (amongst other actions) content that is protected by copyright. It is usually used as a defence against piracy.

Davidson, Eric. "Unlocking Bill C-11: What Are Digital Locks, and Why Should You Care?" The Fulcrum. Fulcrum Publishing Society, 28 Feb. 2012. Web. 23 May 2016.
`http://thefulcrum.ca/news/unlocking-bill-c-11-what-are-digital-locks-and-why-should-you-care`.

b)

In this case the FBI wanted Apple to build software that could break their encryption. This software could be used on any phone instead of a one off decryption key that they could invalidate later. Doing this would make breaking their encryption much easier for future hackers.

Alang, Navneet. "Best-kept Secrets: The Battle to Safeguard Our Privacy." The Globe and Mail. 29 Apr. 2016. Web. 23 May 2016.

c)

The statement makes sense if you are referring to physical locks and keys. The author is thinking of it as if a database has a lock to which there is a matching key, so any thing that has an identical lock can be unlocked with this same key. This is not at all what "breaking encryption" is or really how this works. My best guess is that the author is more thinking of authorization tokens or private keys. It really doesn't make much sense, if I get your "digital key" it will probably only give me access to the one thing that that key was for and only the information on that database you had 1access to. Knowing the secret key to one thing will not suddenly give you access to everything. If we liken it back to the authors lock key metaphor its like each database is protected by a lock that gives out some special keys to people with permission and those keys really only work on some parts of that specific database.

An Introduction to Cryptography. Santa Clara: Network Associates, 1990.

d)

We define privacy as the user have the right to control how their information is handled and where it is going. The magazine article admitting that users unwittingly hand over information acknowledges that it has already violated privacy by our definition since they didn't know they were handing over information and thus not controlling its use.

Sploit 1

The vulnerability is a buffer that gets populated without checking that the thing being loaded into the buffer will actually fit in it. This exploit is a buffer overflow attack on the `print_usage` function. This function loads a large string into a buffer of a set size. This string incorporates a variable that we can control so by setting this value to a very long string we can make the string larger than the buffer it is being loaded into. When this happens the string will continue being loaded into memory, but it will go below the buffer. Due to the structure of that function the buffer we are overflowing is the first variable on the stack so when we write below it we will first overwrite the stack frame pointer and then we can overwrite the return address. By overwriting the return address we can make the `print_usage` function return to our shellcode (I loaded it into a different parameter passed into the function, but it can be anywhere so long as you know the address for it). Once this shell code executes we have root access.

A way to fix this vulnerability is to check the size of the data being loaded into the buffer. Another solution is to just use a language that checks for array out of bounds exceptions. Most languages will throw an error and halt if you attempt to write passed the end of memory that you have allocated.

Sploit 2

This vulnerability is the code executing a executable without checking that it is the correct executable in `show_confirmation`. Here this is exploited by creating an executable file in the local directory called `ls` that creates a shell. When the code runs that command it will run the one in the local path instead of running the global one defined in the PATH environment variable.

A defence against this is to call the command by its full path. Another defence is to not just run commands and instead running c native code for printing the contents of a directory.

Sploit 3

This vulnerability comes from letting people control the environment variable for the execution of your program. This exploit works by changing the PATH variable used by the submit program to first point locally before looking in the standard places. Then the user can create a directory called `bin` in the local directory and populated it with any commands that do whatever so long as they are called in the program. In this case it overrides the `mkdir` function with one that creates a shell.

A defence against this is to not call shell scripts within c code and to instead use c native commands. Another is to check what your environment variables are before proceeding to see if anyone has altered them. You could also scan the executable you are about to run before running it to see if it does something unwanted.

Sploit 4

This vulnerability comes from loading a file into a bugger of a set size without checking the size of the file. A simple buffer overflow attack can be found by loading the correct characters into the file with the return address for the shellcode loaded into memory at the end. It works very similarly to sploit 1.

A defence against this is to check the size of the file you are about to load into the buffer or to stop loading characters into the buffer when its full size has been reached.