Cracking User Passwords

Project Report – 2008

By: Stephen Jones

Supervisor: Mark Ryan

Degree: MSc Computer Security

School of Computer Science
University of Birmingham

Abstract
Passwords have become the most common way for users to authenticate themselves and log in to systems. As more systems are using passwords, it is important that users have strong ones, but they also need to be able to remember them without resorting to bad habits such as writing them down. Most password policies suggest using upper and lower case letters, symbols and numbers in passwords. This is generally more secure than just a word, but may not be as secure as first thought. This paper looks at the ways people generate passwords and a program is created which uses similar methods to attempt to crack user passwords. The results showed that taking a word and inserting numbers or symbols or changing letters for numbers or symbols, create passwords which are straight forward to crack. Methods of creating strong, memorable passwords are then suggested and tested for both memorability and security.

Keywords
Password, Cracking, SHA-1 hashes

Acknowledgements
Help and project guidance: Mark Ryan
Proof Reading: Kelly Ralph, Anne Jones

Contents

Password Cracking

1. Introduction

Passwords have become a necessary part of everyday life; they are used for many things such as logging into a works network, checking email and internet banking. They have become the most common way to secure websites and authenticate the users accessing them. This means that passwords must be strong but also memorable for the user as they may have numerous passwords on different systems. This project will attempt to answer the question of how secure most people's passwords are and investigate into whether there are ways people can generate secure and memorable passwords.

People are encouraged to choose secure passwords containing numbers, letters and other characters. A quick search on the internet on for the phrase "choosing a secure password" finds millions of results which give ideas and guidelines on what makes a secure password.

One example is the Birmingham University Password Policy, the guidelines are as follows:

- The password must be at least eight characters long
- The password must not be a dictionary word (in any language) or names, places, etc
- The password must contain characters from at least three of the following categories:
  - English uppercase characters (A-Z)
  - English lowercase characters (a-z)
  - Numbers 0 - 9
  - Non-alphanumeric (e.g. !, $, #, %)
- The password does not contain a substring of the user's username of three or more characters in length.
- Easy for you to remember
- Difficult for others to guess

Is this actually more secure than not having any restrictions on passwords?

It is definitely secure if the user creates a truly random string of upper and lower case letters, numbers and symbols. The only way a password generated in this manner can be cracked is using a brute force attack. Taking the 96 possible characters on a keyboard, there are 7,213,895,789,838,336 possible 8 character passwords, if you can generate 1,000 every second, it would take about 228,751 years to generate them all to try to crack the passwords. This sounds like a good way to generate passwords and it would be except that a truly random string of upper and lower case letters, numbers and symbols is hard to remember. Also having an increasingly large number of passwords which need remembering is not the approach taken by many people.

Another method which could be used to generate more memorable passwords would be to start with a word which has a meaning to the user. They could then change it in some way adding capitals, numbers and symbols to ensure it meets the requirements in the password policy. For example the user chooses the word "password" and changes an s for $, a for @, o for 0 and capitalises the d, ending up with "p@s$w0rD". This will be harder to crack than just "password" on its own, but how much difference does it make? Is it possible to create a cracking program which uses these kinds of rules in order to crack passwords?

In addition to the physical restrictions such as the ones in the password policy above, passwords must be memorable and easily typed. This is important because if passwords are not memorable, users are likely to write them down, or use the same one for everything, so that they only have one to remember. If it is hard for a user to type a password there is more chance of somebody looking over their shoulder and being able to see which characters they enter. If these two features are not taken into account then the password is still vulnerable even if it would otherwise be secure.

There has always been a trade off between a secure password and a memorable one, but does this have to be the case?

2. Background

2.1 Cryptographic Hash Functions

2.1.1 Properties

A hash function is a one way cryptographic function which takes an input of any length and produces a fixed length output. The output is an encrypted version of the input and is called the hash or message digest. The hash guarantees the contents of the message without revealing what the message is.

There are three main properties of cryptographic hash functions, they are the following:

- **Preimage resistant:** for any hash *h* it is hard to find a message *m* such that *h = hash(m)*
- **Second preimage resistant:** for any message *m1*, it is hard to find a message *m2*, such that *hash(m1) = hash(m2)* and *m1≠m2*
- **Collision-resistant:** it is hard to find two different messages *m1* and *m2* where *hash(m1) = hash(m2)*

The following scenarios help to explain these rules, and show how these lead to a cryptographically secure function.

The first scenario relates to password storage; a computer system stores user passwords in a text file on disk. The passwords are stored as a hashed version of the password so if a hacker manages to get access to the file they cannot easily tell what the passwords are (preimage resistant). If a hacker is trying to gain access to the system using a specific username they will find it hard to find a password which matches the stored hash and isn't the users actual password (second preimage resistant).

The second scenario is to do with signing messages; an employee wants their manager to digitally sign a letter for them, for example a reference. For efficiency this is done by taking the hash of the message and the manager signing that. The employee creates two letters, a truthful one which the manager is happy to sign, and a second one which the employee would prefer as a reference but the manager is not willing to sign. It should be hard for the employee to create two such letters which have the same hash (collision resistant). If it were possible for the employee to create two messages with the same hash, they could get the manager to sign the truthful one and add the signature to the false one because as the messages have the same hash the signature would still be valid.

2.1.2 Uses

Cryptographic hash functions have a number of uses in the computing world; a few examples are as follows:

- **Simple commitment schemes** – This use allows one person to commit to the answer to a question without revealing what their answer was. For example in an electronic coin toss, Alice decides her answer (heads or tails) creates the hash of this answer concatenated with a random number and sends it to Bob. Bob tosses the coin and sends the result to Alice. Alice can then reveal her answer and the random number to Bob who can concatenate them together and run the hash function on the result to prove Alice is telling the truth.

- **Message integrity checks** – When a file is downloaded or message received by Alice and she wants to check that it has not been changed since Bob sent it, Bob can create the hash of the original and Alice can create the hash of the received message. These are then compared and if they are different then a change has occurred.
- **Password storage** – Passwords are not stored in plain text to ensure that anyone getting access to the password file cannot find all the user's passwords. They are usually stored as the hash of the user's passwords. When a user inputs their password for authentication, the hash of the password is created and compared to the stored value; if they match the user is granted access.

2.1.3 SHA-1

The hashing function used in this project will be SHA-1. It is a well know function originally designed by the National Security Agency in America. SHA stands for Secure Hash Algorithm and the 1 is because it was the first in a family of five hash functions.

SHA-1 is a cryptographic hash function which has a message digest length of 160 bits and can take an input up to a maximum size of 2^{64}-1 bits.

It works by processing the input over 4 rounds; each round has 20 computational steps which include some non-linear operations on the inputs. This adds to the security of the algorithm.

These rounds help ensure the security of the function because a small change in the input causes an avalanche effect resulting in a large change in the output. For example:
The SHA-1 encryption of the word "password" is 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8.
The SHA-1 encryption of the word "Password" is 8be3c943b1609fffbfc51aad666d0a04adf83c9d.

SHA-1 is currently used in many protocols and applications including TLS, SSL, PGP, S/MIME and IPSec.

2.2 JohnTheRipper

JohnTheRipper is an open source password cracking program which will work on a number of different hash types. It is a popular piece of software as it is easy to use and it is available on a variety of different platforms.

It has a number of cracking modes, from a simple dictionary attack (taking words from a word list encrypting them and comparing them with the password) to brute force attacks (attempting to encrypt every possible combination of letters, numbers and symbols then comparing them to the password). Another mode, called single crack mode takes any information available about the user (username, home directory name) and uses a set of mangling rules to change them in an attempt to find the password. The most powerful mode available to JohnTheRipper is called incremental mode, it uses probabilities of letter positions in words to try to reduce the search space of the brute force attack.

It does not work on SHA-1 by default, but there are a number of different add-ons for it which allows it to crack SHA-1 hashes. When used for this project one of these add-ons will need to be used.

3. Objectives

3.1 Required Objectives

- Do research into password creation to find out the ways people generate passwords
- Generate rules for creating passwords based on the results of the research
- Implement a password cracking program
- Use the program to crack as many passwords as possible

3.2 Optional Objectives

- Generate a schema to create secure, memorable passwords
- Compare the program with JohnTheRipper for speed and efficiency

3.3 Feasibility

To ensure that it will be feasible to try to generate SHA-1 hashes of a large number of words and compare them to another set of hashes stored in a file, I will first write a quick program which generates a large number of hashes and find out how many hashes it can generate in a given time.

I wrote a program *feasibility.c* which generates the hash of a string. It creates strings from numbers starting at 0 and incrementing it each time a new hash is created. Using a timer in the program I could specify how long the program should run for, and print out the number of hashes created in that time. I can then run it for different periods of time and work out how many hashes I am likely to be able to generate in a given time. I ran it five times for each of 1 second, 10 seconds, 1 minute, 10 minutes and 1 hour and calculated the average for each. The results are shown in Table 1.

Time	Number Tested					Average
	1st Test	2nd Test	3rd Test	4th Test	5th Test	
1 Second	95,064	99,701	100,129	104,809	104,789	100,898
10 Seconds	999,368	1,041,512	1,018,993	1,009,229	1,031,926	1,020,206
1 Minute	6,027,917	6,056,737	6,263,145	6,296,068	6,257,052	6,180,184
10 Minutes	60,188,165	60,173,668	61,915,561	61,614,242	58,684,070	60,515,141
1 Hour	370,451,203	360,053,709	360,255,803	353,925,810	356,485,115	360,234,328

Predicted	Based on resutls for					Average
	1 Second	10 Seconds	1 Minute	10 Minutes	1 Hour	
10 Hours	3,632,342,400	3,672,740,160	3,708,110,280	3,630,908,472	3,602,343,280	3,649,288,918
1 Day	8,717,621,760	8,814,576,384	8,899,464,672	8,714,180,333	8,645,623,872	8,758,293,404
1 Week	61,023,352,320	61,702,034,688	62,296,252,704	60,999,262,330	60,519,367,104	61,308,053,829

Table 1 – Results from Feasibility Test

Table 2 - Prediction of the Number of Hashes in Given Times

Using these results I can predict how many hashes I can generate in a given time. These are shown in Table 2. The prediction shows that in a week if the program runs non-stop it should be able to generate about 61 billion hashes. This result is based on a single machine, so if the program runs on 5 machines each generating a different set of hashes it would be able to generate about 300 billion hashes in a week.

Although this is nowhere near the 1.5×10^{48} possible hashes there are for SHA-1, or even the 7.2×10^{15} possible 8 character passwords. I feel this is quite a large amount and should be enough to crack some user passwords if they are chosen carefully using the results from the research.

4. Method

4.1 Work Plan

This project will be implemented using the following plan. The Gantt chart (Chart 1) shows the estimated timetable for each part of the project.

4.1.1 Research into Password Creation

In order to find the best ways to try to crack people's passwords, I must first try and understand how different people generate their passwords and find common methods which can be used to generate a set of rules. To do this I will first research the ways different companies and websites restrict people when letting them choose their passwords, and then the ways people might generate them. Once I have some ideas of the different methods used, I will create a questionnaire to find the most popular methods of password generation. I will also look at the properties of passwords such as length and ways people remember them.

4.1.2 Generate rules based on the Research

Once I have collected enough data from the questionnaires I will analyse it to find the most common ways of creating passwords. I will then try to generate some rules which people may follow in order to generate their passwords. These rules will form the basis of the program.

4.1.3 Implementation of a Password Cracking Program

The main part of the project is to implement a program which can take a list of SHA-1 password hashes and attempt to crack as many as possible. As mentioned earlier every password is crackable by a brute force attack but this is not feasible given a sufficiently long password. Therefore, to try to reduce the search space for the passwords, I will implement the program using the rules generated from the research initially implementing the most common methods used by people to generate passwords and working towards the less popular. The program will create a possible password, calculate its hash, compare it to the list of hashes given to me and alert the user if a match is found.

4.1.4 Crack as many passwords as possible using the program

The program will then be run on the list of password hashes. It should output any matches found along with details of the number of hashes tried and time taken to crack the password. It should also indicate which rules were being used at the time the password was cracked.

4.1.5 Generate a Schema to Create Secure, Memorable Passwords

By looking at any passwords the program is able to crack and the ones it cannot crack, I will attempt to find a way of generating secure passwords. Once I have come up with a way, I will modify it if necessary so that users can use it to generate passwords which are memorable as well as secure.

4.1.6 Comparison with JohnTheRipper

If time allows, I will compare the program created with JohnTheRipper to see which is more efficient at cracking passwords, and the strengths of the passwords each can crack. I will also test my "secure" passwords to see if JohnTheRipper can crack them.

4.2 Gantt Chart

Chart 1 shows the estimated timetable for the project.

	June		July				August					September		
	14 - 20	21 - 27	28 - 4	5 - 11	12 - 18	19 - 25	26 - 1	2 - 8	9 - 15	16 - 22	23 - 29	30 - 5	6 - 12	13 - 16
Project Proposal	■													
Research / Questionnaires		■	■											
Generate Rules				■										
Write Program					■	■	■	■						
Run Program									■					
Generate Strong Passwords and Test										■	■			
Create guidelines for strong/memorable passwords										■	■			
Time for overrun or additional features												■	■	
Create Presentation												■	■	
Write report	■	■	■	■	■	■	■	■	■	■	■	■	■	■
Finalise report													■	■

Chart 1- Gantt Chart

4.3 Research

4.3.1 Password Security Research

In order to be able to write a program to crack passwords, I first needed to understand how people might create their passwords. To do this I will have to do research into suggested methods for password creation and then create a questionnaire asking people various questions about the construction of their passwords.

I first looked at a number of Journals and websites; Furnell (2007) has done research into Website Password Practices which gives an example of ten different popular internet sites and shows the guidelines and restrictions they have for users creating passwords. It covers things like length of passwords, if they allow usernames or surnames and if the site forces users to have a mix of letters, numbers and symbols in the password. Furnell also includes a list of guidelines on creating a good password and things to avoid.

For this background research, I will also be looking on the internet for websites offering advice and guidelines to people on what makes a good password and things to avoid when creating one.

One such website which provided a lot of information in this area is Strong Passwords and Password Security by Microsoft Security. This site provides information on how to create a good password as well as some things to avoid. As this project is trying to crack real passwords, the creators may or may not have read sites like this, so I will have to consider the ways to create secure passwords as well as ways advised against.

The following ideas based on the list given by Microsoft are common to the advice given by other journals and websites:

- **Make it lengthy** – The longer the password the more secure it is. I will need to find the average length of passwords for people.
- **Combine numbers, letters and symbols** – Having a larger character set increases the security of the password.
- **Use words and phrases easy for you to remember but difficult for others to guess** – Finding the common basis people use for passwords will be important.

- **Avoid sequences or repeated characters** – Sequences such as 1234 or qwerty are common for passwords, so these should be avoided when creating a password. Therefore I will ensure that these are checked for when trying to crack passwords.
- **Avoid using only look alike substitution of numbers and symbols** – if people substitute letters for numbers or symbols I will need to work out what letters are commonly substituted and what they are substituted with.
- **Avoid your login name** – Personal information about a person's password may make it easier to crack, but in this project I will not know anything about the people who submitted the passwords.
- **Avoid dictionary words in any language** – A standard word with nothing changed on it is insecure. I will only be concentrating on English words and names because including other languages will not be feasible in the time allowed for this project.

As this list is common to most websites, most people will probably have seen something like this at some time and base their passwords on these kinds of rules. Therefore I can use this as a basis for my questionnaires.

4.3.2 Password Memorability Research

The security of a password is not the only important thing to consider when choosing a password; it also has to be memorable for the user. If this is not the case, the user may be tempted to write it down, use it for all their passwords or not change it regularly.

Vu et. Al (2007) has done research into the area of password memorability and performed a number of experiments. The first was testing if a password created using a word as a basis of the password then manipulating it in some way is memorable for the user. The second was testing if a password created using the first letters of a phrase then manipulating it was memorable to the user. The results showed that a manipulated word was slightly more memorable for the user, than the letters from a sentence. The paper also showed that if a user is asked to remember their password soon after they created it then they are more likely to be able to remember it in the long term.

4.4 Questionnaires

Using the information gathered from my research I went on to ask people how they generated their passwords. To do this I created a simple web questionnaire which emailed the results to me (See Appendix A) and invited friends on Facebook to fill in, I also emailed a link to the Computer Science MSc mailing list at Birmingham University. Using friends on Facebook I got a large selection of people from different age groups and different backgrounds (i.e. people who are very technical and would be aware of the security implications of passwords and those who may not be). Using the mailing list should ensure I get a selection of people with computer knowledge who should know the reasons for strong passwords and therefore create them. The questionnaire can be found at http://www.steve-jones.org.uk/questionnaire.html.

The following is the list of questions I asked people:

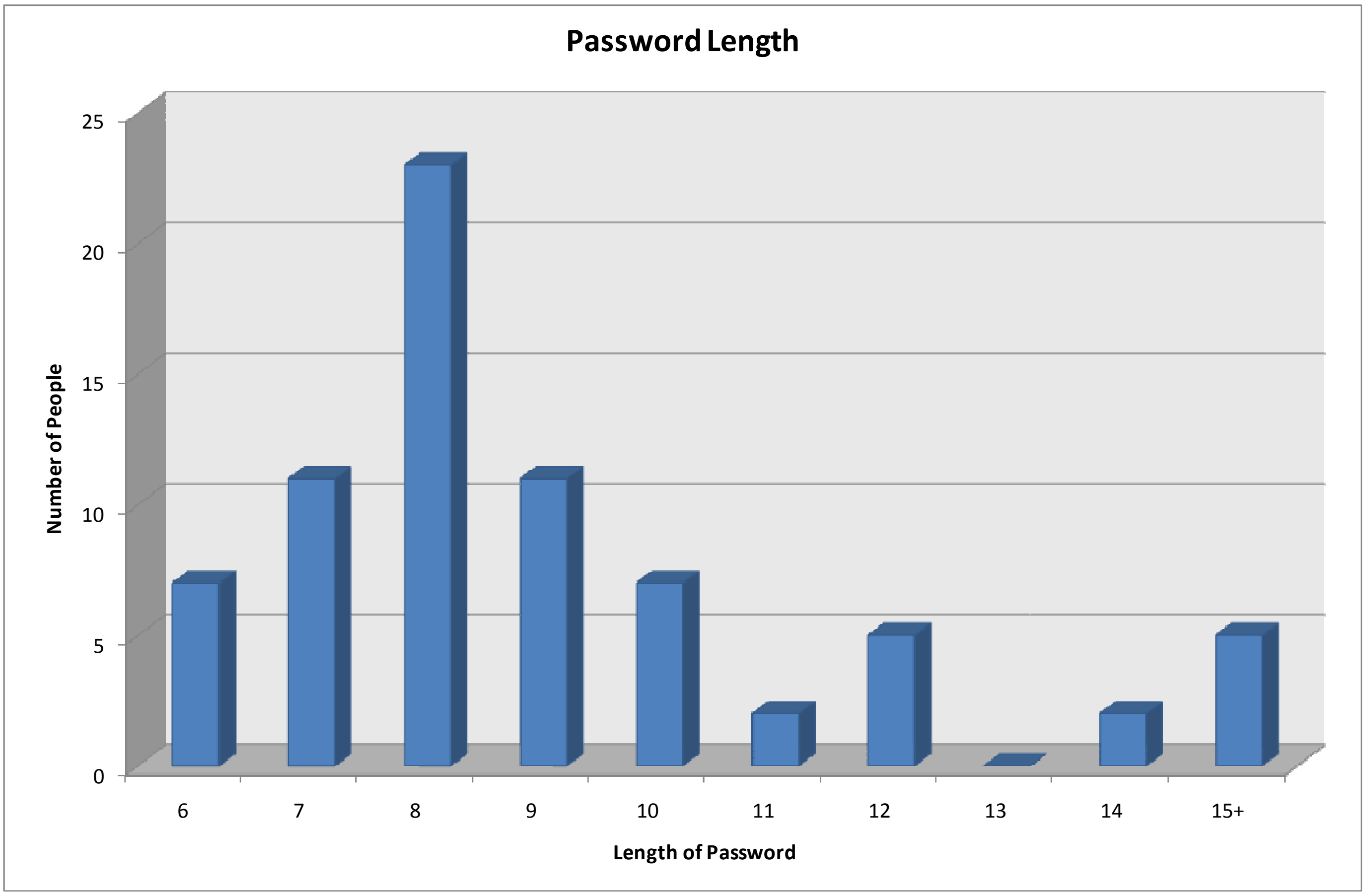
- How long are your passwords usually?
- Which of the following do you use as a basis for your passwords?
    - Random Word
    - A Name (Yours, Family member, Place, etc.)
    - Number plate
    - First letters from a phrase
    - Phone Number
    - Random string of letters/numbers/characters
    - Other

- Which of the following do you use to make up your passwords?
  - Nothing just the word eg: cat
  - Add a number/symbol to the end eg: cat1
  - Add a number/symbol to the beginning eg: @cat
  - Repeat a word eg: catcat
  - Mirror a word eg: cattac
  - Reverse a word eg: tac
  - Capitalise letters eg: caT
  - Change letters for numbers eg: c8t
  - Change letters for symbols eg: ca]
  - Insert numbers eg: c5at
  - Insert symbols eg: ca%t
  - Use leet speak eg: c/-\t
  - Other
- If you change letters for numbers or characters, which letters do you change and what characters do you change them to?
- How do you remember your passwords?
  - It's a simple password
  - It's a short password
  - It's a hard password but I still manage to remember it
  - Write it down
  - Use the same password in different places so only need to remember one
  - It's been the same for a long time
  - Other
- Are there any other ways you have for creating passwords, or any other comments?

4.4.1 Questionnaire Results

The results from any relevant comments have been incorporated into the numerical data. Graphs 1-5 and Table 3 show the results.

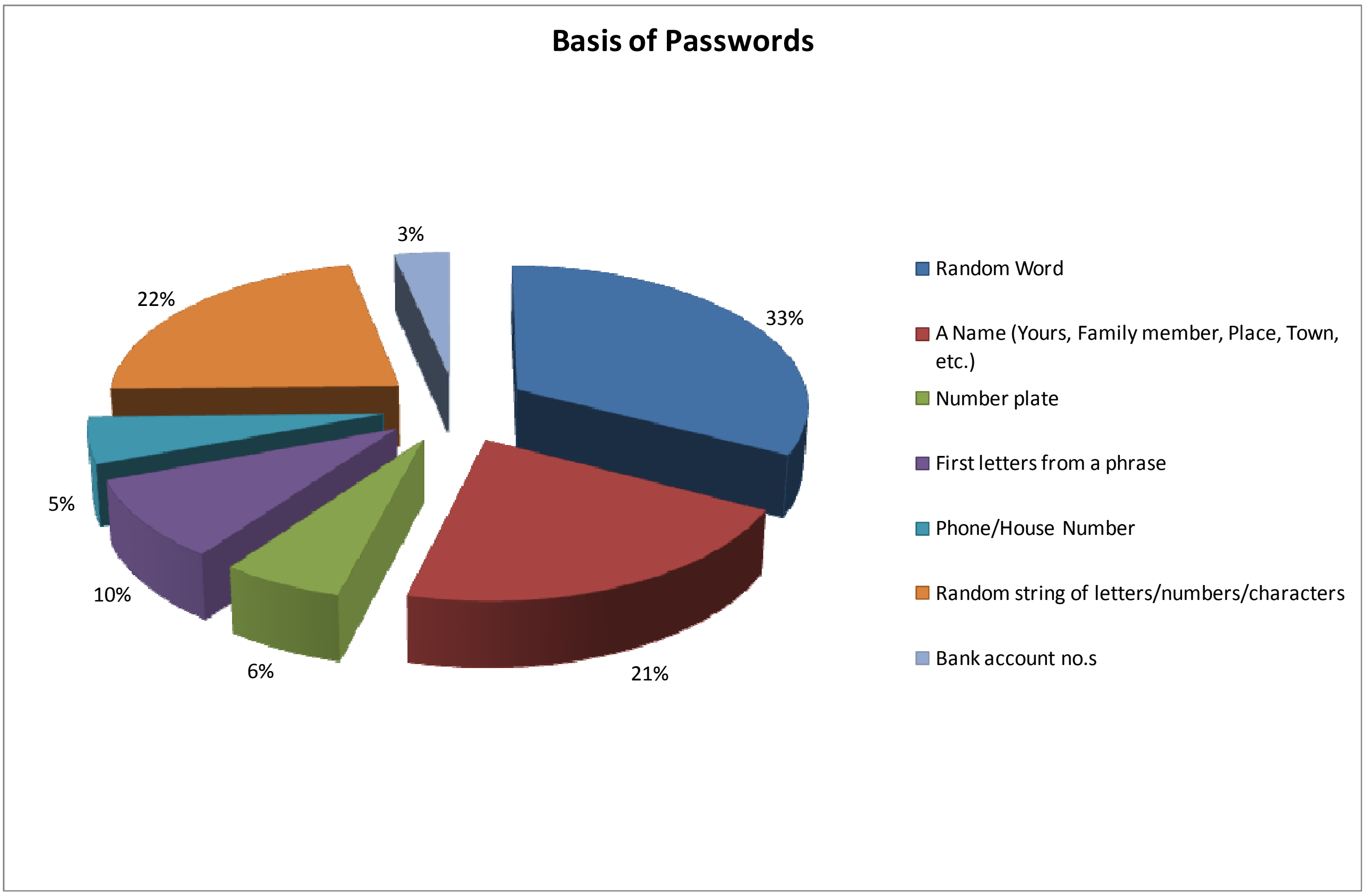
4.4.1.1 How Long Are Your Passwords Usually?



Graph 1 - Password Length Results

Graph 1 show's that out of the people who answered this question over a third gave the answer of 8 characters. As this is by far the most common length I will concentrate mainly on this length when creating passwords to test. Passwords with 7 or 9 characters were the second most common lengths and 6 or 10 characters third, so I will also try passwords of these lengths. As nobody gave an answer of less than 6 characters all the passwords I generate can be between 6 and 10 characters long. By looking at this range of lengths I would cover over 80% of the passwords generated by the people who took the questionnaire. Lengths of passwords greater than 10 characters can be looked at if there is enough time.

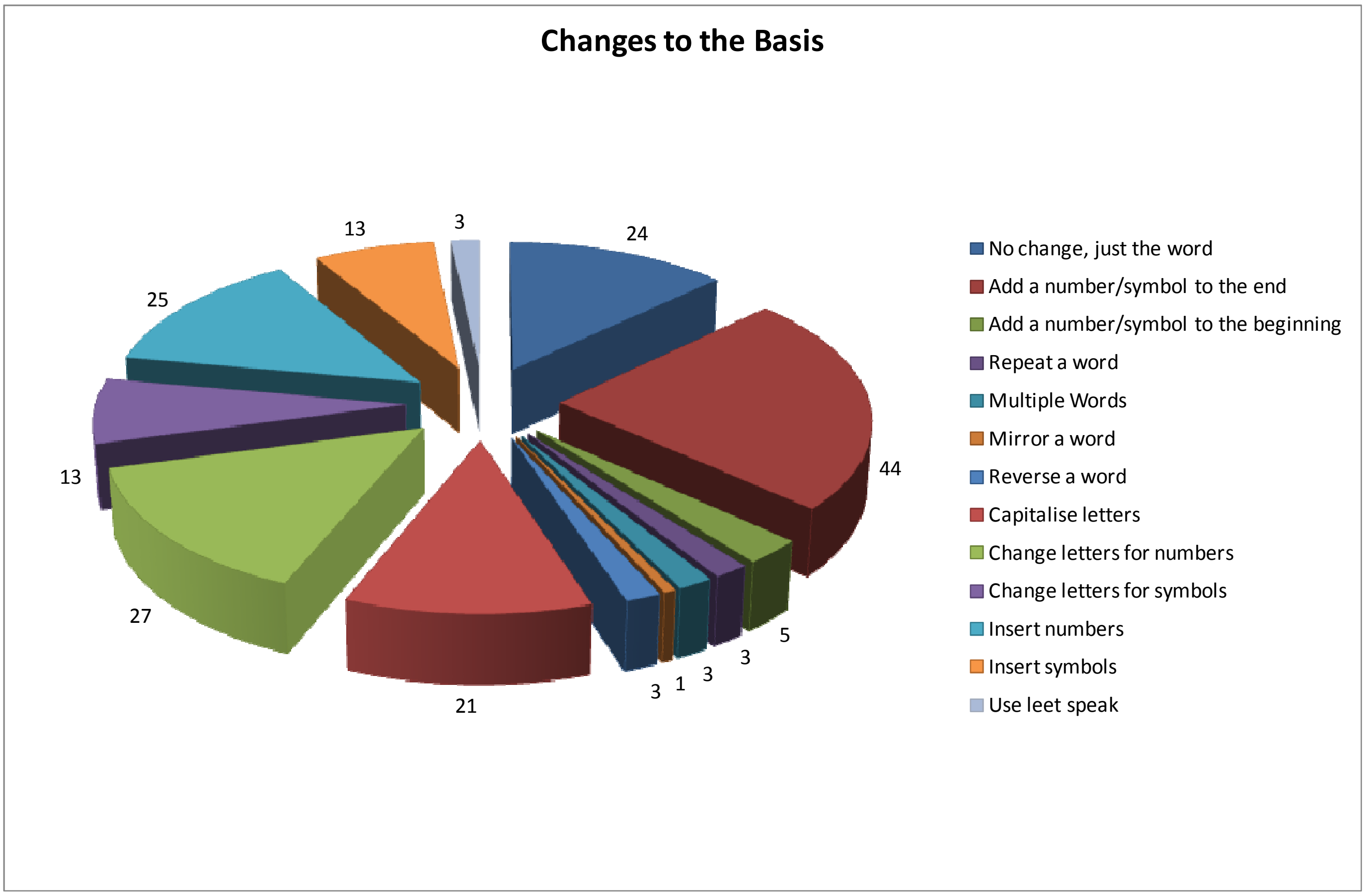
4.4.1.2 Which of The Following Do You Use As A Basis For Your Passwords?



Graph 2 - Basis of Passwords Results

Graph 2 shows that the most common basis for a password is a random word. This can be grouped with the second most popular response which was a name. Passwords using words or names as a basis can easily be generated using a word list or dictionary. The next most common basis was a random string of letters, numbers and symbols; this will be a very hard thing to generate without resorting to a brute force attack on the passwords. It is not really feasible to run a complete brute force attack in the time available but I will add this feature if time allows. Taking the first letter of a phrase as a basis would not be a very easy method to try to crack as there are so many possible phrases. But as only 10% of people do this I feel it is acceptable to leave this and the other methods with even less people out of this project.

4.4.1.3 Which of The Following Do You Use To Make Up Your Passwords?



Graph 3 - Changes to the Basis Results

Even though the choice of no change to the word was not the most popular choice this will be the first thing to test as it is the quickest and simplest to check once I have a word list generated.

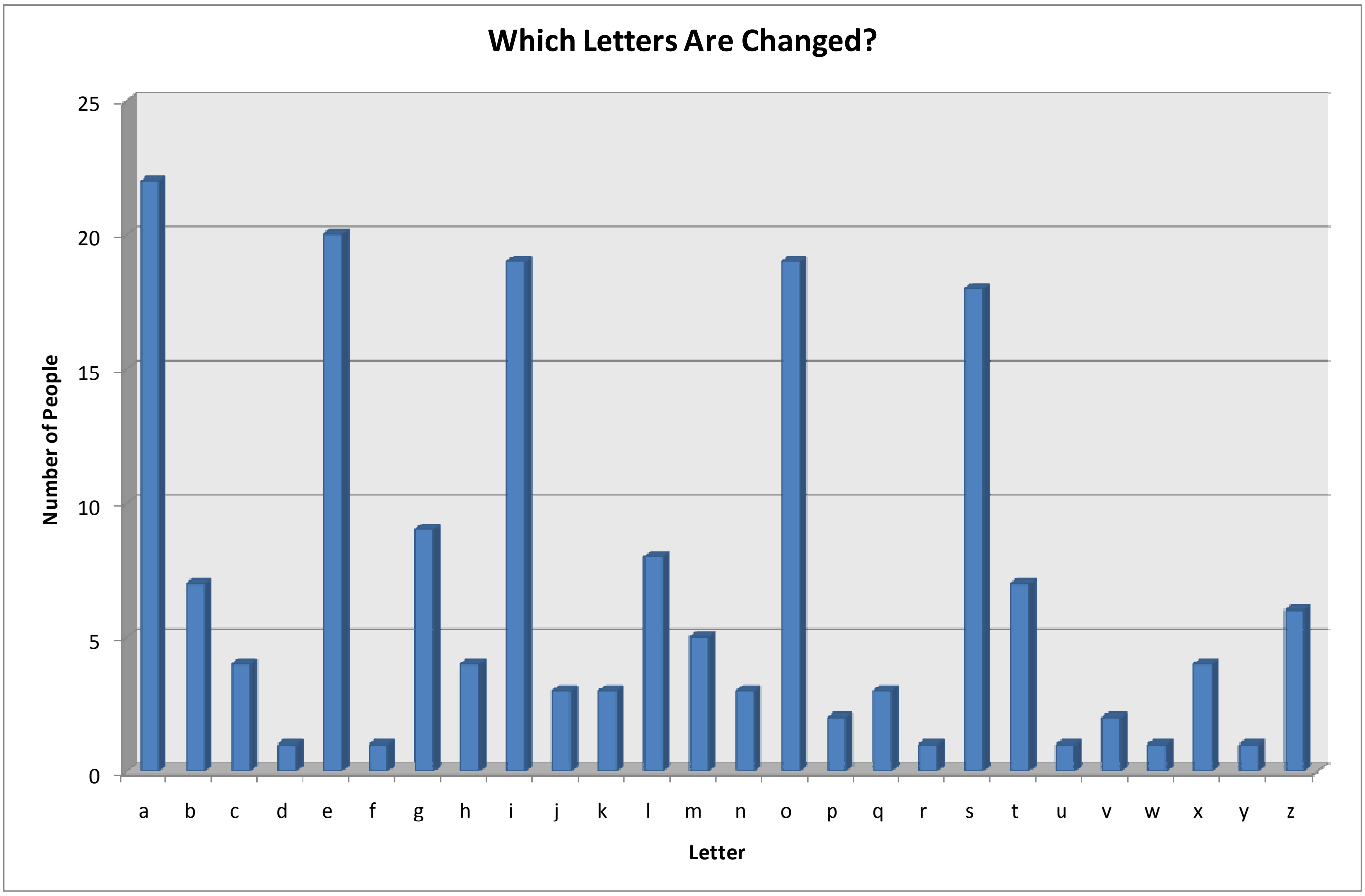
The most common change to the basis is adding a number or symbol to the end. Making an assumption that a number added to the end is unlikely to be longer than 4 digits because it would be hard to remember. I will concatenate each number between 0 and 9999 onto each word in turn. I will also try adding one or two symbols onto each word.

The next most common is changing letters for numbers. This can be grouped with slightly less popular results; changing letters for symbols and capitalising letters in the same piece of code and run together. The results from the next question will be used to see which letters are changed for which numbers or symbols.

The next most common is inserting numbers; this can be implemented in the same code as inserting symbols. I will test words with up to 3 numbers or symbols inserted.

Very few people use the other options, therefore they will not be implemented unless time allows.

4.4.1.4 If You Change Letters For Numbers Or Characters, Which Do You Change and What Characters Do You Change Them To?



Graph 4 – Which Letters Are Changed Results

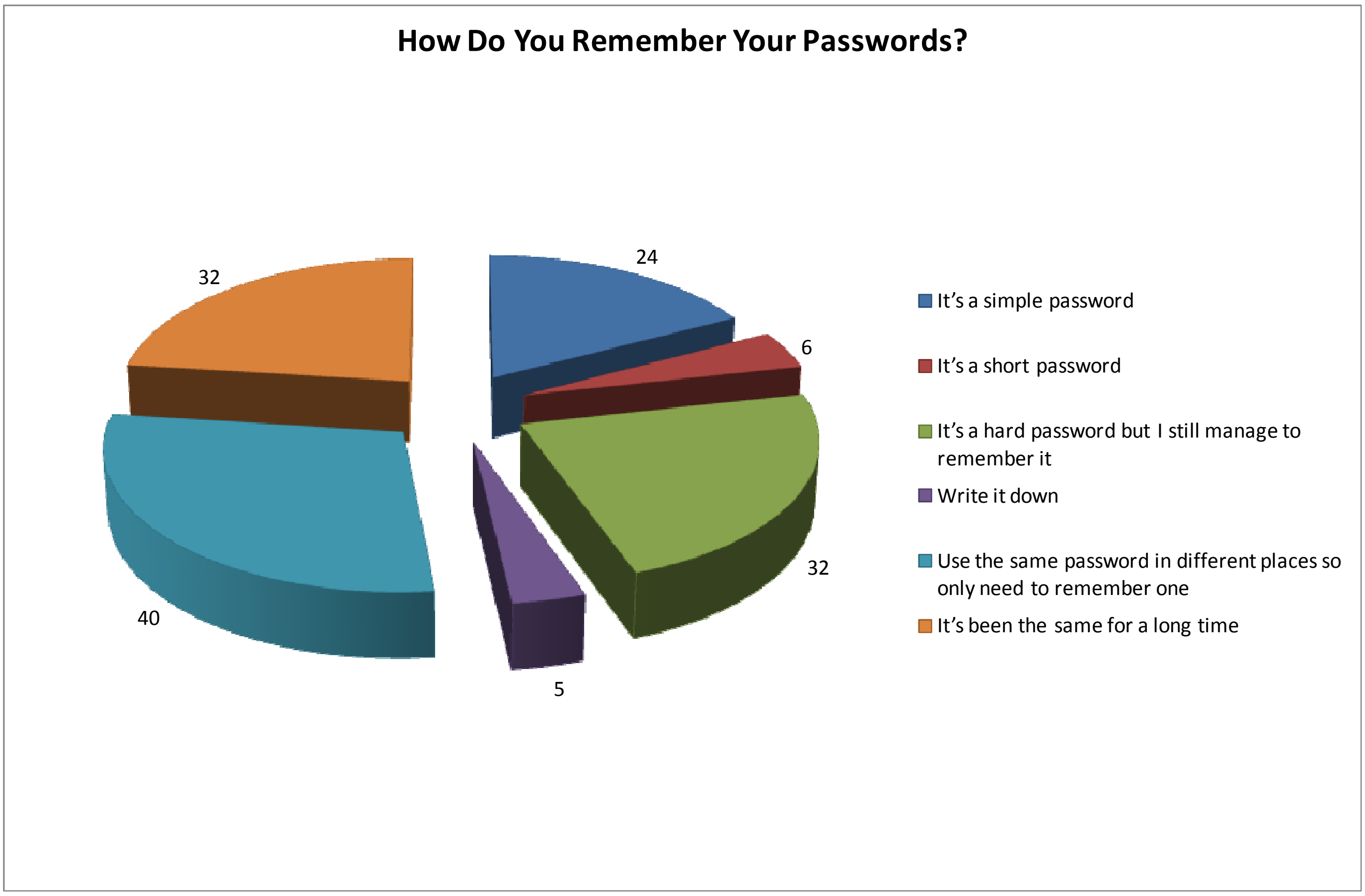
Graph 4 shows how many people change each letter, even though some are more popular than others, this is not very important for my program, as I am going to change every letter in each word. Table 3 which shows what each letter is changed to is a lot more important, as it is infeasible to try changing every letter for every possible symbol, so I will attempt to change every letter to the symbols given by people for changing it to.

This graph would have more relevance if the project took into account which letters were changed and could prioritize them accordingly when changing them. I feel this is beyond the scope of this project in the time given but could be attempted as an extension to the project.

Original Letter	Changed For																													Total
	0	1	2	3	4	5	6	7	8	9	#	¬	!	£	$	^	&	*	(	+	@	<	|	g	m	n	s	y	z	
a					12					1							1			1	7									22
b						1	2		4																					7
c				1															2					1						4
d								1																						1
e				14			1					1		4																20
f				1																										1
g							1			8																				9
h					2						2																			4
i		14											4															1		19
j		2						1																						3
k		1					1															1								3
l		5											2										1							8
m				3																1						1				5
n			1													1									1					3
o	19																													19
p							1			1																				2
q										3																				3
r								1																						1
s						10									6									1					1	18
t		1						5												1										7
u						1																								1
v						1										1														2
w					1																									1
x										1								3												4
y										1																				1
z			3			1														1							1			6

Table 3 - What are they changed to Results

4.4.1.5 How Do You Remember Your Passwords?



Graph 5 - How Do You Remember Your Passwords Results

As this question is related to the section on generating memorable passwords rather than on the ways to crack them, it will not be used in the password cracking section of this project. This graph will be looked at in more detail in the section on creating a memorable password.

4.4.2 Conclusion

From the questionnaire results I have created the following implementation stages for the program, each one will focus on a different method for trying to crack the password hashes.

Stage of Implementation	Basis of Password	Change to Basis
1st Stage	Random Word	Nothing
2nd Stage	Random Word	Numbers 0-9999 added
3rd Stage	Random Word	1 and 2 symbols added
4th Stage	Random Word	Change letters for capitals/numbers/symbols
5th Stage	Random Word	Insert numbers/symbols
6th Stage	None	Brute Force

Table 4 - Implementation Stages

4.5 Implementation

The program will be implemented in a number of stages each adding additional functionality to the program in the form of other methods used to crack the passwords. The stages implemented will be as follows:

4.5.1 Stage 1: Words on Their Own

4.5.1.1 Plan

The first stage of the implementation will be to write a program which can encrypt strings using the SHA-1 algorithm. Once I have this I can extend it to encrypt words from a word list file. These can then be compared with the password hashes given to me. This will involve reading in two files, creating the SHA-1 hash of each word in the word list file and comparing it to every entry in the password hash file. Any matches indicate a cracked password. When one is found it will be printed out along with the plaintext of the password, number of passwords tried before the match was found and how long it took to crack.

4.5.1.2 Structure

To implement this stage of the project, I first wrote a function which would take a buffer containing a string and output the SHA-1 hash of the word. To do this I used the libgcrypt library and checked the output with the standard Linux SHA-1 program to ensure this was correct. I then created a function which would read in all the password hashes to be cracked from a given file, and create a list out of them. The program then read each word in turn from the word list file, encrypted it and used a third function to compare the hashes in the list with this hash. If a match is found, the user is alerted to this fact.

4.5.2 Stage 2: Words Followed by Numbers

4.5.2.1 Plan

Once I had a program which could encrypt a list of words and compare them to the list of hashes, I moved on to the next mode of cracking which involve adding a number onto the end of each word. The numbers added would be up to 4 digits long. By taking the functions used in the first stage I can extend them to add a number onto the end of each word before it is encrypted and then compared. I want to ensure every possible 4 digit number is added so need to loop through 1 digit numbers, then 2 digit numbers etc separately. This will ensure I cover all possible combinations of numbers i.e. 0-9, 00-99, 000-999, 0000-9999 (11110 in total).

4.5.2.2 Structure

As my wordlist contained over 1,000,000 words, and I am limited in the time I have to crack the passwords I am concentrating on only 6-9 character passwords. I decided to split the word list into multiple word lists files each containing words of a specific length. To do this I created a shell script *CreateLetterList.sh* which would take a word list, sort it, remove duplicates and create the required word lists from it. I then changed the program so the user has to input which length of passwords they want to try to crack.

Once these changes were made I created a new section which would take a word list file and add a number onto the end of each word, encrypt the result and compare it to the list of hashes. This will need to take into account the length of the resulting passwords so an 8 character password could be made of an 8 character word, a 7 character word with 1 digit or symbol on the end or a 6 character word with a 2 digit number or 2 symbols on the end etc. To do this the program takes the password length specified by the user, subtracts one, uses each word in the word list containing words of this length and adds a single digit, 0-9, onto the end of each, encrypts and compares with the hash list. It then subtracts 2 from the password length and repeats the process with words of this length. This is repeated up to four times assuming the password length is long enough.

4.5.3 Stage 3: Words Followed by Symbols

4.5.3.1 Plan

This section will involve extending stage 2 to include adding one or two symbols onto the end of each word in turn. This will test every possible combination of one or two symbols which are which can be typed on a standard keyboard. Looking at a standard keyboard all 34 symbols will be included (i.e. ¬ ` ! ” £ $ % ^ & * ( ) _ + { } : @ ~ < > ? | - = [ ] ; ’ # , . / \ ). In the same way as the words followed by numbers section this will also need to take into account the length of the passwords.

4.5.3.2 Structure

Using the similar code to the previous section, the program subtracts 1 from the password length given by the user then using words of this length, loops through an array containing all the characters adding each one in turn onto the end of each word. It then subtracts 2 from the password length and repeats using this length word, but looping twice through the array so every combination of symbols is used. Once a new possible password is formed, it is encrypted and compared with the hash list.

4.5.4 Stage 4: Changing Letters for Capitals, Numbers and Symbols

4.5.4.1 Plan

This stage will allow checking of passwords in which letters in the word have been changed for capitals, numbers or symbols. Using the results from the questionnaires I can see which letters are commonly substituted by which other characters. This information will need to be input into the program and then each letter in turn will need to be replaced by each character it could be changed to.

4.5.4.2 Structure

To write this section I started by putting the questionnaire results showing which letters are changed for which characters into the text file LetterChangeList (See Appendix B for an example). This meant it could easily be changed without needing to recompile the program. I then created an array for each letter of the alphabet and what that letter could change to. Then taking each word from the word list the correct array for each letter in the word was selected. These arrays were then looped through in turn, forming every possible combination of letters from the arrays. Each entry is then encrypted and compared with the hashes in the hash list.

4.5.5 Stage 5: Inserting Numbers and Symbols

4.5.5.1 Plan

Stage 5 of the project will involve inserting numbers and symbols into each word in turn. I will test inserting up to three characters in each word. The inserted characters will be tested in each position in the word, and every possible combination of inserted characters will be tried. This stage also needs to take into account the length of the word used.

4.5.5.2 Structure

This section of code uses a similar array to the one used in Stage 3, but it also includes the numbers 0-9. In the same way as the previous stages, it selects the right length word list, then each character position within that word is changed for each character in the array. The rest of the letters in the word are moved to allow this letter to be inserted. This is repeated for inserting two and three characters in a similar way.

4.5.6 Stage 6: Brute Force Attack

4.5.6.1 Plan

The final section of the program will involve adding a brute force attack. This will test every possible combination of letters, numbers and symbols that can be typed on a standard keyboard.

4.5.6.2 Structure

This section of code is very similar to Stage 3 where letters are being changed for other letters. The only differences are; that there is no word list used and the original word consists of all a's (e.g. "aaaaaaaa" for an 8 character password) and the list of characters it can change to is an array containing every possible character.

4.6 Testing

To ensure the program is working correctly I will now run some testing on it. I will test each method individually and run a number of tests on each. Table 5 shows the tests and results. The Linux sha1 command was used to generate the hashes.

Test	Method	Hash File Contains Hash Of	Word list contains	Letter Change List Contains	Expected Result	Actual Result	Result
1	1	password	password	N/A	Cracked password	Cracked password	Pass
2	2	password0	password	N/A	Cracked password0	Cracked password0	Pass
3	2	password00	password	N/A	Cracked password00	Cracked password00	Pass
4	2	password000	password	N/A	Cracked password000	Cracked password000	Pass
5	2	password0000	password	N/A	Cracked password0000	Cracked password0000	Pass
6	2	password123	password	N/A	Cracked password123	Cracked password123	Pass
7	3	password*	password	N/A	Cracked password*	Cracked password*	Pass
8	3	password**	password	N/A	Cracked password**	Cracked password**	Pass
9	3	password()	password	N/A	Cracked password()	Cracked password()	Pass
10	4	password	password	N/A	Cracked password	Cracked password	Pass
11	4	Password	password	p:P	Cracked Password	Cracked Password	Pass
12	4	paSsword	password	s:S	Cracked paSsword	Cracked paSsword	Pass
13	4	p@ssword	password	a:@	Cracked p@ssword	Cracked p@ssword	Pass
14	4	p@sSw0Rd	password	a:@ o:0 r:R s:S	Cracked p@sSw0Rd	Cracked p@sSw0Rd	Pass
15	5	1password	password	N/A	Cracked 1password	Cracked 1password	Pass
16	5	pa1ssword	password	N/A	Cracked pa1ssword	Cracked pa1ssword	Pass
17	5	password1	password	N/A	Cracked password1	Cracked password1	Pass
18	5	1pa*ssword	password	N/A	Cracked 1pa*ssword	Cracked 1pa*ssword	Pass
19	5	pass*1word	password	N/A	Cracked pass*1word	Cracked pass*1word	Pass
20	5	password**	password	N/A	Cracked password**	Cracked password**	Pass
21	5	123password	password	N/A	Cracked 123password	Cracked 123password	Pass
22	5	p1a2s3sword	password	N/A	Cracked p1a2s3sword	Cracked p1a2s3sword	Pass
23	5	password123	password	N/A	Cracked password123	Cracked password123	Pass
24	6	cat	N/A	N/A	Cracked cat	Cracked cat	Pass
25	6	CAt	N/A	N/A	Cracked cAt	Cracked cAt	Pass
26	6	d0g	N/A	N/A	Cracked d0g	Cracked d0g	Pass
27	6	9!g	N/A	N/A	Cracked pig	Cracked pig	Pass
28	6	...	N/A	N/A	Cracked ...	Cracked ...	Pass

Table 5 - Test Results

As all tests were successful, I can assume the program is running correctly and can now run it on the full list of password hashes.

4.7 Running the Password Cracking

4.7.1 Running Order

4.7.1.1 8 Character Passwords

I first ran the program on 8 character passwords as this is the most common length, and the most common length of word in the word list is 8 characters, this means it is likely to take the longest. All modes will be run on 8 character passwords except for the brute force mode as that will take too long to run.

4.7.1.2 6, 7, 9 & 10 Character Passwords

The next most common lengths of passwords were 7 and 9 characters followed by 6 and 10 characters; therefore these will then be run. Again every mode except brute force will be run.

4.7.1.3 1-5 Character Passwords

As these are relatively short passwords brute force attacks are feasible, I estimate it would take about 30-40 hours for 5 characters. This will ensure that if any passwords of 1-5 characters have been chosen and contain only characters which can be typed on a standard keyboard they will be found. Brute force would not be feasible to run on longer passwords as the time increases 100 times with each character added, so would take 3000-4000 hours (125-167 days) to run with 6 characters.

4.7.1.4 11+ Character Passwords

Depending on how long the other password cracking runs take to complete will determine whether or not I try to crack any passwords greater than 11 characters long. If I do attempt this, I will try the quicker running methods first on each length, then move on to the longer ones.

5. Results

5.1 8 Character Passwords

Table 6 shows the results from running the program with 8 character passwords using Modes 1-5.

Mode	Hash	Password	Time	Hashes Tried
1	d50f3d3d525303997d705f86cd80182365f964ed	drowssap	2s	46,557
1	a2cd955d71f3606e0ce2fe17e68efc49a5c6567d	krabicka	3s	89,755
1	5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8	password	4s	122,467
2	2e0684e6d077d56c0c3452234fc5766d5a2bb0c5	images00	2m 30s	7,094,951
2	5cfa25f7aa00b49d02666d40f44cc882b789bbf2	melons88	3m 23s	8,975,639
2	767fb1139b5759376e0711185044fe25f20f849f	prolog68	4m 14s	10,817,819
2	3a357f71b2c87aa3291594140b1e8a08286845e8	purple74	4m 16s	10,898,825
2	c704f92d6a6aa809e16032b9aab796cacbe7300e	sparky01	4m 57s	12,382,152
2	38bdd82e40107a72c1d8264dbfac73ed4f235034	spooky01	4m 48s	12,422,552
2	edfb995ae41f655c7585a53214ec0dd28a21b175	vera8859	43m 2s	99,646,310
4	d50f3d3d525303997d705f86cd80182365f964ed	drowssap	6h 19m 43s	1,847,203,224
4	8821cbb005377c2c63cf76d53ade1695b355f5e0	green&34	9h 32m 8s	2,339,585,373
4	f92d5f4b642d3633d2092f5786ba7a90e20930fd	infr4R3d	12h 26m 58s	2,682,336,152
4	a2cd955d71f3606e0ce2fe17e68efc49a5c6567d	krabicka	22h 39m 2s	2,837,104,752
4	5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8	password	39h 57m 44s	3,298,522,879
4	d14697e20cc4b4b1123038a21b563b5d36a13607	Pa55w0rd	39h 57m 47s	3,298,692,143
4	d0d29dbcb4e330c1255f400391c8d4a9ee7d42c8	P@55w0rd	39h 57m 48s	3,298,629,423
4	477f26e6c56d49d89c36b43f390f6aacfc814f40	tw1l1ght	66h 31m 19s	3,612,661,539
5	c704f92d6a6aa809e16032b9aab796cacbe7300e	sparky01	1m 40s	4,987,781
5	38bdd82e40107a72c1d8264dbfac73ed4f235034	spooky01	1m 40s	5,006,971
5	e1729ec6c6557248527ff20e7bef2bcd082e9014	66google	52m 47s	158,570,771
5	2e0684e6d077d56c0c3452234fc5766d5a2bb0c5	images00	3h 0m 57s	273,849,283
5	5cfa25f7aa00b49d02666d40f44cc882b789bbf2	melons88	6h 56m 22s	543,606,841
5	3384e42ea71d8f23f5ab90d929a035689a540725	93prolog	10h 37m 51s	842,875,329
5	767fb1139b5759376e0711185044fe25f20f849f	prolog68	11h 35m 32s	1,157,382,032
5	3a357f71b2c87aa3291594140b1e8a08286845e8	purple74	12h 4m 22s	1,238,537,404

Table 6 - Results for Cracking 8 Character Passwords

5.2 6, 7, 9 & 10 Character Passwords

Table 7 shows the results from running the program with 8 character passwords using Modes 1-5.

Length	Mode	Hash	Password	Time	Hashes Tried
6	1	3d4f2bf07dc1be38b20cd6e46949a1071f9d0e3d	111111	0s	5
6	1	7aa129f67fde68c6d88aa58b8b8c5c28eb7dd3a3	albert	0s	5,859
6	1	765b16168d54de20d51fc068dfd4c52918bee7f7	calico	1s	18,715
6	1	6753f0841d07bbd2b4614d362460078ff1a213a8	torres	3s	117,520
6	2	3d4f2bf07dc1be38b20cd6e46949a1071f9d0e3d	111111	0s	42
6	2	0f1defd5135596709273b3a1a07e466ea2bf4fff	hello2	9s	357,623
6	2	24a6487f3f5918f1fcd5fb03caa72d1a0b2f2551	kiku92	1m 7s	3,534,633
6	4	3d4f2bf07dc1be38b20cd6e46949a1071f9d0e3d	111111	2s	74,899
6	4	7aa129f67fde68c6d88aa58b8b8c5c28eb7dd3a3	albert	38m 33s	106,228,001
6	4	765b16168d54de20d51fc068dfd4c52918bee7f7	calico	4h 1m 32s	546,390,197
6	4	6753f0841d07bbd2b4614d362460078ff1a213a8	torres	22h 26m 40s	3,124,609,262
6	4	12a21c939367c2cef277232902e0054901b1c367	v1013t	23h 33m 38s	3,271,581,053
6	5	0f1defd5135596709273b3a1a07e466ea2bf4fff	hello2	18s	1,484,837
6	5	24a6487f3f5918f1fcd5fb03caa72d1a0b2f2551	kiku92	25m 51s	127,984,703
7	1	11a8e0f13451ab44e27f473b792ce55bd4dee6d1	chothia	1s	25,636
7	1	9149c120fab5c39fbafcd6cf5cb17be22123bcdc	tripleh	4s	134,264
7	2	62e4555357a3f6a97eda1357a83572519b2d0898	beast12	3s	124,533
7	2	8c8bb14fdb1eb85fdb64cc88458417f84ee75c70	yellow6	37s	1,292,447
7	3	ca503848d26d2f0c848e1fd10191114ddc84639b	albert!	2s	199,176
7	4	54136668ab4865f3a410e1bea93c8713713a8d53	151176d	49m 53s	142,262,133
7	4	ca503848d26d2f0c848e1fd10191114ddc84639b	albert!	54h 36m 9s	673,090,701
7	4	edd35ba806ed03eb97864ec8cc0b6321b0a14165	cH4lana	34h 5m 27s	3,972,470,712
7	4	11a8e0f13451ab44e27f473b792ce55bd4dee6d1	chothia	35h 36m 12s	4,145,072,068
7	4	595cd892f9aa68f042f76d8cb891e41ca52a4ff1	muigy65	92h 48m 41s	4,447,290,723
7	4	9149c120fab5c39fbafcd6cf5cb17be22123bcdc	tripleh	132h 11m	6,851,225,060
7	4	8c8bb14fdb1eb85fdb64cc88458417f84ee75c70	yellow6	141h 38m 37s	8,402,926,768
7	5	ca503848d26d2f0c848e1fd10191114ddc84639b	albert!	12s	1,079,956
7	5	62e4555357a3f6a97eda1357a83572519b2d0898	beast12	14s	1,257,689
7	5	8c8bb14fdb1eb85fdb64cc88458417f84ee75c70	yellow6	45s	4,362,095
9	2	7679357857a838cae279d6123d61d629d38f32b7	albert123	29s	2,533,824
9	2	8dbfc07b3f68233ff8c0c8e646f8b9c51051c050	hotmail12	1m 37s	5,893,043
9	2	cc09e432f4d7b8a0dd3a8d0c2ecb726a302fea72	hotmail23	1m 37s	5,893,054
9	2	a8b53bf7af3e1d4b74ad5fec4a7a2ee52c23ec8f	sparkle55	2m 53s	14,094,756
9	2	7da3831030ab0c6eb4f43eb62ab06b95f2ffe9ba	banana342	6m 32s	28,051,343
9	2	7a8c31ca5a1ec518105c0609483438bca290dab4	smith2307	5h 25m 4s	889,652,308
9	4	8b7a37448027591890eac437688d8b057faf394a	C0st4r1cA	36m 12s	81,274,827
9	4	aee5253dec9b4a2eb8c44ac96e757a8bb5958771	m00nl1ght	2h 37m 29s	328,038,573
9	4	15b531d185c8965097bc4fbf963006478388c4dc	st4rl1ght	3h 21m 8s	482,739,472
10	2	1fb62e781eb36ea8071f987f31dcc50d90b686ec	ballons123	8m 4s	32,691,624
10	2	0073719c54a46fa0fd4b6ad94d6123df0dd7ee3f	amrita2000	1h 17m 58s	239,172,501
10	2	1482e6639185fde3cb9746db29866dc76bfb5dfb	engage7052	3h 10m 33s	542,997,553
10	2	3dc7a68de304169b0de54bab1c73afe0d972d400	hello11111	4h 9m 5s	689,271,612

Table 7 - Results for Cracking 6, 7, 9 & 10 Character Passwords

5.3 1 – 5 Character Passwords

Running the brute force method on 1, 2 and 3 character passwords did not crack any passwords. This means no passwords exist in the hash file of these lengths, unless there are any which have other characters than the standard 98 used. The times taken and total number of hashes tried can be found in Table 8.

Length	Running Time	Hashes Tried
1	0.004s	98
2	0.113s	9,604
3	8.203s	941,192
4	22m 32s	92,236,816
5	62h 44m 10s	9,039,207,968

Table 8 - Running Time and Total Hashes for 1-5 Character Passwords

Whilst running the brute force method on 4 character passwords three were successfully cracked. Table 9 shows the times and number of hashes before they were cracked.

Hash	Password	Time (S)	Hashes Tried
2a7250f92fb1d4fd60adfb8433b57395e18aed6a	fhtn	58	4,775,064
ea3cd978650417470535f3a4725b6b5042a6ab59	ryan	196	16,230,774
2a3c90346d40e9c540050534d832ceb3e0d25a49	2307	608	50,407,829

Table 9 - Results for 4 Character Brute Force Attack

I then tried running the program in the other modes on 4 character passwords to see if any would find this password and the time taken if found. The results for this are shown in Table 10.

Password	Mode 1		Mode 2		Mode 3		Mode 4		Mode 5	
	Time	Tried	Time	Tried	Time	Tried	Time	Tried	Time	Tried
fhtn	Not Found		Not Found		Not Found		Not Found		Not Found	
2307	Not Found		3s	205,108	Not Found		51s	4,143,609	59s	4,696,010
ryan	1s	41,140	Not Found		Not Found		6m 16s	30,582,933	Not Found	

Table 10 - Running Times to Crack 4 Letter Passwords

Running brute force attack on 5 character passwords cracked the passwords in Table 11.

Hash	Password	Time	Hashes Tried
c249f94e729640e5cdca7a5aa32723ffe6d511c1	fen1x	3h 30m 38s	465,078,820
3368af25d4136ae4b5a9baca4180b8ebb869e1b4	gacko	4h 10m 21s	553,441,099
aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d	hello	4h 51m 4s	649,529,217
29de88b068de73b6665bcc0a540b1523305c7537	hljeb	5h 54m 5s	656,097,654
1d6665923a80615e5a0f85c95348d9a09f679c64	pop91	10h 19m 28s	1,396,878,921
1a27452283b0b46720913760f056377eb0b6388c	reddy	11h 32m 3s	1,571,819,771
2b5c240e6abd88e71ffc225b0459016e4cba9bda	smith	12h 12m 37s	1,671,635,694

Table 11 - Results for 5 Character Brute Force Attack

I also tried running the program in the other modes on 5 character passwords. The results for this are shown in Table 12.

Password	Mode 1		Mode 2		Mode 3		Mode 4		Mode 5	
	Time	Tried	Time	Tried	Time	Tried	Time	Tried	Time	Tried
fen1x	Not Found		Not Found		Not Found		36m	133,986,985	Not Found	
gacko	1s	31,246	Not Found		Not Found		38m 57s	141,595,510	Not Found	
hello	1s	35,763	Not Found		Not Found		47m 44s	163,359,800	Not Found	
hljeb	Not Found		Not Found		Not Found		Not Found		Not Found	
pop91	Not Found		1s	1,764,452	Not Found		1h 34m 17s	292,894,436	3m 39s	8,628,828
reddy	2s	66,694	Not Found		Not Found		1h 39m 48s	302,539,342	Not Found	
smith	2s	74,312	Not Found		Not Found		1h 58m 4s	343,950,378	Not Found	

Table 12 - Running Times to Crack 5 Letter Passwords

The results for both four and five character passwords show that it is a lot quicker to crack passwords using other methods rather than a brute force attack. For example for the password *ryan*, the time taken is 1 second in mode 1 (checking a word list with no changes), compared to 3 minutes for the brute force attack. The brute force attack however does guarantee to find all possible passwords; *fhtn* and *hljeb* were not cracked by any method other than brute force.

5.4 11+ Character Passwords

Due to the running time of the program being more than estimated, I have not got time to test these password lengths. They could be run as an extension to the project.

5.5 Total Hashes and Running Time

Tables 13 and 14 show the total number of hashes and running time for each method

Password Length	Cracking Mode 1	2	3	4	5	6
1	N/A	N/A	N/A	N/A	N/A	98
2	N/A	N/A	N/A	N/A	N/A	9,604
3	N/A	N/A	N/A	N/A	N/A	941,192
4	54,546	332,900	1,810,060	44,411,983	62,098,160	92,236,816
5	91,814	3,874,460	26,464,814	419,174,410	3,728,326,220	9,039,207,968
6	131,335	35,662,740	69,940,526	344,677,887	3,144,377,766	N/A
7	148,595	265,940,750	116,937,540	258,096,307	3,920,882,614	N/A
8	184,839	651,893,450	165,937,605	3,471,557,402	2,798,746,489	N/A
9	118,631	1,064,490,000	187,784,045	685,534,948	4,166,033,526	N/A
10	97,783	1,478,480,500	229,817,950	897,145,793	2,107,717,582	N/A

Table 13 - Number of Hashes Tried

		Cracking Mode					
		1	2	3	4	5	6
Password Length	1	N/A	N/A	N/A	N/A	N/A	0s
	2	N/A	N/A	N/A	N/A	N/A	0s
	3	N/A	N/A	N/A	N/A	N/A	11s
	4	1s	4s	25s	8m 58s	12m 49s	18m 32s
	5	2s	1m 46s	8m 25s	2h 33m 9s	25h 49m 39s	62h 44m 10s
	6	4s	13m 56s	31m 10s	24h 55m 34s	38h 11m 26s	N/A
	7	4s	1h 29m 39s	42m 40s	143h 23m 56s	58h 50m 31s	N/A
	8	5s	4h 50m 44s	1h 12m 18s	72h 7m 9s	60h 49m 3s	N/A
	9	4s	6h 47m 18s	1h 9m 56s	5h 6m 12s	134h 43m 8s	N/A
	10	3s	8h 59m 31s	1h 26m 37s	6h 6m 24s	199h 2m 7s	N/A

Table 14 - Running Times

The total running time was 123 days, 1 hour, 37 minutes and 50 seconds (CPU time) and in this time 39,481,371,648 hashes were tried. Some of these were repeated but assuming 75% were unique this only covers about 2.03×10^{-38}% of the 1.5×10^{48} total hashes possible.

Even though the program ran for so long, over 90% of the passwords were cracked within 24 hours of the program starting. Table 15 shows the number of passwords cracked within different time periods. The average time to crack a password was 6 hours, 23 minutes and 36 seconds.

Time	Passwords Cracked
Less than 1 min	22
1 min to 10 min	12
10 min to 30 min	0
30 min to 1 hour	5
1 hour to 5 hours	5
5 hours to 10 hours	3
10 hours to 24 hours	3
Greater than 24 hours	5

Table 15 - Passwords Cracked Grouped by Time

5.6 Passwords Not Cracked

After completing the running of the program I was given the full list of passwords to see which ones I had not cracked. I also looked at how they might have been generated. Some of them were not cracked because the word used as a basis for the password not being in my list, others were not cracked because they were created using some of the methods found in my questionnaire that I did not implement and some were very complex passwords and I am unsure of how they were created.

Password	Basis Word	Changes							Reason Not Cracked
		Add Number	Add Symbols	Change Letters	Insert Symbols	Add Number at Start	Repeated Words	Multiple Words	
68738237	68738237								All numbers
callibrate	calibrate								Changed Spelling
5rfgy6	<unknown>								Complex password
gy578bh	<unknown>								Complex password
??jV4d+>	<unknown>								Complex password
25#m5Bera/	<unknown>								Complex Password
sp-nso4	sponsor			Y					Different letters changed
pa55w04d	password			Y					Different letters changed
eXch0n[e	exchange			Y					Different letters changed
dondon	don						Y	Y	Method not implemented
treetree	tree						Y		Method not implemented
thebagel	the bagel							Y	Method not implemented
filemonk	file monk							Y	Method not implemented
crowndark	crown dark							Y	Method not implemented
2307smith	smith					Y			Method not implemented
catWrench	cat wrench							Y	Method not implemented
rubberman	rubber man							Y	Method not implemented
yah00!	yahoo		Y	Y					Multiple methods
m0b!1e2	mobile	Y		Y					Multiple methods
deep6mud	deep mud				Y				Multiple methods
fedfour8	fed four				Y			Y	Multiple methods
MrB10bby	mr blobby			Y				Y	Multiple methods
justgo22	just go	Y						Y	Multiple methods
fileMONK	file monk			Y				Y	Multiple methods
bu1ld1ing	building			Y	Y				Multiple methods
jan3sm1th	jan smith				Y			Y	Multiple methods
tilted&666	tilted	Y			Y				Multiple methods
fr3dbl0gg5	fred bloggs			Y				Y	Multiple methods
beach_ed55	beached	Y			Y				Multiple Methods
c0mputer01	computer	Y		Y					Multiple Methods
af04dvp	af04dvp								Number plate
shinythings	shiny things								Too Long
42cliveroad	43 clive road								Too Long
CoventGarden97	covent garden								Too Long
m31o8o14	<unknown>								Unknown word as basis
l33dz4wy	<unknown>								Unknown word as basis
bm079321	<unknown>								Unknown word as basis
cznktnkt	<unknown>								Unknown word as basis
tkfrpfrp	<unknown>								Unknown word as basis
inv-l9qpbr	<unknown>								Unknown word as basis
poutsa	poutsa								Word not in wordlist
marapili	marapili								Word not in wordlist
redbull12	red bull	Y							Word not in wordlist
g00gleplex	googleplex			Y					Word not in wordlist

Table 16 Uncracked Passwords

Table 16 shows an example of some of the passwords which were not cracked and the possible methods of generation for these passwords and the reasons they were not cracked.

There are a number of reasons for passwords not being cracked as follows:

- **All numbers** – Having just a number as a password was not checked. In order to crack passwords like this I would have to implement another method which checked for only numbers.
- **Changed Spelling** – Words spelt incorrectly were not checked. In order to try to crack passwords like these the word list would need to be changed to include all possible spellings of words.
- **Complex Password** – These passwords look like strong passwords and I cannot determine how they might have been created. Altering the questions in the questionnaires and getting more people to answer them might allow me to find out how these passwords were generated and might give me some ideas of how to crack them. Otherwise the only suggestion for cracking these that I can think of would be a brute force attack.
- **Different letters changed** – If letters were changed to characters different to those I found people changed from my questionnaires then they would not have been checked. In order to crack these passwords the LetterChangeList file would need altering to allow for these letter changes. Asking more people to fill in the questionnaire would allow me to find more letters which commonly get changed and find out what they are changed to.
- **Method not implemented** – Some passwords were created using methods not implemented in my program. Methods such as a word with numbers added at the beginning, multiple words or repeated words were not found to be common in my questionnaires and therefore were not implemented. Looking at the uncracked passwords show these methods are used and so implementing them would crack additional passwords.
- **Multiple methods** – No check was made on passwords created using one method followed by a different method. These could be checked with some slight changes to the program, if this were done, the order in which the methods were applied would need to be decided. Additional research could help to provide this information.
- **Number plate** – the program did not check for number plates used as passwords, an additional method could be implemented which checks standard number plate formats.
- **Too Long** – only passwords up to 10 characters long were checked. Any passwords longer than this would not have been cracked no matter which way they were created. The program could be run to attempt to crack passwords with longer than 10 characters if additional time was allowed.
- **Unknown word as basis** – Passwords where the basis cannot be decided would not be cracked. Extending the word list may help solve this problem, or additional questions to find out what other types of things people use as the basis of their passwords.
- **Word not in wordlist** – a password using a word not in the word list as a basis of a password would not be cracked, extending the word list would help solve this problem.

6. Secure Password Generation

As shown by the results of passwords cracked, a password created using a word then changing it by a set of rules is not a secure way to generate passwords, even though it might be memorable. They can be cracked in a short time. Any rule used to create a password can be written into a program therefore making it insecure. The problem with this is that a password created using a random string of characters is extremely hard to remember, especially if it is of a significant length. To create a secure memorable password there has to be some randomness incorporated into it, and then possibly apply some rules to it to make it more complex and therefore more secure.

6.1 Memorability

As shown in Graph 5 – How Do You Remember Your Password from the questionnaire results, many people do things they are advised against to try to remember their passwords.
Methods used for remembering passwords include:

- Having a simple password
- Having a short password
- Writing it down
- Using the same password in multiple places
- Using the same password for a long time

Only about a quarter of the people asked felt their password was a secure password and could remember it without using one of these unadvisable methods.

6.1.1 Simple Passwords

Almost a fifth of people admitted that the only reason they could remember their password was because it was simple. This would mean their password is just a word with little or no changes made to it. This would be memorable for the user but it would be very insecure.

6.1.2 Short Passwords

As previously stated, the longer the password is the more secure it is likely to be. A few people said that their password was easy to remember because it was short. The results for the brute force attack show that a short password can be cracked quickly.

6.1.3 Written Down Passwords

Writing down a password in order to remember it is definitely a bad idea. It would not matter how secure the password was, if the paper it was written down on is found, the password is useless.

6.1.4 Using Same Passwords in Multiple Places

The most common way people remember their passwords is by using the same password for multiple systems, meaning they only have one to remember. This is unadvisable because if someone uses the same password for an internet banking service and an online email service, a hacker would only need to hack into the email system which is likely to have less security than the internet banking system, steal the password and use it to log into the internet banking system.

6.1.5 Using Same Passwords for a Long Time

Using the same password for a long time increases the chance of an attacker finding the current password. If a password takes a month for an attacker to crack but is changed by the user every 2 weeks the attacker will never be able to crack the current password for that user.

6.2 Generation

Taking the previous points into account, I will now suggest some methods of generating secure and memorable passwords.

6.2.1 Length

The results from the brute force attacks on 1-5 character passwords have also shown that a short password is insecure; a brute force attack on passwords up to 5 characters long took about 60 hours. Therefore predictions for the length of time for a brute force attack on 6, 7 and 8 character passwords would be 250 days, 70 years and 7000 years respectively. A malicious hacker with faster computers or a larger number of computers that the program could run on can significantly reduce these figures. For this reason anything less than 8 characters is unadvisable, so the passwords I am generating here will be at least 10 characters long.

6.2.2 Basis

Using a word or name as a basis is not a secure way of generating passwords as these can easily be found using a word list; therefore a more random string of characters is needed for the basis of the password. As this string still needs to be memorable for the user I would recommend using the first letters of a phrase which you know, providing these do not make a word.

For this example I am going to use a line from the song, "Don't Stop Me Now", by Queen; "*Don't stop me now 'Cause I'm having a good time*". Taking the first letters gives "*dsmncihagt*" which is a 10 letter series of characters but not an actual word. This adds more randomness to the basis of the password. There are a large number of phrases, sentences or lines from songs which could be used for this and although it would be possible to create a list of these which the program could use, the vast number would make it infeasible to try.

6.2.3 Characters

As previously mentioned using upper and lower case letters, numbers and symbols is also recommended. By increasing the number of different characters which are used in a password, the total number of passwords which can be generated also increases. Also by using upper case letters, number and symbols, there are 30,000 times more 8 character passwords possible than using only lower case letters, as shown in Table 17. With this in mind, the password should be changed to not only contain a string of 10 lower case characters; it should also include capital letters, numbers and symbols as well.

Using	Characters	Number of Passwords
LC	26	208,827,064,576
LC+UC	52	53,459,728,531,456
LC+UC+N	62	218,340,105,584,896
LC+UC+N+S	96	7,213,895,789,838,340

Table 17 – Number of 8 Character Passwords Using Different Types of Characters
(LC = Lower Case Letters, UC = Upper Case Letters, N = Numbers, S = Symbols)

6.2.4 Changes

To add capital letters, numbers and symbols to the password the same kinds of changes can be done as found from the results to the questionnaires. It is better if letters are changed for number or symbols which do not look like the letter, for example change an '*a*' for '*!*' instead of '*@*' and change '*o*' for '*7*' instead of '*0*'. This of course makes does make it less memorable but will increase security. Adding one or two unlikely symbols as well as a few likely ones should help keep the balance between security and memorability.

6.2.5 Passwords

Using the ideas suggested above I have generated the 5 passwords shown in Table 18.

Basis	Letters	Changed
Don't Stop Me Now Cause I'm Having A Good Time	dsmncihagt	*Ds39ci#aG+*
Clowns To The Left Of Me Jokers To The Right	cttlomjttr	C2t+o3j2tr
Today Is Gonna Be The Day That They're Gonna Throw It Back To You	tigbtdttgtibty	t!Gbt@ttGtib2y
We All Live In A Yellow Submarine Yellow Submarine Yellow Submarine	waliaysysys	Wal%aYs*3!
I Believe In A Thing Called Love Just Listen To The Rhythm Of My Heart	ibiatcljlttromh	IbIa=cljl2tr%m#

Table 18 - Generated Passwords

6.3 Strength

6.3.1 Website Test

To test the strengths of these passwords I will first test them on a number of websites which give an indication of the strengths of passwords. I will be using Google, Microsoft and MSN as these are well known companies with a large number of users. Google judges passwords as Too Short, Weak, Fair, Good or Strong. MSN uses Weak, Medium or Strong. Microsoft uses Weak, Medium, Strong or Best. Table 19 shows the passwords tested on each website.

Password	Rating		
	Google	MSN	Microsoft
Ds39ci#aG+	Strong	Strong	Strong
C2t+o3j2tr	Strong	Strong	Strong
t!Gbt@ttGtib2y	Strong	Strong	Best
Wal%aYs*3!	Strong	Strong	Strong
IbIa=cljl2tr%m#	Strong	Strong	Best

Table 19 - Strength Tests on Generated Passwords

The reason why only *t!Gbt@ttGtib2y* and *IbIa=cljl2tr%m#* gave a result of Best on the Microsoft website is because they recommend having a password of at least 14 characters which only these passwords have.

6.3.2 Cracking Test

The passwords were tested using my program. None of the passwords were cracked when running the program using all cracking methods. This could be due to the fact that as I wrote the program and knew the rules as to how it is cracking the passwords I could easily create a password which does not follow these rules.

Therefore, I will run JohnTheRipper on these passwords to see if that cracks any. I do not know exactly which methods JohnTheRipper uses so it would be difficult for me to create a password in a way that does not follow these rules. Again none of the passwords were cracked running JohnTheRipper on them for 7 days.

Although I would now class these passwords as fairly secure they do not look as though they would be easy to remember, but as they have been made using a set of rules, the same rules could be used to help remember the password.

6.4 Memorability

6.4.1 Memorability Test

Taking the two passwords, *Ds39ci#aG+* and *Wal%aYs*3!* I will ask three people to try to remember them over a period of time. I will give them the password, explain how it was created and ask them to try to remember it. I will then ask them at intervals of 1 hour, 1 day and 3 days what the passwords were.

		Password	
		Ds39ci#aG+	**Wal%aYs*3!**
Person A	**Test 1**	Ds39ci#aG+	WaliaYs*3!
	Test 2	Ds39ci#ag+	Wal%aYs*3!
	Test 3	Ds39ci#aG+	Wal%aYs*3!
Person B	**Test 1**	Ds39ci#aG+	Wal%Ays*3!
	Test 2	Ds39ci#aG+	Wal%aYs*3!
	Test 3	Ds39ci#aG+	Wal%aYs*3!
Person C	**Test 1**	dsmncihagt	waliays*3!
	Test 2	ds39ci#aGt	Wal%ays*3!
	Test 3	Dsm9C!#aGt	Wal%aYs*3!

Table 20 - Memorability Test Results
(Test 1 = 1 Hour, Test 2 = 1 Day, Test 3 = 1 Week)
(Red = Incorrectly Typed)

Table 20 shows the results of these tests; they show that these passwords are still memorable to some people, while others have some problem remembering them. If the users were to create similar passwords themselves, they may find them easier to remember. I found the passwords fairly memorable and believe that if I was typing them every day or a few times a day then I would be able to remember them quite quickly.

6.4.2 Increasing Memorability

As always there will be a trade off between memorability and security. To make them more memorable they could be shortened in length slightly to 8-10 characters and some, but not all, letters changed for more obvious characters like '*o*' for '*0*'. For example changing "*C2t+o3j2tr*" could become "*C2+l0Mj2Tr*" which might make it slightly easier to remember, but not sacrifice much security. Pressing the shift key for every other character, except the '*0*' in the middle will help the user decide which letter should be capitals and symbols. If this was done all the way through, a rule could be written into a program to help crack it.

As previously mentioned, Vu et. Al (2007) did some research into the memorability of passwords. In this paper it is shown that using the first letters from a sentence is less memorable than using a word as the basis of the password, but I have shown that it is a lot more secure, and therefore favourable. There are other ways of increasing the memorability of passwords. One thing shown by Vu is that if the user was made to remember the password within 5 minutes of creating it, they were more likely to remember it long term. Putting this into practice would mean after the password creation process, the user would have to log into their account using their new password.

Another method of increasing memorability of multiple passwords is to have one standard password and add a code onto the end to represent the system it is being used for. Table 21 shows some examples of this.

System	Password
Hotmail	P&s5WorDHML
Gmail	P&s5WorDGML
Ebay	P&s5WorDEBY
Amazon	P&s5WorDAMZ
etc...	

Table 21 - Multiple System Password Examples

7. Performance Analysis

7.1 Comparison with JohnTheRipper

7.1.1 General Comparison

I will start the comparison with JohnTheRipper by looking at how the programs function and what they test.

JohnTheRipper:

- Multiple modes of cracking such as a word list attack, an attack which uses character frequency tables to create additional words using most common letters first and a brute force attack.
- Considers frequency of words, so would check commonly used passwords first before others.
- Cannot easily be run in parallel, only one instance can be run at a time.
- Concentrates on words up to 8 characters in length.

My Program:

- Multiple modes of cracking such as a word list attack, adding numbers or symbols to the end of the words or inserting numbers and symbols.
- Takes words in alphabetical order, aardvark is tried before password.
- Can easily be run in parallel, multiple instances can be run testing different lengths of passwords and using different modes.
- Can run on any length of password.

As JohnTheRipper could only run on a single processor and I could run my program on multiple processors at the same time, however JohnTheRipper is more efficient at cracking passwords; it ran for about 21 days compared to the 123 CPU days mine could run for in the 3 weeks.

7.1.2 Comparison of Cracked Passwords

I will now look at the passwords cracked by my program compared to those which were cracked by JohnTheRipper.

Totals:

Number of passwords cracked by my program:	55
Number of passwords cracked by JohnTheRipper:	78

Breakdown of Passwords Cracked:

Number of passwords cracked by both programs:	36
Additional number of passwords cracked by only my program:	19
Additional number of passwords cracked by only JohnTheRipper:	42

Table 22 shows the full results and which passwords were cracked by each program, and reasons why certain passwords were cracked by one program but not the other. I feel my program is comparable to JohnTheRipper because it has cracked a lot of passwords and as previously mentioned, additional methods could be implemented which would allow it to crack more. The complexity of passwords cracked by both programs is similar.

JohnTheRipper did seem to crack the simpler passwords a lot quicker than my program but the more complex passwords took longer. As JohnTheRipper does not give any indication of time taken or number of password hashes tried I cannot give quantitative data for this.

Password	Cracked By My Program	Cracked By JohnTheRipper	Comments
2307	Yes	Yes	
111111	Yes	Yes	
151176d	Yes	Yes	
93prolog	Yes	Yes	
albert	Yes	Yes	
albert!	Yes	Yes	
beast12	Yes	Yes	
calico	Yes	Yes	
cH4lana	Yes	Yes	
chothia	Yes	Yes	
drowssap	Yes	Yes	
fen1x	Yes	Yes	
fhtn	Yes	Yes	
gacko	Yes	Yes	
hello	Yes	Yes	
hello2	Yes	Yes	
hljeb	Yes	Yes	
images00	Yes	Yes	
kiku92	Yes	Yes	
krabicka	Yes	Yes	
melons88	Yes	Yes	
muigy65	Yes	Yes	
password	Yes	Yes	
pop91	Yes	Yes	
prolog68	Yes	Yes	
purple74	Yes	Yes	
reddy	Yes	Yes	
ryan	Yes	Yes	
smith	Yes	Yes	
sparky01	Yes	Yes	
spooky01	Yes	Yes	
torres	Yes	Yes	
tripleh	Yes	Yes	
tw1l1ght	Yes	Yes	
v10l3t	Yes	Yes	
yellow6	Yes	Yes	
66google	Yes		
albert123	Yes		Too long for JohnTheRipper
amrita2000	Yes		Too long for JohnTheRipper
ballons123	Yes		Too long for JohnTheRipper
banana342	Yes		Too long for JohnTheRipper
C0st4r1cA	Yes		Too long for JohnTheRipper
engage7052	Yes		Too long for JohnTheRipper
green&34	Yes		
hello11111	Yes		Too long for JohnTheRipper
hotmail12	Yes		Too long for JohnTheRipper
hotmail23	Yes		Too long for JohnTheRipper
infr4R3d	Yes		
m00nl1ght	Yes		Too long for JohnTheRipper
P@55w0rd	Yes		
Pa55w0rd	Yes		
smith2307	Yes		Too long for JohnTheRipper
sparkle55	Yes		Too long for JohnTheRipper
st4rl1ght	Yes		Too long for JohnTheRipper
vera8859	Yes		
10011982		Yes	All Numbers
(Hu$h)		Yes	
2qasde3		Yes	
5rfgy6		Yes	
Achtuead		Yes	
af04dvp		Yes	Number plate
bbking		Yes	
bdhst0		Yes	
bm079321		Yes	
c0nch1ta		Yes	
cznktnkt		Yes	Not in Word List
dondon		Yes	Repeated Words
f1lem0nk		Yes	Multiple Words/Methods
fedfour8		Yes	Multiple Words/Methods
ferdosys		Yes	
filemonk		Yes	Multiple Words
g0rBal48		Yes	
gthusb5		Yes	
gy578bh		Yes	
justgo22		Yes	Multiple Words/Methods
k239auk		Yes	
kewell		Yes	Not in Word List
l33dz4wy		Yes	
lemesos		Yes	Not in Word List
lovelost		Yes	Multiple Words
lovenigt		Yes	Multiple Words
m0b!1e2		Yes	Multiple Methods
m3rcur1o		Yes	Multiple Words/Methods
marapili		Yes	Not in Word List
myriddin		Yes	Multiple Words
oktoberz		Yes	
orangsky		Yes	Multiple Words
pellaras		Yes	Not in Word List
poutsa		Yes	Not in Word List
takisole		Yes	
testtest		Yes	Repeated Words
thebagel		Yes	Multiple Words
tkfrpfrp		Yes	
treetree		Yes	Repeated Words
vIsiOo2		Yes	Multiple Methods
yah00!		Yes	Multiple Methods
yedstgi		Yes	

Table 22 - Comparison with JohnTheRipper

8. Evaluation & Conclusion

8.1 Project Achievements

I feel the project was successful and has met the objectives and requirements set out at the beginning of the project. The three main requirements were the following:

- **Research into Password Creation** – the internet research and questionnaires were a successful way to research password creation and the ways in which people generate their own passwords.
- **Implementation of a Password Cracking Program** – From the research I implemented a password cracking program which successfully cracked 55 real user passwords from a list of 244, it uses a number of different modes to crack the passwords and can crack passwords of multiple lengths. The results of my program were comparable to the results of the open source password cracking program JohnTheRipper.
- **Generating a Schema to Create Secure, Memorable Passwords** – This objective was met because the passwords created were tested and found to be secure and memorable for people.

8.2 Problems Encountered

The main problem I found while doing the project was that the running times were longer than estimated for some of the methods. To reduce the running times of the program I had to reduce the size of the word lists for some of the methods. If there had been more time available for the project, longer program runs would have been possible and therefore longer word lists giving a higher number of possible hashes tested.

If I had started with a smaller word list of the most common words, I could have run the program a lot quicker and therefore tried longer passwords. This would have reduced the overall running rime of my program greatly and made it more comparable to JohnTheRipper. I believe this would have resulted in more cracked passwords. The word lists could then have been changed to include less common words if time allowed.

8.3 Final Conclusion

Passwords generated using a set of rules are not necessarily as secure as people first think. They can easily be broken by a program designed to mimic the human thought process and crack passwords using the same set of rules.

Generating secure and memorable passwords will always be a challenge, and there will always be a trade off between the two. Using a set of rules will make more memorable passwords but as previously explained not necessarily secure ones. Therefore a compromise has to be made. By using the first letters from a phrase or sentence, then applying a set of rules to the resulting letters can help make a password memorable, and as there are so many possible phrases it increases the security because it is infeasible to try them all.

8.4 Further Work

8.4.1 Additional Running Time

As previously mentioned, if more time were available then the program could have been allowed to run for longer. This would increase the number of possible words manipulated and tested, increasing the probability of cracking passwords.

In addition to this more time would also have meant passwords greater than 10 characters in length could have been tested.

8.4.2 Additional Research

Another improvement to the project could be made by increasing the research into user password generation. By having more people answer the questionnaires and changing the questions asked could give more information about how people generate passwords, leading to new methods of generation which have not been considered in this project. It could also give additional possible letter changes for the Letter Change mode.

8.4.3 Additional Methods

From the research I conducted and any additional research done, additional methods could be found which could be implemented to increase the number of hashes produced.

Some examples of additional methods could be:

- Add numbers to the beginning of a word
- Repeat a word
- Concatenate two words
- Test words commonly used as passwords such as admin or password first

8.4.4 Improve Methods

Improvements could be made to existing cracking methods; these could include adding more digits or symbols onto each word or inserting more characters into each word.

Another improvement would involve prioritising more commonly changed letters in the letter change mode, or prioritising more commonly used words in each word list.

9. References

9.1 Papers

An Assessment of Website Password Practices, by Steven Furnell, *2007*
Journal of Computers & Security 26, *pages 445-451*

Improving Password Security and Memorability to Protect Personal and Organizational Information, by Kim-Phuong L. Vu et. Al *2007*
Journal of Human Computer Studies 65, *pages 744-757*

The Usability of Passphrases for Authentication: An Empirical Field Study, by Mark Keith, Benjamin Shao, Paul John Steinbart *2007*
Journal of Human Computer Studies 65, *pages 17-28*

9.2 Websites

Birmingham University Password Policy, by The Support Team
Address: http://supportweb.cs.bham.ac.uk/policies/passwords.php
Updated: 22 July 2008
Accessed: July-September 2008

Strong Passwords and Password Security, by Microsoft Security
Address: http://www.microsoft.com/protect/yourself/password/create.mspx
Updated: 22 March 2006
Accessed: July-September 2008

Password Checker, by Microsoft
Address: https://www.microsoft.com/protect/yourself/password/checker.mspx

Updated: Unknown
Accessed: July-September 2008

Create a Google Account, by Google
Address: https://www.google.com/accounts/NewAccount
Updated: Unknown
Accessed: July-September 2008

Sign Up, by MSN
Address: https://accountservices.passport.net/reg.srf?roid=2&wa=wsignin1.0&rpsnv=10&ct=1221483252&rver=5.5.4177.0&wp=LBI&sl=1&lc=2057
Updated: Unknown
Accessed: July-September 2008

Does Your Password Pass The Test? By HongHai Shen
Address: http://googleblog.blogspot.com/2008/06/does-your-password-pass-test.html
Updated: 6 April 2008
Accessed: July-September 2008

My SHA-1 Example, by Eugene Styer
Address: http://people.eku.edu/styere/Encrypt/JS-SHA1.html
Updated: Unknown
Accessed: July-September 2008

Choosing a Strong Password, by Joe Sanjour, University of Maryland
Address: http://www.cs.umd.edu/faq/Passwords.shtml
Updated: 3 October 2002
Accessed: July-September 2008

Choose (and Remember) Great Passwords, by Gina Trapani
Address: http://lifehacker.com/software/passwords/geek-to-live--choose-and-remember-great-passwords-184773.php
Updated: 5 July 2006
Accessed: July-September 2008

Guidelines for Choosing a Strong Password, by Lockdown.co.uk
Address: http://www.lockdown.co.uk/?pg=password_guide
Updated: 22 January 2004
Accessed: July-September 2008

Various Pages, by Wikipedia.org
Address: http://en.wikipedia.org/wiki/Main_Page
Updated: Various
Accessed: July-September 2008

9.3 Other Resources

9.3.1 JohnTheRipper

JohnTheRipper Password Cracker, hosted by The Openwall Project
Homepage: http://www.openwall.com/john/
Version: 1.7.0.1 for Windows

JohnTheRipper SHA1 Patch, Part of **1.7 + jumbo patch build for Win32,** by Thomas Springer
Address: http://www.openwall.com/john/contrib/john-1.7-multipatch-win32mmx-v02.zip
Accessed: July-September 2008

9.3.2 Library

Libgcrypt, by GNU
Version: 1.4.0
Reference Manual: http://www.gnupg.org/documentation/manuals/gcrypt/

9.3.3 Code

SHA1 C/C++ Procedure? By tomchuk
Address: http://ubuntuforums.org/archive/index.php/t-337664.html
Updated: 14 January 2007
Accessed: July-September 2008

System's Programming in C/C++, by Eike Ritter
Address: http://www.cs.bham.ac.uk/~exr/teaching/lectures/systems/07_08/lectures.php
Updated: 4 March 2008
Accessed: July-September 2008

9.3.4 Word Lists

Word Lists, by Outpost9.com
Address: http://www.outpost9.com/files/WordLists.html
Updated: 7 December 2005
Accessed: July-September 2008

Kevin's Word List Page, by Kevin Atkinson
Address: http://wordlist.sourceforge.net/
Updated: 17 June 2007
Accessed: July-September 2008

10. Bibliography

Choosing a Strong Password, by Joe Sanjour, University of Maryland
Address: http://www.cs.umd.edu/faq/Passwords.shtml
Updated: 3 October 2002
Accessed: July-September 2008

Choose (and Remember) Great Passwords, by Gina Trapani
Address: http://lifehacker.com/software/passwords/geek-to-live--choose-and-remember-great-passwords-184773.php
Updated: 5 July 2006
Accessed: July-September 2008

Guidelines for Choosing a Strong Password, by Lockdown.co.uk
Address: http://www.lockdown.co.uk/?pg=password_guide
Updated: 22 January 2004
Accessed: July-September 2008

11. Appendix A - Questionnaire

11.1 Website

Print out of the website used for the questionnaire:

11.2 Sample Email

An example of the email generated by the website showing a set of results:

11.3 Appendix B - LetterChangeList

A print out of LetterChangeList the file used by the program to identify which letters get changed for which numbers and symbols:

12. Appendix C – CD Information

12.1 Files on CD:

The following list of files is included on the CD:

- CreateLetterList.sh – Shell script to create the letter list files from the word list
- feasibility.c – code for feasibility testing
- include.c – additional functions used by the cracking program
- LetterChangeList – file containing the characters each letter gets changed to
- passwordFile.hash – file containing sha-1 hashes of the passwords
- readme.txt – explanation of how to run the program
- Report.pdf – this report
- sha-1.c – main program code
- WordList – word list for cracking

12.2 Running the Program

12.2.1 Cracking User Passwords

Compiling the program:

```
gcc -lm -lgcrypt -Wall -Werror -o sha-1 sha-1.c
```

Running the program:

```
./sha-1 <length> <hashfile> <mode> [letterchangelist]
```

Where:

`length` = length of passwords to crack
`hashfile` = file containing sha-1 hashes of passwords to crack
`mode` = cracking mode:

1 - words on their own
2 - words followed by numbers
3 - words followed by symbols
4 - words with letters changed for numbers or symbols
5 - words with numbers and symbols inserted
6 - brute force attack

`letterchangelist` = file containing list of letters to change, only used in mode 4

Examples:

Try 8 character words:

```
./sha-1 8 passwordFile.hash 1
```

Try 8 character passwords with letters changed for numbers/symbols

```
./sha-1 8 passwordFile.hash 4 LetterChangeList
```

Try a 3 character brute force attack

```
./sha-1 3 passwordFile.hash 6
```

12.2.2 Running Feasibility Testing

Compiling the feasibility program:

```
gcc -lm -lgcrypt -Wall -Werror -o feasibility feasibility.c
```

Running the program:

```
./feasibility <time>
```

Where:
time = running time of the program in seconds

12.2.3 Creating Letter List Files

Running the program:
```
./CreateLetterList <wordlist>
```

Where:
```
wordlist = Word list file
```