Matt Christensen (mrc621)

November 1, 2021

CS 465 (001) – Clift, Frederic M

Project #7 – Password Cracking Report

Experiment

In order to test the strength of passwords we needed a way of generating the /etc/passwd and /etc/shadow Unix password file formats. To do this we created a python script which properly handed each of the file formats correctly:

```
def generate_passwd(username):
    ret = username + ":x:" + user_id + ":" + group_id + ":" + user_info + ":directory:shell11"

    return ret
```

```
def generate_shadow(username, password):
  encrypted_password = crypt.crypt(password, crypt.METHOD_MD5)

  ret = username + ":" + encrypted_password + ":" + changed_days + ":" + min_days + ":"
        + max_days + ":" + warn_days + ":::"

  return ret
```

These functions are used to create files such as passwd:

"*us:x:1000:1000:GECOS:directory:shell11*"

and shadow:

"*us:1KUgvksR/$Vcqk.eHJ8A4PFwBRVTb941:30:0:99999:7:::*".

We then "unshadowed" the files by running "*unshadow passwd shadow > unshadowed-.txt*" and ran John the Ripper in an attempt to crack serval passwords. The results of this experiment are listed in the table below:

Password	Time (hr:min)	Reliability	c/s
password	0:00	100.00%	400
Password	0:00	100.00%	15466
password123	0:00	100.00%	13840
password123!	0:01	100.00%	69950
KyleStory23!	0:02	100.00%	103145
aNs2Werm?aGazineproDuc?4e	1:02	18.00%	15921
N"K=^\9]qgnFds<+	1:05	20.00%	16432

Questions

1. *Assuming that you used your setup for this lab alone, how long do you calculate that it would take to crack a 6-character alphanumeric password? 8-characters? 10-characters? 12-characters? (use the c/s measurement from your experiments).*

Here we assume the average comparisons per second to be 15,000. We then calculate the totally possible number of combinations to be $(26 + 26 + 10)^{len} = 62^{len}$. The average case is found by multiplying half of the number of possible combinations by one over the rate of comparisons. We find the worst possible case by assuming we will find our password on the very last comparison; thus, it is the number of all possible combinations multiplied by one over the rate of comparisons. We find:

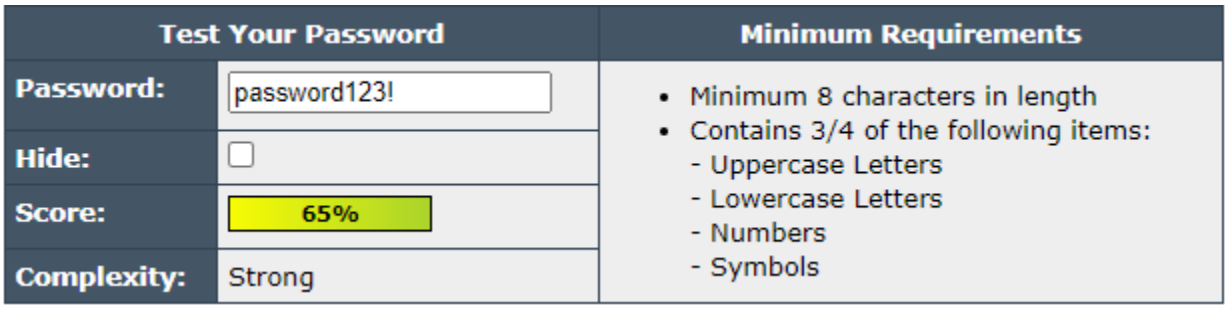
Length	Combinations	c/s	Average Case (Days)	Worst Case (Days)
6	56800235584	15000	22	44
8	218340105584896	15000	84236	168472
10	839299365868340000	15000	323803768	647607535
12	3226266762397900000000	15000	1244701683024	2489403366048
14	1240176943465750000000000	15000	4784633269543800	9569266539087600
18	1832527121610300000000000000000000	15000	7069934882755770000000	14139869765511500000000

2. *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.*

I think that the password meter can be better than nothing for non-technical users that do not put in the time to learn what constitutes a secure password. But at the end of the day, the

strength of a password is mainly found in length as long as your password isn't in a wordlist. Making sure your password isn't in a wordlist can be facilitated in a couple of ways. First, don't reuse passwords (if your password is leaked somewhere else you could be compromised). Second, don't use common passwords or common password patterns that could be easily generated. Finally, it is important for the person storing your password to do so securely (salted and hashed).

By way of example, even though the password meter says that our password of "*password123!*" is strong, because it is in our lookup list, we can find it in one second (see the first table in the report).



As for the recommended minimum length of password – this depends on a few things. Mainly, the rate of comparisons over a given time. With our previous estimation of 15,000 computations per second, this was found on an older computer (built in 2014). Maybe we estimate a 400% improvement in calculations per second to future proof our password for another ten to twenty years. Thus, we are looking at ~60,000 per second. And maybe the attacker has access to twenty-four servers with these previously calculated specs. Thus, over the three servers, we get a total of ~1,500,000 comparisons per second. Finally, lets assume that after three days the attack will give up and move to the next password. Thus, we would need a password length of 10 in order to withstand the average case of 8,870 years. This is assuming that brute force is needed to crack the password (because the password did not appear in the attacker's wordlist). Pretty good!

Length	Combinations	c/s	Average Case (Years)
10	839299365868340000.00	1500000	8871.34

3. *Recently, high-end GPUs have revolutionized password cracking. One tool, ighashgpu, is able to perform 1.3 billion MD5 hashes per second on an AMD Radeon 5850 (a 2-year-old, mid-to-high range video card). Whitepixel, another tool, claims that it can perform 33.1 billion hashes per second using 4 Radeon 5970s. Consider your calculations in question #1, and redo them assuming you had access to a system with 4 Radeon 5970s. Do your answers for question #2 change?*

Assuming our comparison rate to 33.1 billion hashes per second, we update our table:

Length	Combinations	c/s	Average Case (Days)	Worst Case (Days)
6	56800235584	33100000000	0	0
8	218340105584896	33100000000	0	0
10	839299365868340000	33100000000	147	293
12	3226266762397900000000	33100000000	564064	1128128
14	1240176943465750000000000	33100000000	2168262811	4336525622
18	183252712161030000000000000000000	33100000000	32038979831219500	64077959662439000

In order to achieve a similar level of security against cracking measures, against the rate of 33.1 billion hashes per second, we would need a length of thirteen:

Length	Combinations	c/s	Average Case (Years)
13	2000285392686700000000000	33100000000	96077

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?*

In some ways SHA-512 does offer a strong security claim than MD5. MD5 is considered cryptographically broken, which means that certain properties are no longer necessarily guaranteed. Like being able to find hash collisions. Additionally, if passwords are not salted MD5 is very insecure as many rainbow tables have been created over the years and makes hash lookup trivial and quick.

However, when it comes to brute forcing passwords, it simply comes down to how fast it takes to compute the hash. In this regard, SHA-512 once again proves itself more secure because of its slower performance:

Hash	Case 1 (ms)	Case 2 (ms)	Case 3 (ms)	Case 4 (ms)	Case 5 (ms)	Case 6 (ms)
MD5	627.4	765.6	1488.8	839	1029.4	1738.2
SHA-1	604	748.2	1325	916.8	1009.6	1632.4
SHA-256	737.8	851	1504.4	1168.2	1260	1963.6
SHA-512	1056.4	1158.8	1837.4	1118.4	1227.4	1923

Because each hash takes longer to generate the comparisons per second will drop and passwords will take longer to crack.

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

Salts do increase the security of a password. As discussed above, by including the salt in the hashing of the password we prevent attackers from using pre-generated tables of hashes (think rainbow tables) and simply attempting a lookup on the leaked passwords. Additionally, by using a different salt for every password, work done to crack one password doesn't translate to another – because the different salts will cause the hash to be different, even if it is the same password.

So, salting always adds security and should always be done.

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?*

If anything, I think the fact that online attacks are so hard makes it even more important to defend against offline ones. Because the main front of password defense is diverted to offline attacks, we need to make sure that resistance to such attacks are as strong as possible and are not the weakest link of the system. If offline defense is weak then the moment passwords are leaked attackers would have instant access to accounts, not giving time to system administrators and users to react.

7. *OPTIONAL QUESTION: The sheer power of GPUs make John the Ripper pale in comparison, despite all the heuristics John the Ripper employs. With hardware continually becoming more powerful, do you foresee a day in the near future when minimal password lengths will be too large for a typical person to remember? If so, what types of vulnerabilities may arise from such a scenario? What would you recommend as the next step in password evolution?*

I don't think passwords, based on the calculations in this paper, will necessarily grow to be so long that we cannot easily remember them; unless quantum computing makes a large leap, but even this should be preventable with quantum resistant algorithms. However, this is assuming that the password is not readily found in a wordlist. To get a password that is guaranteed to not be found on a password list then one could argue it needs to be truly random and not reused.

Remembering ten random characters for dozens of sites is a lot to ask for. And in my opinion, the next step in password evolution could be secure password managers. All one needs to do is make sure the manager is crypto-safe and then remember one really important and good password. Then the app can remember the long and random passwords for the user.

Conclusion

In conclusion, it is important that we think of ways to protect the user from backend attacks while also learning how to best educate users on best password practices. Both of these areas serve as potential risks for the inevitable attacks that will occur.