1)

a) Privacy - You have lost control of who gets to see information about you. Your doctor has given away your information for a use you did not intend.

b) Integrity - You are not getting the right data that you asked for. By replacing websites with malware, the attacker is providing malware (unwanted) instead of the websites you wanted to use (wanted)

c) Confidentiality - The hacker is an unauthorized person accessing you data. To be confidential, the system must limit access to data to authorized parties.

d) Privacy - The parent has lost control of information about themselves. The child can use it in ways the parent did not intend for.

e) Confidentiality - Your password is now available for use by unauthorized parties.

f) Availability - The buried internet cable may now be severed and not provide internet services as needed.

2)

a) Interception - You request a new address so you make your way to the central desk to retrieve it. On your way back to your desk, a peg bandit bumps into you and snatched the peg out of your hand. The bandit now has access to the address that is supposed to be yours.

Prevention - You use two pegs per address, each having one half of the address on it. You must retrieve the pegs with a partner before taking them back to your desk. This way, an attacker would have to steal the pegs from both you and your partner to intercept the data, highly unlikely.

b) Interruption - You request a new address so you make your way to the central desk to retrieve it. On your way back to your desk, the same peg bandit manages to steal the peg out of your hand and burns it, making it unreadable and the address lost.

Recovery - The central desk keeps a copy of each peg and has a sheet showing which address has been assigned to which person. If a peg is destroyed in transit, you simply return to the central desk and ask them for the copy.

c) Modification - The peg bandit managed to steal some pegs while the central desk wasn't looking and has changed the addressed written on them by erasing and rewriting them to point to some malicious data.

Prevention - The pegs have the addresses engraved in them instead of written on them. This way an attacker would have difficulty modifying the addresses.

d) Fabrication - The peg bandit has shown up to the convention with pegs of his own that have addresses written on them that point to some malicious data he wishes to spread. He leaves the pegs with the rest of the real pegs at the central desk, which are then handed out.

Deterrence - The central desk has all the pegs placed in a box with labels reading "Warning: alarm will sound if opened" and other security like measures (even though the security measures are not real). The peg bandit sees this and does not want to be detected, so he avoids opening the box, thinking that only the central desk has access to it.

3)

a) A digital lock, in context to copyright protection, is a means of Digital Right Management (DRM), used to prevent illegal copies of media from being made. They make it more difficult to for users to rip content from CDs, and DVDs, to be later distributed without the copyright owner's permission. "DRM technologies try to control the use, modification, and distribution of copyrighted works (such as software and multimedia content)" (https://www.priv.gc.ca/resource/fs-fi/02_05_d_32_e.asp). It is widely seen in the gaming industry which loses billions of dollars annually to piracy.

b) The difference with the government's request is that this time, they are asking Apple to disable some of IOS's security features in order for them to brute force their way into the iphone. The problem with creating a special version of IOS like this, is that it would allow anyone who could get a copy of it to be able to also brute force their way into an iphone. This would also allow the government to gain access to any iphone they wanted, not just the one in question in the San Bernardino case. The case has brought up the debate about data privacy between the government and technology companies, not a new topic in the world today. What made the case so high profile compared to previous requests from the government for assistance to gain access to private data, is the idea that the government would have the ability to access any encrypted data on any iphone. This idea sorted defeats the purpose of encryption in the first place, and as such has become a topic of major debate.

c) While what the author is saying is not 100% accurate, it does make some sense. The author is saying that the government is asking Apple to supply some sort of decryption software to be able to find out the password for an iphone. As such, anyone who would have access to this decryption software would then be able to find the password to any device that uses Apple's encryption technology and be able to access it. What the government is actually asking for is for Apple to disable some of it's security features (max amount of incorrect passwords, remove delay between password entries) to be able to brute force their way in. While this is not a key to every iphone outright, theoretically someone who could gain access to this version of IOS could possibly also brute force their way into any iphone.

d) By using the words "unwittingly hand over", the author is not really talking about privacy in the way we understand it. In our understanding of privacy, the user gets to control information about themselves, including who gets to see and use it and for what reasons. When the author mentions that users unwittingly hand over information, he is already talking about a breach of privacy. This goes against the Personal Information Protection and Electronic Documents Act which says that companies need to identify the purpose of data collection and obtain consent. By unwittingly handing over information, the companies attaining our info are already breaching these two principles. The privacy the author is talking about is simply about keeping our data out of the hands of unauthorized parties, this deals more so with security than privacy.

Works Cited

Arjun Kharpal, Apple vs FBI: All you need to know, http://www.cnbc.com/2016/03/29/apple-vs-fbi-all-you-need-to-know.html

Adam Satariano, The Behind-the-Scenes Fight Between Apple and the FBI, http://www.bloomberg.com/news/features/2016-03-20/the-behind-the-scenes-fight-between-apple-and-the-fbi

Dan Misener, Digital locks don't always work as intended, http://www.cbc.ca/news/technology/analysis-digital-locks-don-t-always-work-as-intended-1.1041761

Jane Lytvynenko, Unlocking Bill C-11: what are digital locks, and why should you care?, http://thesheaf.com/2012/03/25/unlocking-bill-c-11-what-are-digital-locks-and-why-should-you-care/

Digital rights management, https://en.wikipedia.org/wiki/Digital_rights_management

Digital Rights Management and Technical Protection Measures, https://www.priv.gc.ca/resource/fs-fi/02_05_d_32_e.asp

Programming Part

sploit1.c (Buffer Overflow)

There is a buffer overflow vulnerability in the copy_file() function. It uses a call to fgetc to copy the contents of the submitted file one character at a time. The loop used for this call has no bounds check and copies the contents of the file into the fixed size buffer, buf[2048]. If the file is larger than this buffer, we can keep writing to the stack and overwrite the return address of the copy_file() function. I create a string of NOP instruction followed by the shellcode. This string is passed as an environment variable who's address is found using gdb. The address of the environment variable is written to the file to be copied over and over so that it overwrites the return address during the file copy. Now the return address points to the shellcode and it is executed when the copy_file() function completes.

sploit2.c (Format String Exploit)

There is a format string exploit in the check_forbidden() function. There is a printf statement that takes it's argument as the source file name, which we control. We can pass along a string that will overwrite the return address of the printf function. The difficult part is calculating this string. Using gdb we find the return address of the printf function is stored at address 0xffbfd5ec. We will write over this address 4 bytes at a time (2 writes). Again, we store the shellcode in an environment variable at address 0xffbfd7e5, so we will overwrite the return address with this value. We use two garbage a characters in the format string to align it properly in memory, the format string is the 114th address on the stack.

d7e5 = 55269 - 10 (characters already written) = 55259

ffbf = 65471 - 55269 (characters already written) = 10202

So we write the 2 addresses of the return address (upper and lower half) plus the 2 garbage "a" characters (10 bytes), then we write 55259 characters and use a %n to write this (d7e5) to the lower half of the return address. We then write 10202 characters and us %n to write ffbf to the upper half of the return address. The printf function then returns to the shellcode and we gain root access.

sploit3.c (TOCTOU vulnerability)

It looks like there is a TOCTOU vulnerability in the get_logfile_name() function that calls getuid() and getpwuid(). By exploiting the time gap between the authentication and write to the logfile, we could possibly create a link to the password file and write to this file instead. We could then create an entry with a uid of root and pwd of our choosing and log in as a root user with these credentials. I was not able to develop this exploit.