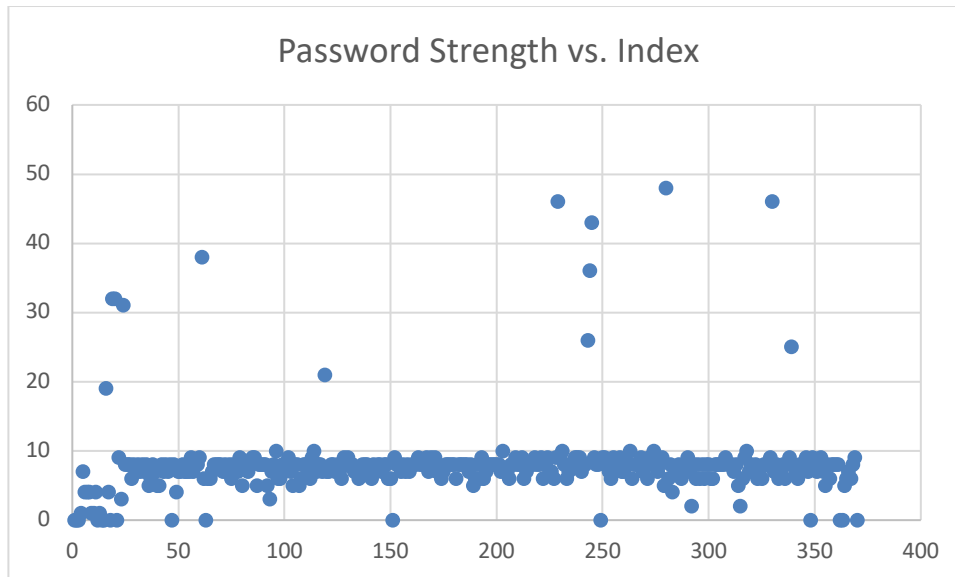


Topic 1

Step 2: Analyze Banned Passwords

Question 1: A scatter plot of the password strength distribution: X-axis is index, Y-axis is strength

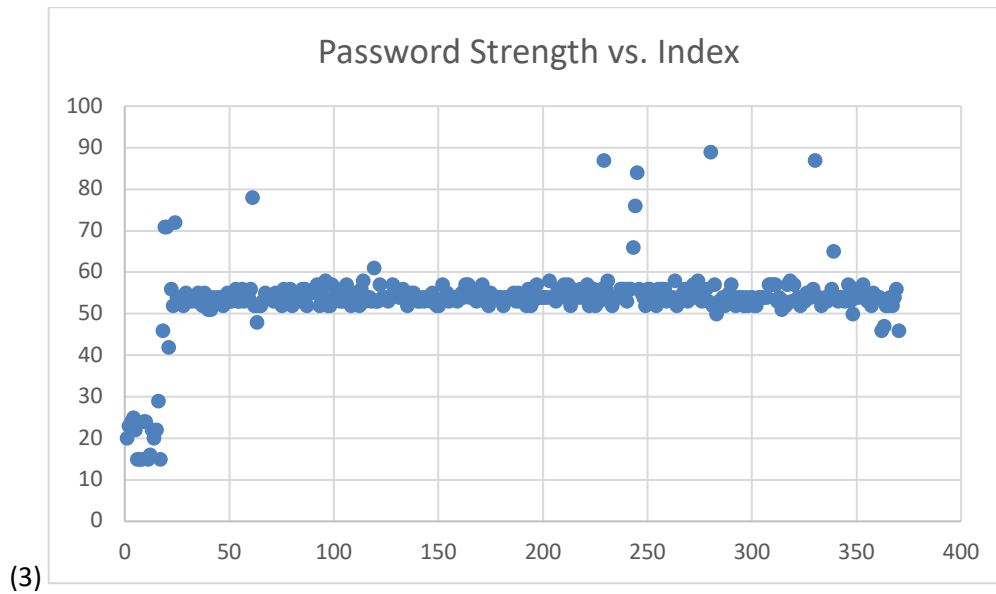
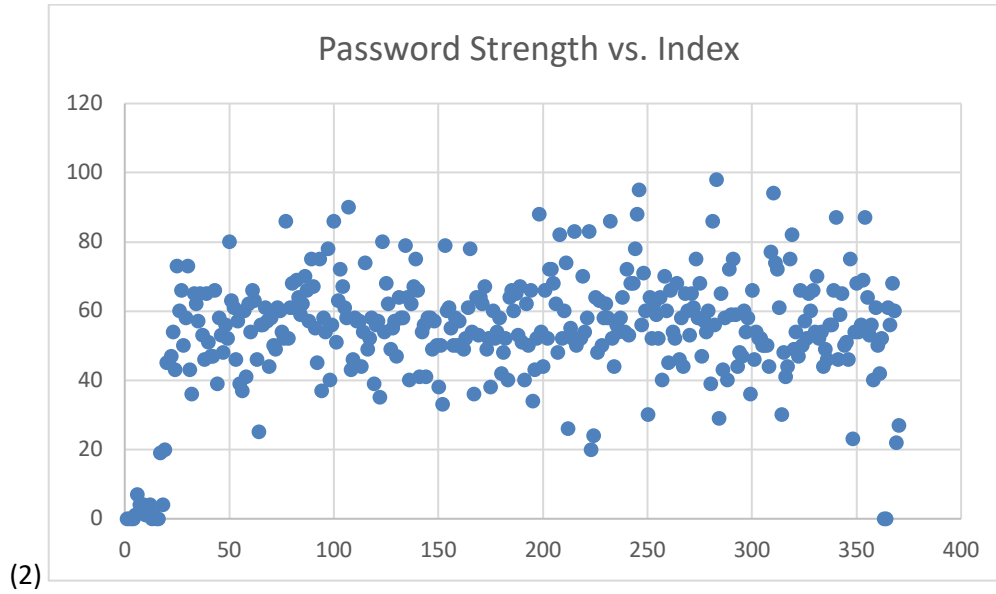


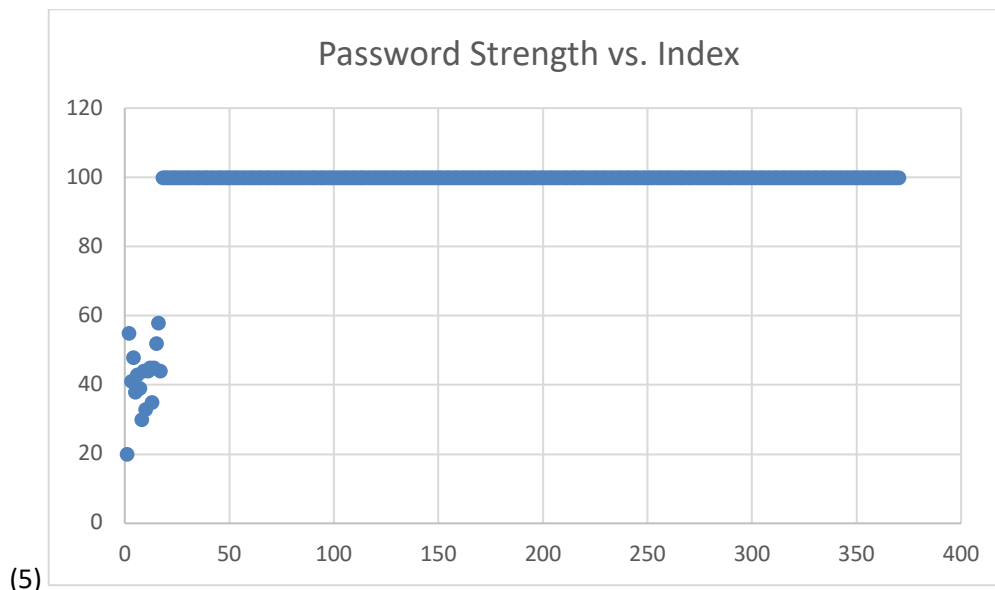
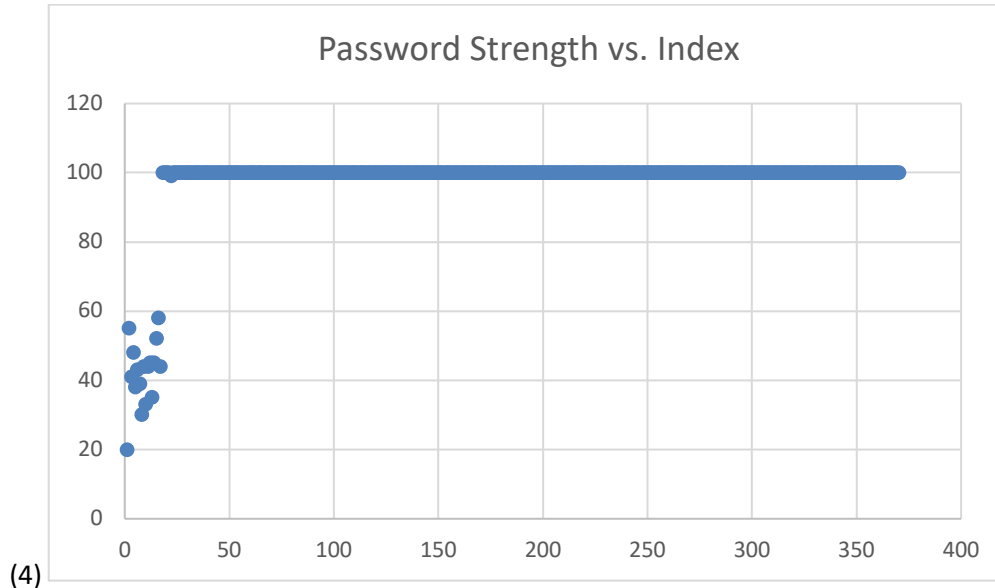
Question 2:

Things that TPM takes into consideration are: number of characters, uppercase letters, lowercase letters, numbers, symbols middle numbers or symbols, requirements (min 8 characters, contains ¾ of uppercase, lowercase, numbers, symbols), letters only, numbers only, repeat characters (case insensitive), consecutive uppercase letters, consecutive lowercase letters, consecutive numbers, sequential letters (3+), sequential numbers (3+), and sequential symbols (3+). We think it is a fine password checker. It is not particularly great because if you simply have a long password it will give it a high enough score. For example, "ILiKeCoMpUtErScleNcE" gets a score of 97, but it probably would not be a good password as it says I like computer science. The bulk of the score here comes from the length. Simply adding one number to the end of that gives it a score of 100, but it definitely is not an uncrackable password.

Step 3: Modify Passwords to Generate New Password Files

Question 3: Scatter plots of the password strength distribution: X-axis is index, Y-axis is strength





Note: The numbers of the files correspond to the step that I generated them on, so (2) is making the replacements to original file, (3) is adding four random digits, (4) is creating combinations, and (5) is the replacements to (4).

Observations: (2) is already stronger than the original file. (3), where we added four random digits to the end of each password also is stronger than the original file. Between (2) and (3), (2) has a lot more strength variability than (3), so adding four digits to the end of each password made them such that they were all about the same strength. (4) and (5) are both significantly stronger than the original file, and the other files, they both have strengths of 100 for almost all of the passwords.

Step 6: Use JTR To Crack the Passwords In the Hashed Password Files

Question 4: Incremental will try all character combinations as possible passwords, but wordlist will just try the words in the wordlist and the words with the rules applied from the wordlist.

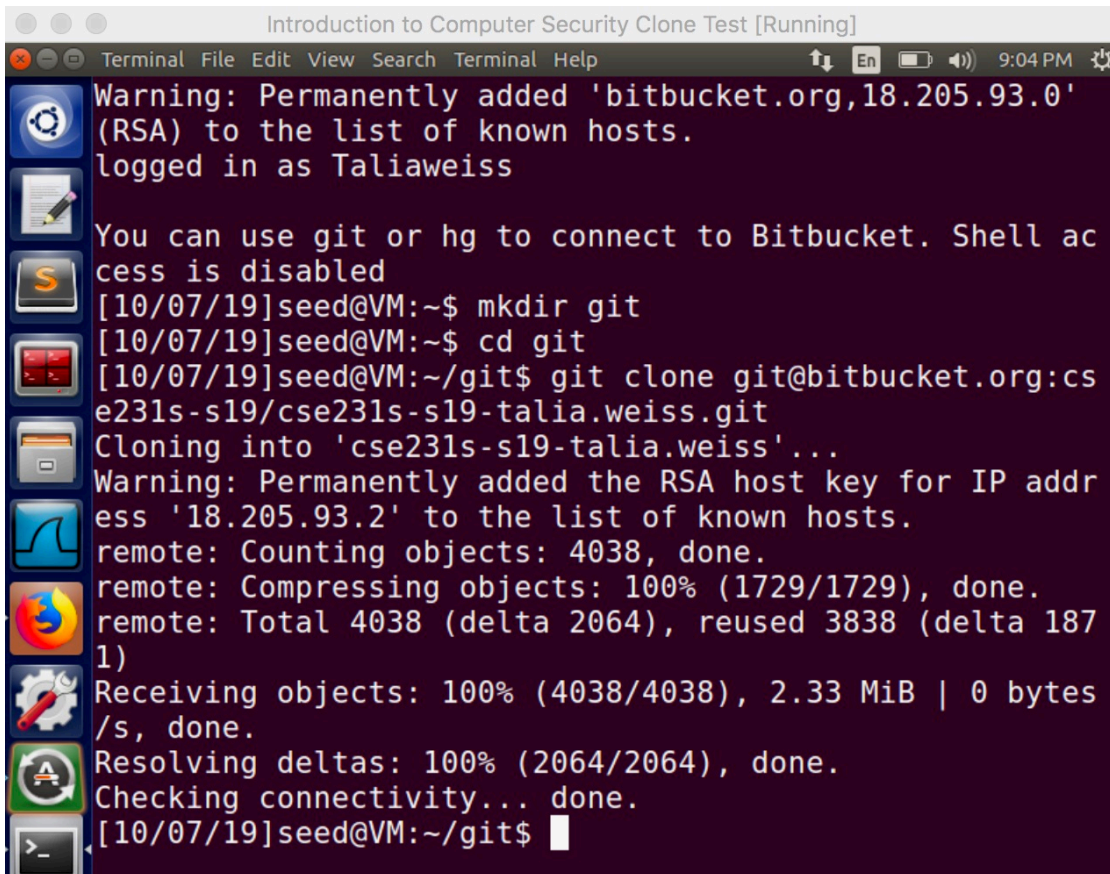
Question 5:

Password File	Average Password Length (characters)	Incremental Method	Wordlist: Default, Rules: None	Wordlist: Large, Rules: None	Wordlist: Default, Rules: All	Wordlist: Large, Rules: All
(1) - No change	6.514	345	368	370	370	370
(2) - Large number of replacements	6.515	28	32	42	51	70
(3) - Four random digits added	10.514	0	1	2	316	316
(4) - Combined Words	21.070	0	1	1	1	1
(5) - Combined Words with multiple replacements	21.0	0	1	1	1	1

Question 6a: Typically the larger file cracked more passwords, and using all rules cracked more passwords than not using any rules. Also, using a wordlist cracked more than incremental method (probably due to the time constraint).

Question 6b: The results for (4) and (5) were the same, our guess is that it is because the password length was on average the same.

Topic 2



```
Introduction to Computer Security Clone Test [Running]
Terminal File Edit View Search Terminal Help 9:04 PM
Warning: Permanently added 'bitbucket.org,18.205.93.0'
(RSA) to the list of known hosts.
logged in as Taliaweiss

You can use git or hg to connect to Bitbucket. Shell ac
cess is disabled
[10/07/19]seed@VM:~$ mkdir git
[10/07/19]seed@VM:~$ cd git
[10/07/19]seed@VM:~/git$ git clone git@bitbucket.org:cs
e231s-s19/cse231s-s19-talia.weiss.git
Cloning into 'cse231s-s19-talia.weiss'...
Warning: Permanently added the RSA host key for IP addr
ess '18.205.93.2' to the list of known hosts.
remote: Counting objects: 4038, done.
remote: Compressing objects: 100% (1729/1729), done.
remote: Total 4038 (delta 2064), reused 3838 (delta 187
1)
Receiving objects: 100% (4038/4038), 2.33 MiB | 0 bytes
/s, done.
Resolving deltas: 100% (2064/2064), done.
Checking connectivity... done.
[10/07/19]seed@VM:~/git$
```

1. Name of file: id_rsa
2. Name of file: id_rsa.pub
3. Local machine checks the server's validity via CA (needs to authenticate server) —> if valid, machine sends request to server —> server validates machine's digital signature by using CA's public key --> if passes, server sends back answer to command (such as the requested repository)