Vincent Yasi

CS 370 -Intro to Security-

Problem Set 4

User Authentication and Passwords

Q1[10 pts]: You are designing a password system with randomly selected passwords. The alphabet for the passwords is the set of alphanumeric characters in English both upper and lower case and the integers 0-9. You are told that the attacker can make 250,000 guesses each minute.

a. **If the passwords are 7 characters long, how long until the attacker has a 50% probability of correctly guessing user's passwords in an offline attack.**

   As the characters possible are the upper- and lower-case letters, as well as 0-9, that gives us a set of 62 possible characters. As each password is 7 characters long, that gives us 62^7 possible passwords.

   If we put the information we have into Anderson's Formula ($P = (\frac{TG}{N})$), we can solve for T, which is the time until P is 0.5. We get $0.5 = \left(\frac{T*250{,}000\ per\ minute}{62^7}\right)$, or 13.4 years (7,043,229.21 minutes).

b. **How long do the passwords need to be to ensure that the 50% success rate is not reached until after 2 years?**

   To answer this, we use $T = 1051200$ minutes (2 years) in Anderson's Formula, and solve for x in $N = 62^x$. This gives us $0.5 = (\frac{1051200*250000}{62^x})$. Solving for x, this gives us about 6.54, which means the passwords need to be about 6 and a half characters, or 7 in reality (which we can see above is actually much more than 2 years).

c. **If the users select their own passwords, does this affect the relevance of your calculations from parts (a) and (b)? Explain your answer.**

   The calculations above are based on an assumption that all passwords are random and evenly distributed in the set of all possible passwords. If users choose their passwords, this will likely not be the case. Therefore, some passwords (like common words that are 7 characters long) will be much more common than others. This will lower the time needed to figure them out by dictionary attacks, etc., as well as technically shrink the size of the set.

Q2 [4 pts]: iPhone 6 includes a fingerprint scanner which the user can choose (not) to use. Do you think activating fingerprint scanning would increase the security of the cellphone? Why or why not?

It would increase the security as fingerprints are very hard to reproduce and cannot (typically) be hacked from things like data breaches. They are also unique to each individual and easily accessible. Their use is also very easy for the individual (as long as the reader is good) and does not pose much of an additional hurdle to a user to use their phone. Fingerprints provide a simple, unique, difficult to replicate security feature.

Q3 [3pts]: Bloom filter is an efficient way to preemptively reject bad passwords with high efficiency, but it has a false positive rate (incorrectly rejecting good passwords). What can you do to decrease the chance of a false positive?

There are two main ways the chance of a false positive can be decreased. The first is to increase the number of possible hash results (and increase the size of the resulting hash table used to store Bloom filter results). This decreases the chance that multiple different hashes of bad passwords will overlap to give all 1s for the hash of a good password.

The second method is to use hash algorithms with high collision resistance. With high collision resistance, the hashes are less likely to produce the same hash for both a bad password and a good password, which is another possible cause of false positives.

Q4 [3 pts]: Why will a bloom filter never give a false negative (accept a bad password)?

The hash table is built from the hashes of bad passwords. The data that is used to check if the hash of a password is one of a bad password is literally the hashes of bad passwords. If a password is bad, its hashes are already in the table. If its hash is not in the table, then by definition it is not a bad password (as long as all hashes of bad passwords are indeed in the table).

Q5 [3 pts]: It is common practice not to store user's password in clear text. However, if an attacker has seized control of the password database, he is likely already capable of modifying any user data on the site as an administrator. Why bother hashing the passwords then?

Hashing passwords does not protect against modifying the passwords from attackers with admin privileges. Instead, what it does is protect the password itself from being read or decrypted. An attacker with appropriate privileges and knowledge can modify the password, but the hashing protects against being able to know what the password initially was. This information can then not be easily used elsewhere or decrypted to gain additional information.

Q6 [3 pts]: It is common practice to salt the user's password in addition to hashing. What attack does this practice prevent?

This protects against brute-force and similar attacks. Hashing a password offers a certain level of protection and difficulty to trying to decipher it (let us say n time/effort to get a useable result). Adding salting to the hash then increases this another level, as not only does the attacker have to break the hash, they then need to test each possible salt, as well. If we have, say, k salts that can be used, the overall effort now rises from n to kn, orders of magnitude larger. This is where increased protection with salting comes from.

Q7 [4pts]: Does a "salt" used in password hashing need to be kept secret? Why or why not? Compare and contrast "salts" and "initialization vectors (IVs)" used in CBC encryption mode.

The salt does need to be kept hidden. As the hashing algorithm is often public (as it typically has pre-image resistance), the salt is what gives the multiplicative effect described above in Q6. If the salt was also known, then the attacker would know the hash algorithm used and which salt with that hash. This would negate the above protection and turn it back into an n level effort (as they do not need to check all salts for the hash). This is further compounded by the fact that hashes do not (typically) have keys, and so there is not that added layer of protection/random to the attacking process. In a way, the salt IS the key, and so should be treated like such.

IVs, on the other hand, are used in block ciphers which also have keys. The IV is just used as the initial value/block fed into the cipher, but the encryption/randomness comes from the interactions of the key and message blocks to give the encryption. Therefore, even if the IV is known, as long as the key (and preferably the blocks) is kept secret, the overall scheme is still secure.