CS 5435 – Computer Security

Homework 1

Name: Bar Kadosh, Ben Kadosh *Fall 2019*

For this assignment, exercises in sections 1 and 2 required python code deliverables, which have been submitted along with this report. These python scripts contain comments that explain our code and approach. This report captures the responses to exercises 3.1-3.4.

Exercise 3.1

1. In an offline brute-force guessing attack, what limits the number of guesses q made by an attacker? In an online (also called remoted) brute-force guessing attack, what limits the number of guesses q made by an attacker?

> In an offline brute-force guessing attack, password hashing is one of the main factors that limits the number of guesses, q, made by an attacker by making it computationally time intensive for the attacker to calculate any given password. This is achieved by using a large number, c, of iterations to re-hash the password. Furthermore, using random per-user salts will increase the complexity of the password and require attackers to spend even more time determining what a password is. Using salts for passwords will further slow down the attacker and decrease the number of guesses, q, an attacker can make.
>
> In an online brute-force guessing attack, lockout mechanisms will limit the number of guesses, q, made by an attacker before locking them out. For example, after 10 unsuccessful attempts, the system might lock the user out for a certain period of time. This severely hinders the attacker, as brute forcing is no longer a viable option with such few attempts.

2. Suppose the adversary makes some number of q guesses. What is the optimal guessing strategy and what is its probability of success (as a function of p)? This is called the q-success probability of the distribution p.

> The optimal guessing strategy in this case would be to identify the q passwords with the highest probability of success and guess each of them.

The probability of success for the optimal guessing strategy would be the summation of the probability of each of the most probable passwords, where i = 1 is the case for the most probable password and each subsequent password up until the qth password is the next most probable:

$$\sum_{i=1}^{q} p(pw_i)$$

3. Define Shannon entropy and give an example of a distribution p whose q-success probability is high and whose Shannon entropy is also high. Explain why Shannon entropy is a misleading estimate of password strength.

Shannon entropy can be defined as the measure of randomness for a given probability distribution. As the distribution has the possibility for more and more "random" outcomes, or events with low probabilities, the distribution becomes flatter and the Shannon entropy increases. A distribution with higher entropy is one that has a fairly high degree of randomness, whereas distributions with lower entropies have less randomness for each event in the distribution.

The following is an example of a distribution with high q-success probability and high Shannon entropy: suppose there are 100,002 possible passwords for a given probability distribution and that q = 2 for the number of guesses to properly guess the password. Now suppose that:

$p(pw_1) = 0.2$
$p(pw_2) = 0.1$
$p(pw_3)$ through $p(pw_{100,002}) = 0.000007$ (each)

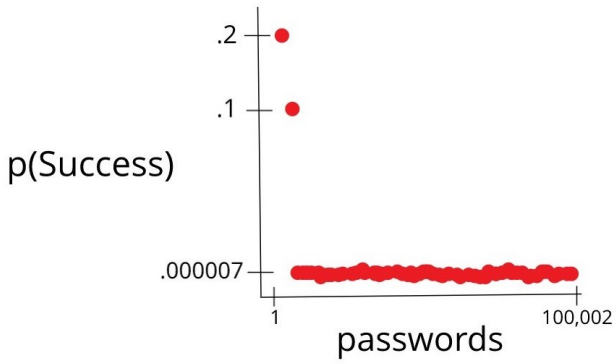

Figure 1: p(Success) vs passwords

This results in a q-success probability of 0.3 and an entropy of approximately 3.85. This distribution shows both a high q-success probability and a high Shannon entropy.

This can be generalized to similar distributions with q guesses as well: in general, a distribution p with q guesses will have a high q-success probability and high Shannon entropy if the q most probable passwords are all fairly probable (greater than 0.01 as an arbitrary threshold example) and the n remaining passwords in the distribution all have fairly low probabilities (less than 0.00001 as an arbitrary threshold example), where n is significantly greater than q. This means that the vast majority of the passwords have very low probability (high Shannon entropy) but that the q-success probability is still high if the q most probable passwords have high probability.

Lastly, Shannon entropy is a misleading estimate of password strength because as we saw in the example above, the distribution as a whole could be viewed as random and with mainly low probabilities but still have several events with high probabilities leading to a high q-success probability. If there is a high q-success probability, then the attacker could attempt to exploit only those events with high non-random probabilities in an easy fashion, while the Shannon entropy of the distribution is still high and indicates that it should be difficult to find success as an attacker (which is not actually the case in such scenarios).

Exercise 3.2

Salts for password hashing prevent use of rainbow tables and, more generally, ensure that cracking effort against one users hash is not reusable for another users hash. Some web services will additionally employ a pepper, which is a single random value also added to the password hashing input and stored separately from the password hashes. Explain the purpose of a pepper.

A pepper is quite similar to a salt – however, a pepper is a secret value that is stored separately from the hash output (meaning that it is not available to view with the hashed password like a salt is). The purpose of the pepper is to add an additional security measure by diversifying risk. For example, unless the attacker is able to crack the pepper, they can not begin to attempt cracking the hashes. As we have been learning, while brute force hashing can be slowed down with salts, the only limit to cracking the password is time. However, if a pepper is added in, the hacker can not begin attempting to crack the hash until they figure out the secret pepper key. Additionally, peppers arent necessarily hackable – the attacker must figure out the secret key. Having a pepper and a salt provides the benefits of both and allows for a multi-layered and diversified security strategy.

Exercise 3.3 Describe how you would modify the web service to limit the efficacy of online guessing attacks.

One could modify a web service to limit the efficacy of online guessing attacks by:

1. Using a smaller threshold for lockouts (lock out the user after 10 failed attempts, as opposed to 1,000, for example).

2. Enforcing strong random password usage during registration through a password meter to reduce the risk of attackers using common passwords to breach an account. By making sure users are creating strong passwords, we limit the probability of attackers breaking in by guessing probable/poorly chosen passwords.

3. Requiring dual factor authentication when logging in. This ensures that even if a password is compromised, the actual individual can have a second line of defense and can confirm if the person trying to access their account is actually them or an attacker.

4. Potentially analyzing the IP address from which the log in is being attempted and establishing a threshold the system can use to decide if it is likely that the user is actually present in the geographic region the IP supports (for example, if a user that registered and typically logs in from New York has an attempt come through from Russia, the login should be considered suspicious).

Exercise 3.4 A colleague suggests you might strengthen password selection by enforcing a password composition policy, such as requiring an upper case letter, lower case letter, and symbol. Provide an explanation of why this is a bad idea. What should you do instead to encourage stronger password selection?

Enforcing a password composition policy would be bad for several reasons, the most notable of which is that it focuses more on composition (as a proxy for complexity), and less on actually assessing password complexity itself.

For example, password composition alone could still easily accept a weak password such as (Str@wberry1) and disallow a strong password such as (*#24@msxpr!l) because it doesn't have capital letters. Note: the strong password could be strengthened with the use of capital letters, but they were omitted for the purposes of the example.

In addition, password composition could actually work against users. If we were to enforce a password composition policy, one which would be clearly visible to attackers, we would in essence be reducing the cardinality (size) of the set of all possible passwords, making it easier for the attacker to generate all permutations of possible passwords and test them more quickly.

Instead, to encourage stronger password selection, one should employ a password strength estimator to assess password complexity and prohibit users from using passwords that are deemed not strong enough. Strength and complexity can be determined based on many things: password length, common words/patterns appearing in the password, unexpected letters or words following each other in the password, usage of a combination of uppercase, lowercase, numeric, and symbol characters or lack thereof, etc.