CS458
Assignment 1
Andrew Chin
A8chin
20378330

1.
(a) A person installs spying software on their partner's phone.
This is a breach of privacy because private information can now be transmitted to a third party without the knowledge or control of the primary user. Depending on how the person was able to get access into their partner's phone (software hacking vs being loaned), this could also be a compromise of confidentiality.

(b) Google sells your demographic information to malicious hackers.
This is a breach of privacy because you are no longer in control of your personal demographic data. It can be seen and used by another without your permission.

(c) Someone breaks into a dating site and steals user account information.
This is a breach of confidentiality and privacy. This breaches confidentiality because an unauthorized entity, the hacker, has accessed information they did not has permission to. This is also a breach of privacy because the users no longer have control of the account information.

(d) Hackers secretly replace the software/firmware in your car.
This is a compromise of confidentiality because unauthorized hackers are gaining access to your car's system and replacing content/software. This is also a compromise of integrity because the driver is expecting to use certain software/firmware, but is actually using something else.

(e) A government replaces an illegal site with an identical-looking one designed to deliver malware to its users.
This is a breach of integrity. A user will request one site, but receive a replica whose purpose is different from the original .

(f) Lizards shut down the Internet.
This is a compromise of availability because a large amount of data is no longer there whenever you need it.

2A.
Interception:
Prescription drugs are particularly susceptible to an interception because they MUST be received from a dedicated location, a pharmacy. This means attackers can simply target customers of high traffic pharmacies such as Shoppers, Loblaws, etc, and intercept patients receiving their medication.

Interruption:
Pharmacies are typically part of larger stores or groceries out of convenience for the customer. This opens a vulnerability to the pharmacy as it is not in an isolated environment from the rest of the store. A fire, gas leak, or even a bomb threat against the store as a whole would force the pharmacy to close down as well preventing medicine distribution.

Modification:
Modification threats can also hit prescription drugs due to the way prescriptions are transcribed. When a physician prescribes medicine to a patient, they give the patient a hand written note describing quantity and type of medication to the pharmacist. Patients can modify this prescription note, and in particular modify the quantity to increase the amount to be received. This is dangerous as now the patient can overdose, sell, or share the medicine more freely.

Fabrication:
Chemical formulas for many common medicines have now become public due to expiring patents. This enables unauthorized third party drug manufacturers to produce and distribute non-standardized or verified drugs for much cheaper. These fabrications can be indistinguishable from legitimately produced medicine or be radically different and dangerous for human consumption.

2B.
You can prevent and deter the interception attacks described by adding armed security guards around premises of the pharmacy. This would deter petty thieves from attempting to attack and can also stop anyone that chooses to attack the pharmacy.

Interruption attacks could be deflected by having multiple pharmacies in close proximity to each other. This deflects attacks because even if one pharmacy becomes unavailable, there will be several others to fill the prescriptions. At worst, interrupting service would cause a minor inconvenience.

A detection of the modification attack described can be calling the physician to verify the prescription details. If the doctor can verify the details, then modification attacks can be detected.

Recovering from a fabrication attack can be in the form of improving existing medicine to make it more effective or longer lasting. Since newer formulas can have renewed patents therefore protecting the chemical formula. This mitigates the demand for third party medication since they won't be as effective.

3.
Duqu is a Trojan that can compromise Microsoft Windows systems. Appearing as a Microsoft Word Document, a zero-day exploit was utilized through the Win32k TrueType font parsing engine to trigger execution of malicious code. Duqu, like stuxnet, targeted industrial control systems and passed through the network to vulnerable machines, making it spread like a worm.

ZeuS is a Trojan that targets and infects Microsoft Windows systems. It also has the capabilities to install CryptoLocker Ransomeware. ZeuS, like other Trojans, spread through online downloads that seem innocuous, until run by an authorized user; ZeuS often targets sensitive banking information by using key logging and form grabbing.

Conficker is a worm that targets Microsoft Windows systems. It would target users with dictionary attacks to gain admin permissions and then spread itself to other systems that had the Microsoft Windows Server Service RPC Handling Remote Code Execution Vulnerability. Conficker was even updated to be spread through removable media.

Cryptowall is ransomware that affects Windows systems. It spreads around through spam emails, malicious ads from compromised websites, or through other malware.

4.
Sploit1
The first exploit targets a buffer overflow that occurs in copy_file(). This function will blindly copy contents of a source file, regardless of size, into a statically sized buffer(1024 bytes).

The exploit works by passing in a source file that is larger than 1024 bytes with shellcode strategically placed within. First the submit program will check this source file for viruses by searching for 'bin/sh', looking at 1024 bytes at a time. To bypass this check, 'bin/sh' needs to start at the end of a 1024 byte fragment and continue at the start of the next immediate 1024 byte fragment. The remaining bytes of the source file should be filled with the starting address of the shellcode so thzat when the buffer overflow occurs in copy_file(), its return address will point to the start of the shellcode.

This vulnerability can be fixed in several ways. One such way is dynamically allocating a buffer within the copy_file() function so that a buffer overflow doesn't occur. Another way would be to limit the size of the source file to 1024 bytes so that the static sized buffer wont overflow and the 'bin/sh' regex cannot be bypassed.

Sploit2
The second exploi1t targets a buffer overflow that occurs in the print_usage() function. This function has a call to snprintf() that will try to fill a 171 byte buffer with 640 bytes of data, thus causing a buffer overflow.

First, we must find a location to store our shellcode. For convenience, we will pass our shellcode in as the second argument to the submit program. The exploit then works because snprintf() copies a user supplied value to static sized buffer. The user provided value passed to snprintf() is argv[0], which is typically the name of the program itself. Instead, we set argv[0] to be an array filled with the address pointing to our shellcode stored in argv[2].
Finally, passing the '—help' flag to the submit program will trigger print_usage() and thus trigger the buffer overflow.

The vulnerability can be fixed by synchronizing the size of the buffer to the number of bytes snprintf() is looking to copy. As is, the buffer size is 171 bytes while snprintf() is looking for 640 bytes. If these two values were identical, this buffer overflow vulnerability would be present. Another option would be to print a static usage message, which would completely remove the need for a buffer.

Sploit3
The third exploit is a format string attack. It targets a vulnerability in the check_forbidden() function within the submit program. There is a printf() call with the source file name as its argument, so we are able to control exactly what is passed to printf() and execute a format string attack.

We first setup the exploit by passing our shellcode as the second argument to the submit program to discover what address it will be stored at. We also use GDB's 'info frame' to learn where the EIP for the check_forbidden() function call is stored. Now that we know where the EIP is stored, we add that address to the start of our format string so that we can overwrite the EIP with the address of our shellcode.

This vulnerability can be fixed by limiting the size of the source file's name. The format string that was used needed 116 copies of '%08x' in order to push the argument pointer far enough to reach itself. If we limited the length of the source file name, this format string attack would not be viable.

Sploit4
The fourth exploit targets a format string vulnerability in the print_version() function. There is a call to snprintf() which will fill a character buffer with a

predetermined message AND a user supplied string, so we are able to launch our attack through here.

The attack starts by passing our shellcode as our second argument to submit to discover where in memory it will be stored. We also again used GDB to learn where the EIP of the print_version() function call is stored. Similar to the previous exploit, we add the address of the stored EIP to the start of our format string so that we can overwrite it to the address of our shellcode.

Printing out a static version message instead of using a format string can stop this attack. The current version message checks the name of the calling program, but it should away be named “submit”. Another vulnerability is the call the snprintf() which is filling the character buffer that is being passed to printf(). This character buffer is 641 bytes in size, which is definitely too large for the need it fulfills. Reduce the size of the buffer so that a format string cannot be long enough to work.