Homework #1

Radhika Mattoo, rm3485@nyu.edu

1.

a. The vulnerability in the Equifax incident is the use of the flawed software, Apache Struts. The threat was that the flaw was disclosed in March, but Equifax failed to detect and fix the bug. This allowed attackers to exploit their software vulnerability and gain access to private information of 145.5 million people. Thus, this incident was a violation of confidentiality. What we have learned from this incident is that investing in a security team and updating software as soon as it's available is an absolute must when you have users that trust you with their PII. The main security control that Equifax could've used to prevent/mitigate this attack was to encrypt their data (part of communication security) to conceal data against unauthorized access, which they shockingly did not do. (Source: https://www.cbsnews.com/news/equifax-ex-ceo-hacked-data-wasnt-encrypted/)

b. The vulnerability in the Uber data breach was keeping AWS credentials in their core code. The threat was that the code was publicly published on Github, allowing attackers to easily access the credentials. This lead to attackers obtaining the personal information of 57 million people. Instead of reporting the breach to those whose data was compromised, Uber instead quietly paid the attackers a $100,000 ransom to delete the data and kept it a secret for over a year. Thus, this is a violation of confidentiality. What we learned from this breach is to make sure you never publish code with sensitive data (and to add sensitive config files to your .gitignore), but also to not trust the attackers themselves. Just because they were paid their ransom does not necessarily mean they actually deleted all the sensitive information they obtained. Some security controls that could prevent/mitigate this are to encrypt your data (communication security), putting their code in a private repository (access controls) and teaching their developers to double check their code and commits before publishing (admin controls). (Source: https://www.wired.com/story/uber-paid-off-hackers-to-hide-a-57-million-user-data-breach/).

c. The vulnerability in the NHS cyber attack is a bug in Microsoft software. The threat was that many users don't always install updates/patches on their computer, or delay it for a long time. Thus, attackers were able to use social engineering and phishing emails to trick users into opening attachments that

download the malware, called WannaCry. Thus, companies like the NHS and FedEx along with countries including Russia, Taiwan, Ukraine, and India found their files were permanently encrypted and required a hefty ransom to decrypt them, or else have their files deleted. Thus, this is a violation of confidentiality, integrity, and availability. It violates confidentiality, as attackers were able to access a user's private file system. It violates integrity because they encrypt the files, thereby modifying data. Finally, it violates availability because the user cannot access their files until they pay the ransom. What we have learned from this incident is that software updates and security patches should be installed as soon as possible in order to protect oneself against an attack. Security controls that could prevent/mitigate this are to keep OS software up to date so that there are no bugs/holes in how someone can access the system (access control). Another security control is teaching users how to identify phishing emails, probably through security training (admin controls).

2. Typing in your username and password every time you want to log in to a website can be tedious – with a different password for every website, it can become ridiculous to remember all your passwords, aka it becomes a burden. So, many users use browser autofill to store login information and autofill the username and password fields on login pages, aka an 'end-run' around security mechanisms. While this minimizes the burden of security mechanisms enforcement on the user, at the same it opens them to exposure of their sensitive login data. Last month it was found that popular ad firms like Adthink and OnAudience were using third party scripts to create hidden login forms on a website's page, which is then autofilled by the browser. The agency then simply has to log out the values inserted into the fields in order to obtain the user's login information. I believe that the principle can in fact be applied to the design of the autofill feature by tweaking when exactly the browser autofills forms. For example, the browser should never autofill forms that have a hidden attribute. While this is just one example, there are other ways to prevent this kind of confidentiality violation without increasing the burden on the user. (Sources: https://teiss.co.uk/news/ad-firms-steal-user-credentials/, https://www.theverge.com/2017/12/30/16829804/browser-password-manager-adthink-princeton-research).

3. Based on what I've previously read, I believe that the leading causes for preventable security breaches include not encrypting sensitive data, injections, and phishing. I generally agree with the arguments made by Ross Anderson, specifically that "most security failures are due to implementation … errors" and that developers "fail to ensure that the skills and performance required of various kinds of staff are

included…in the certification loop" (pgs. 12, 11). These statements are reflected in each of my leading causes listed above. As seen in the readings from question 1, many major companies simply do not even encrypt sensitive data, meaning any attacker that could gain access to their system has unfettered access to PII (implementation error). Similarly, injections are #1 on OWASP's top 10 web vulnerabilities, and can expose sensitive database information or even crash the entire database (implementation error). Finally, phishing is a result of social engineering that 'tricks' untrained or unaware users into giving attackers sensitive information (skills of various staff). While I agree with several things Anderson states, I still believe this paradigm shift he discusses has a long way to go before there is a fundamental change in cryptosystem design.

Algorithm Explanation for Password Cracking Program:

The first thing I did when I started this portion of the homework was read up on the given Java hashing function and Base64 encoding. I decided to take an object-oriented route and created a Password class representing the password I want to crack. After reading in the password text file and creating Password object out of them, the question became how do I actually crack the passwords? Since the instructions specified it was allowed to use a dictionary attack, I began downloading various different large wordlists online, mostly from Github. I then read in each line of the dictionary, added the Password object's decoded salt, hashed the word, and compared the result to the Password object's hash. This was repeated for every password object. I found that this dictionary (https://github.com/berzerk0/Probable-Wordlists/blob/master/Real-Passwords/Top95Thousand-probable.txt) worked best for me, and actually cracked 6 of the passwords. The question then became, what am I missing with the last 4 passwords? The answer was permutations of the dictionary. I used the functions permutation, buildNumbersAndSpecial, and mixCases to modify the existing wordlist and hashed these new strings to see if they matched. Unfortunately, I was not able to use all permutations generated in my dictionary attack in the allotted time.