Assignment 1

Written Response Question

Q1a)

Confidentiality is compromised.

A breach of confidentiality happens when information provided by the end user in confidence is supplied to a third party without his/her consent. If Google supplied that information, it would've been a breach of the Confidentiality principle.

Q1b)

Availability is compromised.

As Google had supplied the information to the FBI, from the bureau's perspective availability would have been compromised as the flow of data would be disrupted.

Q1c)

Privacy is compromised.

Confidentiality is when the user willingly provides data to a third party on the condition that the information isn't divulged.

Privacy is the right of a person to be left alone and is the more appropriate principle that was compromised. In this scenario, the FBI invaded the privacies of users whose calls were intercepted.

Q1d)

Integrity is compromised.

By removing her team's data from the database of Google's, she has modified the data in an unauthorized manner which affects the accuracy of the data in the database.

Q2a)

Modification. The hacker had hacked Google's database and removed the data from it.

Q2b)

Interception. The hacker's activity is being monitored by the FBI without their knowledge.

Q2c)

Fabrication. The FBI created entries in a database that is virtually indistinguishable from the real thing.

Q2d)

Fabrication. Again, the data is being modified and is not distinguishable from the real thing.

Q3a)

Deflecting. Plant false information on someone (patsy) to steer the FBI in the wrong direction.

Q3b)

Detecting. Monitor the FBI's communications to prepare for possible raids.

Q3c)

Recovering. Become an informant for the FBI in exchange for leniency/immunity from prosecution.

Q4a)

Trojan. It spreads when malicious email attachments are opened. Once installed and the user restarts his computer, the program does URL calls to websites owned by the KOVTER developer to generate revenue via click-fraud.

Q4b)

Worm. ILOVEYOU is also sent via email attachments as a VBS script file. User who thought that it was a normal txt file would run it which in turn runs the Visual Basic script. This would result in random files on the machine being overwritten. The worm would then spread itself by sending a copy of the script to all email addresses in the Windows Address Book.

Q4c)

Worm. Mirai spreads itself by scanning vulnerable IoT devices running Linux and attempting to log into them via factory default usernames and passwords. The infected device then waits on a command and control server which assigns a target for the devices to flood with requests causing a DDoS attack.

Q4d)

Worm. NotPetya spreads itself through a variety of methods including a backdoor implemented in an accounting software package (M.E. Doc) which is used by majority in the Ukraine as well as via the EternalBlue and EternalRomance exploits. It masquerades as ransomware with a similar ransom screen as Petya but encrypts files on the computer with no way to revert the encryption.

Sploit 3

This exploit uses a TOCTTOU vulnerability found in the program.

The vulnerability exists at line 244 and line 297.

Time of check

```
path_real = realpath(path, NULL);
if (path_real != NULL && strstr(path_real, forbidden) == path_real)
```

Time of use

```
if (copyFile(src, dst) < 0)
```

The time of check conditional statement checks whether our path has a valid path i.e. the path cannot contain /etc/ as the passwd file is in that directory. These lines of code prevents symbolic links from working at the time of check since realpath() would resolves the symbolic link to an actual path. If we were to create a symbolic link to the /etc/ directory it would fail.

We can, however, create the symbolic link AFTER this check and BEFORE line 297 when the file will be restored such that it is restored into /etc/

The flow of the program is as follows:

1. Create a passwd file containing a `hacker::0:0:root:/root:/bin/bash` entry. This is a root alias entry with UID of 0 which gives it root privileges. Note that this alias does not have a password.

2. Backup our created passwd file.

3. Restore our passwd file from the backup directory. At the same time, create a symbolic link with the name “passwd” that redirects to “/etc/passwd”. If timed correctly, during the

copyFile() function's fopen, the destination would be pointed to /etc/passwd instead of /share/passwd via the symbolic link.

4. Our restored file will overwrite the existing passwd file.

5. Run "su hacker". This will give us root access.

Of course, we would have to run the sploit multiple times since the timing window is small. Our sploit uses the Linux C's fork() function to create a child function which creates a symbolic link. The parent function is the one running the restore command.

How can it be fixed

The simple fix for TOCTTOU vulnerabilities is to immediately use the file after authenticating it so that the timing window for an attack is so small that its nearly impossible to exploit.

Sploit 5

Vulnerability in backupV2

As the pdf states, backup and backupV2 differs by only one line of code. The Linux bash's diff command allows us to compare our two disassemblies generated using objdump.

```
791,792d790
< 8d 9d 60 ff ff ff       lea    0xffffff60(%ebp),%ebx
< 83 c4 f4                add    $0xfffffff4,%esp
795,800c793
< e8 99 f3 ff ff          call   8048918 <strlen@plt>
< 83 c4 10                add    $0x10,%esp
< 89 c0                   mov    %eax,%eax
< 8d 14 18                lea    (%eax,%ebx,1),%edx
< 52                      push   %edx
< e8 5b f4 ff ff          call   80489e8 <strncpy@plt>
---
> e8 a6 f2 ff ff          call   804884c <strncat@plt>
```

The difference between the two binaries is that strncat is used in backupV2 at line 285 of backup.c instead of the strncpy function. This is easily deduced as this portion of the disassembly is found towards the end of the main function.

Through inference with the original backup.c program, we can surmise that the replacement at line 285 is:

```
strncat(errorMsg, path, sizeof(errorMsg) - strlen(errorMsg)) //assumed
backupV2 code
```

The problem with using strncat is that, unlike strcpy, it appends a NULL character '\0' at the end of the concatenated string. As the buffer errorMsg is 152 bytes and the buffer already has `strcpy(errorMsg, "Not your file: ")` copied into it, this leaves us with 137 bytes of character left to use. If we were to make the "path" variable 137 bytes long, strncat appends an additional NULL character at the end thereby overflowing it by one byte.

But what does overflowing it by one byte do? If we look at how the variables are declared at the start of the main function:

```
int cmd;
int permissionsError = 0;
char errorMsg[152];
char *path, *path_stripped, *path_real;
char *forbidden = FORBIDDEN_DIRECTORY;
char *src, *dst, *buffer;
struct stat buf;
```

By overflowing errorMsg by one byte, we overwrite the permissionError variable and set it to 0 since a NULL character has an int value of 0. The code

```
if (permissionsError) {
    fprintf(stderr, "Exiting due to permissions error: %s\n", errorMsg);
    return 1;
  }
```

Is supposed to guard against invalid file paths. However, since we've set permissionsError to 0 before the conditional statement, we've effectively bypassed this check.

This means that as long as our filename is 137 bytes long, we would be able to create a symlink from our passwd file that has been named to be 137 characters long (we'll called it longName for brevity's sake) even if it contains the forbidden directory /etc/.

This is almost an exact replica of our TOCTTOU exploit. Since the code

```
if (path_real != NULL && strstr(path_real, forbidden) == path_real) {

    permissionsError  = 1;
    strcpy(errorMsg, "Not allowed to access target/source: ");
    strncpy(errorMsg+strlen(errorMsg), path, sizeof(errorMsg)-
strlen(errorMsg));
 }
```

Sets permissionsError to 1 as our symlink is pointed to /etc/password from longName. However since we overwrite the permissionsError to 0, our path name goes through to copyFile. The /etc/passwd file is then replaced with our own created longName passwd file which contains a root alias with no password.

We then simply execute “su hacker” to gain root access.

How can it be fixed

When using strncat we must make sure that our buffer is able to accommodate the appended ‘\0’. As such, the number of characters that can be concatenated should be sizeof(errorMsg) – strlen(errorMsg) – 1. The minus one is to accommodate for the NULL byte.