Tom Hills
CS583, Dr. Ruoti
Project 7: Password Cracking

REPORT:

For this lab, I used JohnTheRipper (JTR) jumbo version from repo, https://github.com/magnumripper/JohnTheRipper, as this was the recommended program in the lab writeup by Dr. Ruoti. I had thought about using Hash Suite for Windows since it was a GUI program and would be easier to understand and use. However, I wasn't willing to spend 40 bucks to get a version that would be able to handle passwords up to 8 characters long. I tried to enable GPU usage by downloading appropriate drivers, but Ubuntu had issues and so I abandoned this idea in favor of using the significantly slower CPU processing.

My "average rate for password checks a second" was dependent on the strategy I used. Figure 1 below shows the output from the "status" command in JTR for each strategy that I wasn't able to complete all the way:

```
iLower:
0g 0:00:54:36  0g/s 21985p/s 186138c/s 186138C/s

iAlpha:
1g 0:01:32:23  0.000180g/s 46522p/s 408914c/s 408914C/s

iAlnum:
1g 0:19:29:50  0.000014g/s 15961p/s 112507c/s 112507C/s
```

Figure 1: Rates for Incremental (Brute Force) Methods of JTR

In Figure 1, the units have the following definitions:

- g/s: successful guesses per second
- p/s: candidate passwords tested per second
- c/s: "crypts" (password hash or cipher computations) per second
- C/s: combinations of candidate password and target hash per second

All three strategies listed in Figure 1 were incremental (i.e., brute-force) strategies as follows:

- iLower: brute-force guessing with alphabet {*a – z}*
- iAlpha: brute-force guessing with alphabet {*a – z, A – Z*}
- iAlnum: brute-force guessing with alphabet {*a – z, A – Z, 0 – 9*}

As can be seen, the rates vary significantly based on the alphabet used for brute-force methods.

Unfortunately, these were the last strategies I used. To start, I used the wordlist.txt file provided by Dr. Ruoti combined with preset mangling rules provided in the jumbo version of JTR. In particular, I used

the “Jumbo” rules which ran, in order, the preset mangling rules “Single”, “Wordlist”, and “Extra”. This was, by far, the best strategy and netted me 8 out of 13 total successful passwords guessed in less than ten minutes. The rest of my strategies were brute-force attempts based on the provided password requirements of the writeup. First, I tried all lower case letters a through z. This netted me 3 out of 13 total successful passwords guessed in about 3 hours (Figure 1 says 54 minutes and 36 seconds, but there was a previous run performed for testing purposes which didn’t get recorded) before I decided to move to the next strategy due to time constraints. Next, I started over with using all alphabet characters, both lower and upper case and performed this for 1 hour, 32 minutes, and 23 seconds. After seeing no passwords cracked and considering that numbers could be included to, I decided to put most of my remaining time into a strategy that included all lower and upper case alphabet letters as well as all single digits. I continued this strategy for 19 hours, 29 minutes, and 50 seconds which provided another 2 out of 13 total successful passwords guessed.

Figure 2 below lists the user and passwords I was able to successfully crack from the initial 20 provided.

```
user1:41t3rn4t3::::::
user4:int3rn4ti0n411y::::::
user5:lkyru::::::
user6:forwarding::::::
user7:increased::::::
user8:31iz4b3th::::::
user12:uddg::::::
user14:longitude::::::
user15:ta::::::
user16:revolutionary::::::
user17:lr4::::::
user19:cincinnati::::::
user20:lsv::::::

13 password hashes cracked, 7 left
```

Figure 2: Usernames with Successfully Cracked Passwords

Users 1, 4, 6, 7, 8, 14, 16, and 19 were cracked using the provided wordlist.txt and “Jumbo” mangling rules, Users 12, 15, and 20 were cracked using the “iLower” strategy, and the remaining passwords for Users 5 and 17 were cracked using the “iAlnum” strategy.

Tom Hills
CS583, Dr. Ruoti
Project 7: Password Cracking

QUESTIONS:

1. Assuming that you used your setup for this project alone, how long do you calculate that it would take to crack a 6-character alphanumeric password? 8-characters? 10-characters? 12-characters? Give exact estimates for either the average time needed to crack a password of the given length and composition, or the max time needed. Show your work.

   ANSWER:
   For alphanumeric passwords, I would have used my "iAlnum" strategy which had an *average* guess rate of 112,507 c/s. The alphanumeric character set has 62 members so a 6 character password would have $62^6 = 56{,}800{,}235{,}584$ possibilities. Likewise, an 8 character password would have $62^8 = 218{,}340{,}105{,}584{,}896$ possibilities, a 10 character password would have $62^{10} = 839{,}299{,}365{,}868{,}340{,}224$ possibilities, and a 12 character password would have $62^{12} = 3{,}226{,}266{,}762{,}397{,}899{,}821{,}056$ possibilities. From these values, we can simply divide by the guess rate to produce the average number of seconds to run through all possibilities:
   - 6 character password: 504859.5694845654 seconds ≈ 0.015998 years
   - 8 character password: 1940680185.0986695 seconds ≈ 61.497707 years
   - 10 character password: 7459974631519.285 seconds ≈ 236,397.185366 years
   - 12 character password: 2.8676142483560132e+16 seconds ≈ 908,710,780.5456 years

2. Recently, high-end GPUs have revolutionized password cracking. An RTX 3080 (a recent high-end GPU) is able to check 2.5 billion passwords a second using MD5 crypt. Using this guessing rate, estimate how long it would take to crack the passwords described in question 1.

   ANSWER:
   Substituting 2.5e9 c/s for 112, 507 c/s and performing the same calculations, we get:
   - 6 character password: 22.7200942336 seconds ≈ 7.1997E-7 years
   - 8 character password: 87336.0422339584 seconds ≈ 0.002768 years
   - 10 character password: 335719746.3473361 seconds ≈ 10.638535 years
   - 12 character password: 1290506704959.16 seconds ≈ 40,894.529515 years

3. Do you think that the password meter is a good indication of actual password security? From the results of your experiment, what is your recommendation for minimum password length? Be creative in your response. Imagine what hardware and resources a potential attacker might have, and briefly justify your assessment of the attacker's capabilities.

   ANSWER:
   a) No, I don't think the password meter is a good indication of actual password security because users have a tendency to use predictable patterns which can completely undermine security when an attacker uses a dictionary or other word mangling rules on a wordlist.

b) My recommendation for minimum length passwords is 12 😊. Reasoning? It's VERY unlikely that attacker will have enough resources to brute force even an alphanumeric password of 12 characters minimum. If you expand the password alphabet further, it only gets exponentially more difficult. And let's face it: It's not hard to come up with a 12 character password, so might as well require it!

4. Fedora 14 and other modern Linux distributions use a SHA-512 (rather than MD5) for hashing passwords. Does the use of this hashing algorithm improve password security in some way? Why or why not?

   ANSWER:
   If you take two hashing implementations and pass in salted passwords, then the better implementation is the one with higher:
   A. Resistance to preimages: given x, it is infeasible to find m such that HASH (m) = x.
   B. Resistance to second-preimages: given m, it is infeasible to find m' distinct from m and such that HASH(m) = HASH(m').
   C. Resistance to collisions: it is infeasible to find m and m', distinct from each other, and such that HASH(m) = HASH(m').

   In particular, if one implementation has a better collision resistance (item C), then it will also have better resistances for items A and B inherently. So yes, if we assume properly salted implementations for both SHA-512 and MD5 (which we should), then using SHA-512 improves password security by the simple fact that having a longer hash (512 bits for SHA-512 and 128 bits for MD5) increases hash collision resistance (item C).

5. Does the use of a salt increase password security? Why or why not?

   ANSWER:
   Yes, definitely. Using rainbow tables, wordlists with mangling rules, and related attack strategies can have a catastrophic impact on password security as user-chosen passwords almost always exhibit recognizable character patterns greatly reducing the "guess space". This vulnerability can be averted by adding randomness to the HASH input which salting accomplishes.

6. Against any competent system, an online attack of this nature would not be possible due to network lag, timeouts, and throttling by the system administrator. Does this knowledge lessen the importance of offline password attack protection?

   ANSWER:
   No, because offline attacks don't have the benefit of being monitored and controlled as the question describes for an online attack. Unless a perfect system can be created which completely prevents theft of server password files, keys, and/or other sensitive security data, offline protection measures should ALWAYS be considered when implementing security.