Using Natural Language Processing to Create Personalised Password Wordlists

Zack Anderson (zack.a@myself.com)

BSc (Hons) Ethical Hacking

28th April 2020

Supervised by Dr. Ethan Bayne (e.bayne@uad.ac.uk)

School of Design and Informatics
Abertay University

Table of Contents

Table of Figures

Table of Tables

Acknowledgements

I would like to thank my supervisor, Dr. Ethan Bayne, for his feedback and support during this project. I'd also like to thank my friends for making university the best 4 years of my life so far, and all my lecturers who have taught me so much and inspired me to study this industry. And finally, my parents, who helped me in every way while at university.

Abstract

This paper documents the investigation of applying natural language processing technology to building customised password wordlists. Upon testing the proof-of-concept tool created for the project, it was found that utilising natural language processing along with common password mutations can significantly increase the efficiency and reliability of a wordlist, cracking passwords in a median 0.132% of the attempts using the RockYou wordlist. Testing of the tool also showed that it and the natural language processing model were lightweight enough to be run in virtual environments, which are common among security researchers and students.

Natural language processing has progressed massively in recent years since the inception of the Word2Vec algorithm, allowing people to interact with computers in new ways such as virtual assistants, which are becoming more common in smartphones, speakers, and even televisions. By first investigating how people typically iterate or mutate passwords, this project proposes integrating natural language processing technology with common password mutation techniques to reduce the amount of effort required to crack passwords in a dictionary attack scenario. This is presented in a proof-of-concept tool which utilises a model trained on the Word2Vec algorithm to build a customised wordlist which can crack passwords more reliably and more efficiently than generalised wordlists.

The Literature Review covers previous literature and research relating to natural language processing, human factors which affect password choice, password policies, password wordlists, and password attack mitigations. The Methodology chapter, which is broken down into the research, development, and testing stages, then goes on to describe how this proof-of-concept tool was developed. The first stage of the methodology, the research stage, describes the creation and distribution of the survey as well as how previous literature was sourced – before moving on to the development stage, which describes the key functions of the tool and discusses the rationale behind design choices. The chapter rounds off

with a description of how testing was carried out with findings presented in chapter 4.

Keywords: Passwords, Password Cracking, Natural Language Processing, Password Choice, Password Psychology

Abbreviations, Symbols and Notation

Abbreviation	Meaning
RAM	Random Access Memory
CBOW	Continuous Bag-of-Words
SG	Skip-gram
OSINT	Open-Source Intelligence
GPU	Graphics Processing Unit
2FA	Two-Factor Authentication
GB	Gigabyte
MH/s	Mega-Hashes per Second

1 Introduction

1.1 Background

Natural Language Processing (NLP) is a branch of machine learning focused on allowing computers to understand natural human languages such as English. This has proven a challenge in the past as computers commonly operate with structured data, whereas human languages are rather ambiguous in nature. NLP models are trained using large sets of text known as corpora, which the algorithm then analyses using a variety of techniques to help disambiguate and derive context from the words and stores this data in the form of a model.

Passwords have been used as verification for far longer than computers have existed, such as secret phrases said to ensure another person can be trusted, or examples in literature where a secret phrase allows the protagonist through a door or past a guard. Passwords became commonplace with the advent of multi-user systems which could be accessed from distributed networks to verify that a user is *"trusted"* and assign them appropriate privileges. When networks became further distributed with the creation of the internet, passwords increased in popularity due to their simple yet uniquely customisable nature which makes them easy for people to remember. Passwords have become a common point of attack for security researchers and penetration testers as passwords creation, for ease of use, is left in the hands of the user. This means they will often be constructed using memorable words, phrases or acronyms, which can be vulnerable in security terms as they are predictable. This means passwords are often easily guessable among those who are less aware of secure password practices. Attackers will usually use a wordlist – a file containing thousands or millions of known passwords and passphrases – to crack passwords with the obvious limitation that if the password of the target is not found within the wordlist, it will not be cracked. This is known as a dictionary attack (CySecurity, 2011).

Bill Gates predicted the death of the password in 2004 outlining their shortcomings as a security mechanism, stating that *"they just don't meet the*

challenge for anything you really want to secure" (Kotadia, 2004). The reason for this he states is that people use the same (or similar) passwords across systems or store them insecurely by writing them down somewhere.

Security consultant Mark Burnett further discusses this issue in his book *"Perfect Passwords"* where he outlines the common patterns in passwords, the psychology behind them, and how natural language reduces password entropy as some characters are seen more frequently than others (Burnett, 2006d). He stated in this book that most passwords can be sorted into 7 main types:

- Wordlist passwords – These are simple passwords which contain dictionary words or names.
- Wordlist passwords with numbers – The same passwords as above, but with a number tacked onto the beginning or end.
- Wordlist passwords with obfuscation – A wordlist password with common character substitutions, such as swapping an *"a"* for *"@"*.
- License plate passwords – A password containing common shorthand or abbreviated words such as *"sk8"*.
- Doubled wordlist – A password consisting of dictionary words or names typed out twice.
- Patterns – Password consisting solely of common patterns such as *"QWERTY"* or *"123456"*.
- Random – A password which is randomly generated or pseudo-random in nature.

Since Bill Gates predicted their death in 2004 passwords have only become more common with almost every account created on a web service requiring one. To force users to create more secure passwords, many systems or websites will suggest measures or even enforce policies of complexity in passwords. Password policies can state that a password must contain numbers, special characters (such as punctuation characters), capital letters, or set passwords to expire after a time interval. There is little evidence to support that these policies create more secure passwords and can instead do the opposite as to make passwords more memorable when forced to change them, people will often use a number of

iteration methods which rather than making the next password truly unique often mean a progression or guessable sequence is used instead.

A study by Carnegie Melon University's CyLab found that implementing more complex password policies led to more easily brute-forceable passwords, with the most secure policy simply implementing a minimum length of 16-characters (Kelley, et al., 2011). A brute-force attack entails attempting every possible character combination a password can have – meaning that effort required to crack the password rises exponentially as length increases. By specifying that some characters must appear at least once, password policies can instead help to accelerate this process by specifying certain characters which must appear. Another paper presented at the Open Identity Summit 2016 suggests that these complex password rules which cannot be read by password managers such as LastPass can also lead to users choosing weaker passwords (Horsch, et al., 2016).

With passwords now a common method of authentication on most web services, secure storage of passwords has become a focal point of governing bodies to protect users. Large businesses now know not to store passwords in plain-text as RockYou did in 2009, which resulted in their database becoming a popular password wordlist among hackers. Nowadays, this negligence can lead to large fines under new guidelines such as the European Union's General Data Protection Regulation (GDPR) (Cubrilovic, 2009). This does not mean that companies are now invulnerable and, despite increasing regulation, there is evidence to support that the issue is still growing in scale as illustrated in Figure 1, which shows the number of records leaked in data breaches from 2010 to 2019 (Identity Theft Resource Center, 2019).

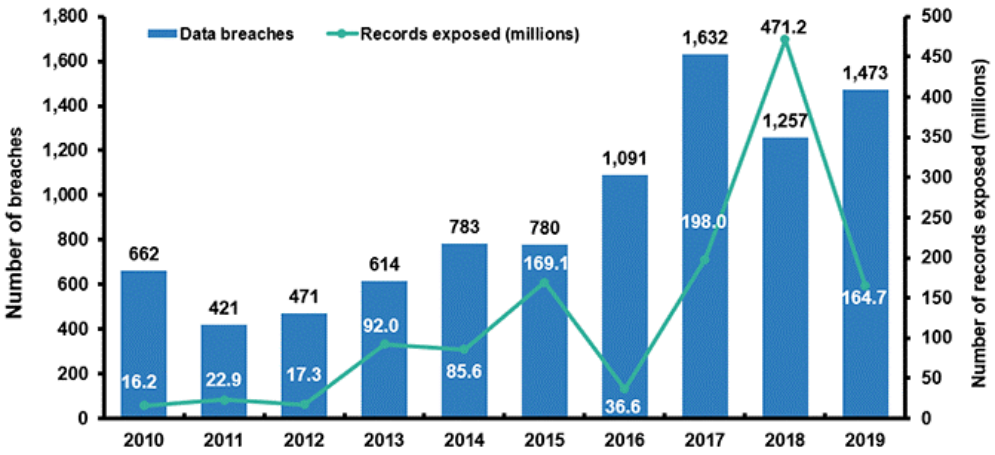

Figure 1 - Number of Data Breaches and Records Exposed, 2010-2019

A 2019 report by Verizon found that 29% of breaches occurred using stolen credentials and that 32% involved some form of phishing campaign – where the attacker attempts to gain information about the target, including login information, hobbies, interests, etcetera (Verizon, 2019). This can be executed by directing the target to an apparently reputable website or by sending an e-mail with a spoofed source, both of which are designed to lure the target into divulging information. This means that if a data breach includes password data and the same password or a progressively similar password is used across multiple websites and accounts, it should be possible to predict other passwords of the target using information gained from data breaches or phishing campaigns if what is stated in Burnett's book is true. This paper proposes a proof-of-concept tool called Natural Language Passwords which was created to illustrate to users with simple passwords that a data breach may affect them more than they realise by showing just how easily their alternate passwords can be cracked. The tool was constructed by analysing common password patterns submitted to a survey and utilises a natural language processing model to generate a customised wordlist based on a known password.

1.2 Research Questions

This paper will discuss the following questions:

i. Can a natural language processing model be used to create a more diverse customised wordlist for dictionary attacks using a known password?

ii. What trends are commonly found in a user's password iterations?

iii. Can these be predicted and used in conjunction with a natural language processing model to create a more reliable and efficient customised wordlist once one password is known?

1.3 Project Aims

This project aims to create a proof-of-concept tool which can generate wordlists to crack passwords more efficiently and more reliably than other popular wordlists. The tool should be lightweight enough to be run in virtual environments, which are often used by cyber security researchers and students to conduct research. These virtual environments commonly have low system resources such as processing power and memory meaning usage of such resources should be kept as minimal as possible.

1.4 Structure

The second chapter in this paper is a literature review which goes covers the existing literature on topics relating to this project, including natural language processing, human factors that affect password choice, password wordlists, and password attack mitigations. Chapter 3 then goes on to explain how the project was executed, including the distribution of a survey, development, and testing. Chapters 4 and 5 go on to present and discuss the findings from this testing, challenges faced during execution, and any abnormalities in test results. Chapter 6 then highlights any conclusions from this project and possible future work.

2 Literature Review

The following chapter provides an overview of academic sources used during the research phase of this project. Firstly, discussing previous research in the field of natural language processing, then on human factors in passwords, password wordlists used by attackers and penetration testers, and finally, mitigations for password attacks.

2.1 Natural Language Processing

Natural language processing (NLP) is a branch of machine learning concerned with enabling computers to understand natural languages such as English. This is a challenge as computers operate with structured data, whereas natural languages are often ambiguous in nature. This is achieved by analysing the syntactic and semantic structure, disambiguating the words, and tagging parts of speech contained in sentences to ensure they are grammatically correct and derive context from the words in the sentence or document. NLP Models are trained using large sets of sentences or documents known as corpora, which can be annotated to provide enhanced results in training algorithms that support supervised learning.

2.1.1 Word2Vec

Word2Vec is a machine learning model utilising a single-layer neural network to compute continuous vectors of words for large datasets with billions of words, containing millions of words of vocabulary (Mikolov, et al., 2013a). Word2Vec models are trained using a context window, which specifies the number of words before and after a given word are included to derive its context. This paper puts forward 2 architectures for this model: a continuous bag-of-words model (CBOW), and a continuous skip-gram model (SG). The CBOW model is appropriate for predicting words given their context, where the input could be *"Wi−2,Wi−1,Wi+1,Wi+2"* and output would equal a predicted word, represented as "*Wi"*. The skip-gram model is essentially the reverse of this, where given the

word *"Wi"*, it can predict words within a range before and after the word, illustrated below in Figure 2.

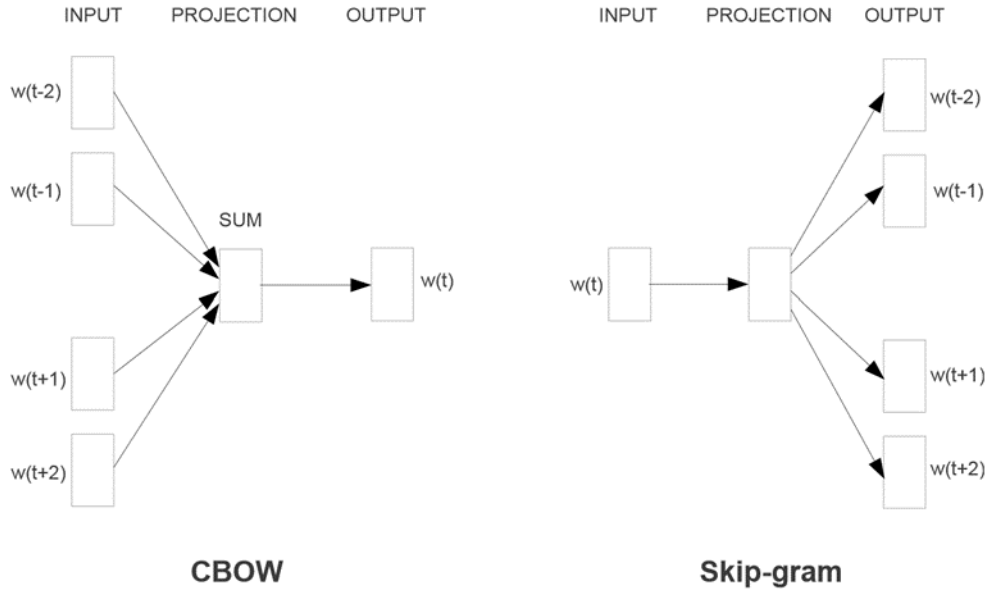

Figure 2 - Illustration of Continuous Bag of Words vs Skip-Gram Models

The results of this paper show that the skip-gram model was approximately 30% more accurate at semantic tasks than a CBOW model and performed almost identically in syntactic tasks, with the drawback of taking significantly longer to train.

2.1.2 FastText

The FastText model builds upon the skip-gram model by classifying each word as a vector representation of its character *n*-grams (Bojanowski, et al., 2017). For example, the word "*moonlight"* contains the character *n*-grams of "*moon"* and *"light"*, and as such can derive that these words are most likely related. This also allows FastText models to calculate vectors for out-of-vocabulary words. Results in this paper report a 70.1% accuracy in syntactic tasks for their skip-gram model, which was outperformed by their FastText model with 74.9% accuracy. The drawback of this model is that if text corpora are not spell checked, it will often report common misspellings as related words.

2.1.3 Gensim

Gensim is an open-source Python library which allows for fast, memory-efficient, scalable semantic analysis of plaintext documents (Řehůřek & Sojka,

2011). It allows for training of models such as Word2Vec, Doc2Vec, and FastText, using corpora larger than RAM by using iterators and includes a function which allows clients to query a model with a specified word to return the most similar word(s). The library was written by the Faculty of Informatics at Masaryk University and is partially supported by both the Czech Ministry of Education and the European Union Competitiveness and Innovation program – allowing the library to remain well-maintained.

2.2 Human Factors

2.2.1 Human Language

Human languages tend to increase predictability, also known as removing entropy, of passwords by creating predictable series' of characters, such as the well-known *"'i' before 'e' except after 'c'"* rule (Burnett, 2006a). This means that there is an uneven distribution of characters found in passwords and that people tend towards lowercase letters and numbers in their passwords. In contrast, passwords generated by computers feature a near-even distribution of characters.

2.2.2 Contextual Bias

People often tend to choose passwords which relate to their personal lives or environments to make passwords easier to remember. Furthermore, people often choose related words to lengthen or change their password. In his book, Burnett gives a list of passwords relating to the base word of *"dragon"* and in many of these examples, such as *"dragonfire12"* and *"dragonball"*, there is a relation to common characteristics associated with dragons or pop culture.

2.2.3 Password Policies

Many companies nowadays implement password policies for both employees and customers. These are rules which a user's password must comply with to be accepted by the system to force users to create more secure passwords. Since there is no standardised format for specifying these rules, further complications such as manual configuration when using password generators to create a password which meets policy specifications. This inconvenience can often lead

to users falling back on weaker passwords rather than using a password generator to create a strong password (Horsch, et al., 2016).

There is also no quantifiable data to suggest that password policies make passwords more difficult to brute-force. A study carried out by Carnegie Melon University measured the effectiveness of common password policies by simulating how many guesses it would take for a brute-forcing model to crack passwords created under various policies (Kelley, et al., 2011). To gather passwords, the study used Amazons Mechanical Turk service to gather over 12,000 plaintext passwords. The study found that contradictory to common belief, the passwords most resilient to brute-force attacks were not those which enforced requirements such as the inclusion of numbers and letters. Instead, it was those which simply enforced a length of over 16 characters in length which created the greatest resilience to the simulated brute force attack.

In his book, Burnett also backs the idea that complex password policies can lead to more predictable passwords and suggests a similar ruleset requiring a high minimum length with fewer restrictions on characters (Burnett, 2006b). He suggests this to prevent user frustration and make password iterations less predictable, specifically in environments where the user is forced to change their password after a given amount of time. Longer passwords also mean that password *"cycles"* can be longer, as the number of possible character combinations rises exponentially as length increases and therefore so does the effort required to crack the password.

2.2.4 Password Choice

Previous research suggests that users often become attached to their passwords (Burnett, 2006c). This means that, even in systems where users are required to choose new passwords after a certain amount of time has passed, they will often choose to stick with a similar password as it is memorable. Worse still is that some users will go as far as exploiting the reset mechanism to reverse the changes they were forced to make. This has led many security researchers to conclude that often humans are the weakest link in the chain of security (Schneier, 2000). A security system can be impeccable, but it is only as secure as

its weakest point which, in many cases, is the point of user access. According to Schneier, this happens because for many people the threat in a cyber landscape is not immediate. Since users cannot see someone breaking into their computer and often regard themselves as too insignificant to be the subject of a hacker, they often do not worry that they may well be. The landscape of cybersecurity has changed substantially since this book was written, but these facts are still relevant to this day.

2.3 Password Wordlists

2.3.1 Common Wordlists

A wordlist is a list of common strings which can be used to perform dictionary attacks on various services. Tools designed for use by penetration testers will often contain a set of wordlists out-of-the-box. These wordlists often contain the most common web application directories, usernames, and passwords; and occasionally are sorted to target specific software or services. Common password wordlists are often comprised of the most frequently occurring passwords in password breaches – such as the *"top-207-probable-v2.txt"* wordlist, which was constructed by counting passwords that occurred more than 350 times in a record of 2 billion passwords (berzerk0, 2019). Other common wordlists are dumps from specific online services or companies, such as the infamous *"rockyou.txt"* wordlist which originated from a database dump of plaintext passwords of users who interacted with the social network application developer "*RockYou"* (Cubrilovic, 2009).

2.3.2 Customising Wordlists

Customising wordlists using known information on a target can significantly reduce the amount of effort required to crack the user's password, or find obscure passwords that would not be in a generalised wordlist (Weidman, 2014). Methods such as Open-Source Intelligence (OSINT) gathering can be and are used by penetration testers and hackers to identify interests, hobbies, pets, and many other attributes of a single target which may contribute to their choice of password. Tools such as CeWL can also be used to enumerate web applications for unique strings which can be used to generate or add to a customised wordlist

(Wood, 2016). Tools such as "*Crunch"* can also be used, however, this essentially provides the same functionality as a brute-force password attack (Offsec Services, n.d).

In his talk at *Securi-Tay 2019,* Robin Vickery discussed how he typically customises wordlists by including nearby towns, football teams, and words used in the targets industry to personalise his wordlists (Vickery, 2019). He then went on to discuss using natural language processing to automate this process by training a word2vec model and retrieving the most similar words by their vector to feed into his custom wordlist.

2.4 Password Attack Mitigations

There are several mitigations commonly used to combat password attacks, such as password policies mentioned earlier in this chapter, to prevent an attacker from gaining access using a password authentication mechanism. These can be sorted into two main categories: online and offline mitigations.

2.4.1 Online Mitigations

A common threat to any user authentication mechanism open to the internet is password attacks in the form of brute-forcing or dictionary attacks. (OWASP Foundation, 2020) outline the challenges presented when trying to mitigate online authentication attacks. One common mitigation is blacklisting IP's launching brute-force attacks. However, many brute-force tools can relay packets through proxy servers meaning that blocking one simply starts the attack from another IP address. Another recommended mitigation from OWASP is to lock accounts. After a given number of failed authentication attempts, the system will lock the account being attacked for several minutes. While this will not prevent a brute-force attack, it will substantially increase the time required to crack the password of the target to the point where most attackers will give up but could allow attackers to carry out denial-of-service (DoS) attacks by locking multiple accounts.

Other mitigations include:

- Adding failure strings as HTML comments, meaning some popular cracking tools cannot detect when they have identified valid credentials.
- Use of the *"Completely Automated Public Turing Test to Tell Computers and Humans Apart"* (CAPTCHA) authentication system.
- Creating an authentication delay of a few seconds to slow down attackers further.

While none of these solutions can completely mitigate online password attacks, used effectively or in combination, they can create enough of a nuisance to discourage potential attackers.

2.4.2 Offline Mitigations

Offline mitigations are the last line of defence against password cracking. Offline attacks require the attacker to have a copy of the password storage file found on the authentication server, which allows them to attack the password limited only by the cost of computing the hash function (Blocki, et al., 2018). This is also a much stealthier option as the attacker will not leave logs on the server beyond copying the file. Simply hashing the stored passwords once will not do much to slow an attacker in the case of an offline attack. In his case, using a strong hashing algorithm thousands of times and using a salting string, the effort required to crack passwords can be greatly increased, allowing more time for potential victims to change their passwords.

One example of this is the Django Password-Based Key Derivative Function 2 (PBKDF2) (1Password, 2019). Passwords are salted using a secret string (for instance, a master password) and hashed thousands or hundreds of thousands of times. This method prevents crackers from making optimal use of graphics processing units (GPU's), thereby increasing the effort required to crack the password significantly.

2.5 Summary

This research within this chapter has covered the traps users often fall into when choosing passwords, how penetration testers use this to their advantage, and how developers implement mechanisms to help mitigate these attacks. This research

has provided insight into how natural language processing can be potentially used to help penetration testers in their mission to improve client security, an idea which this project builds upon.

3 Methodology

The methodology explains the considerations and steps taken while completing this project – starting with completing background research by distributing a survey and analysing past works outlined in the previous chapter, before moving on to the development of the script. Finally, the formulated testing procedure of the final script is discussed.

3.1 Research Stage

3.1.1 Survey

The first step of the research stage is to distribute a survey to understand and analyse common iteration patterns in passwords. This survey can be hosted using Google Forms and distributed on social media platforms. In accordance with Abertay University's code of ethics – the participants should be above the age of 18, given a brief of the project and how their data will be used and asked for explicit consent to anonymously collect and process this data. Due to the anonymised storage of the data, participants will also not be able to withdraw from the survey once they submit. This survey asks participants about their:

- Technical competency.
- Age group.
- Behaviour when choosing new passwords for social and important accounts (such as online banking, etc.).
- Frequency changing passwords and their reasoning for doing so.
- Typical reaction when notified of a data breach.
- Use of two-factor authentication (2FA) and password management services.

The survey also asks participants to create passwords given various sets of rules such as including numbers and special characters, as well as asking them what they would change their previously inputted password to if it was involved in a data breach. This data can be used to determine patterns in changing passwords when users are supplied with various rules, with the hypothesis of using the same password but adding numbers or substituting characters for special characters (for example, swapping *"a"* for *"@"*) or using a similar word with a similar password structure.

3.1.2 Previous Works

The research stage also involves looking into previous works and academic papers relating to the topics which will be covered during the development of this project by searching for papers on Google Scholar. These topics include information on natural language processing, factors which make human password choice predictable, customised wordlists, and password attack mitigations. Using this information, the scope of work to be completed in the development stage of the project can be established.

3.2 Development Stage

3.2.1 Requirements & Targets

As most cyber-security students and penetration testers use their tools in a virtual machine (VM), the script had to be as lightweight as possible. The main issue with this will be training a comprehensive model that is not so large it cannot be loaded into a machine with 4GB or 2GB of RAM. This should be a consideration when choosing corpora for the model and the size of wordlists generated by the script.

The target for this tool is to generate a password wordlist which will successfully crack the user's new password in fewer attempts than a conventional wordlist. Widening this scope, it would be preferable if it were also able to crack more passwords than other wordlists.

3.2.2 Language & Libraries

The script will be written in Python 3 due to its vast community support and external libraries, which are useful for quickly creating more efficient and robust code. This section will provide an overview of the libraries required in the development stage of this methodology.

The Gensim library, as discussed in the previous chapter, can be used to filter documents and train natural language models (Řehůřek & Sojka, 2011). The library allows the developer to pre-process documents using a simple pre-processing function or by using a custom filter. By default, the

"simple_preprocess()" function converts all words to lowercase and tokenises each word, but also contains functionality to de-accent letters and convert each string to Unicode (Řehůřek, 2019). The library also includes algorithms for training and loading FastText and Word2Vec models, including Word2Vec models generated using Google's original C implementation.

Another library which will be used is the Python regular expressions library (Python Software Foundation, n.d). This powerful library can be used to apply regular expression functions to strings in a Python script, which can be used to filter numbers and special characters from strings.

The Natural Language Toolkit (NLTK) is a useful natural language processing library. It contains an extensive set of tools and corpora which can be used to train and test natural language models, making it the go-to library for many writing Python scripts utilising natural language processing.

3.2.3 Natural Language Processing Model

The first step of development is to choose an appropriate machine learning algorithm. Options considered could include Word2Vec CBOW, SG, and FastText, all of which are discussed in the previous chapter. More advanced models, such as Google's Bi-directional Encoder Representations from Transformers (BERT) (Devlin, et al., 2019), are likely over-complex for this project and contain additional features and overhead which will make the script more demanding to run.

Test models can be trained using the Gensim Python library in an integrated development environment (IDE) of personal choice. The test models use a corpus of tweets which was initially used in a 2010 paper to determine if a user can be located based on the context of their tweets (Cheng, et al., 2010). The body of the tweet should be extracted from any metadata and then filtered for tweets containing English only characters (A-Z, 0-9, and special characters). Words such as usernames and hashtags, which start with the at symbol and hashtag symbol accordingly, should also be filtered from the tweet to prevent them potentially being added to the model. This filter can be created using the regular

expressions library for Python. To test the models, they can be queried for the top 10 nouns in the English language and a collection of words commonly found in passwords (Dictionary.com, n.d.) (Miessler, 2018). A complete list of the words used can be found in Appendix B – Test Words. The results given by each model can then be analysed and assessed by the researcher to determine the most suitable model for the final script.

The final model is then trained using the optimal settings determined from testing. By using multiple corpora, the model can gain a wider understanding of formal and informal language and slang words. The first corpus is based on a database of Reddit comments from May 2015, which were stored in an SQLite Database (Kaggle, 2019). By selecting the body of these comments and filtering out deleted comments using an SQL query, comment bodies can be saved in a comma-separated value (CSV) format using an SQLite database viewer. Non-Latin character comments, URL's, and words starting with "*/u"* or "*/r"*, which refer to usernames and subreddit names accordingly, can then be filtered out using regular expressions to prevent the model from potentially suggesting them. This corpus gives the model examples of common slang and abbreviations used on the internet and how they correlate to words in the English dictionary.

The second corpus is the Westbury Labs Wikipedia corpus (Shauol & Westbury, 2010). This corpus contains a snapshot of all English articles over 2000 characters from Wikipedia created before 2010 with all irrelevant material such as external and navigation links removed. This corpus provides a more formal set of documents with correct grammar and spelling to the model.

If using a Word2Vec model, the model can also be used to strip a password to its *"base word"* by querying the model for each possible word when removing numbers and character substitutions. An example of this would be stripping the password *"p4ssword24"* back to the base word of *"password".* If the model does not return a result, this means the word was not in the training corpora and can be skipped.

Each of the corpora can be filtered using Gensim's pre-processing tools. The *"simple_preprocess"* tool can be used to quickly convert each word to lowercase and tokenise each string, or a custom filter can be used. By using a custom filter, each document can be stripped of punctuation, numbers, and stop-words (Řehůřek, n.d). This will help improve model accuracy and reduce the time required to train the model.

Stop-words filtered out when training the model should also be filtered out when querying the model. This can be done using the NLTK corpus of English stop-words. These can be imported into a set, which will allow the script to efficiently check if each word in the passphrase is contained in the stop-words set.

3.2.4 Number Mutations

Number mutations are common numbers frequently seen at the start or end of passwords to increase complexity or meet password rules. To efficiently apply number mutations to password, the most frequently used numbers must be found. Using the *"darkweb-top-10000"* wordlist, each password can be processed using a Python script which extracts numbers that occur at the start and end of strings (Miessler, 2019). By counting the occurrence of each number, these can then be sorted from most to least popular and stored in a CSV file, a copy of which can be found in Appendix C – Number Occurrences. Any number which occurred less than 5 times in this wordlist is not added to the output file to keep the final custom wordlist short, which will optimise attack efficiency.

```
for password in wordlist:
        number = filter_regex(\d+)
                if number not in list_of_numbers:
                        append_to_list(number)
                else:
                        increment_occurence_of_number(number)
```

Figure 3 - Number Occurrences Pseudocode

3.2.5 Character Mutations

A character mutation is a common substitution used to replace the original character – for example, replacing the letter *"E"* with the number *"3"*. To

mutate characters in passwords, a Python dictionary containing a character and relevant mutations can be used to mutate each password character.
Common password mutations can be found by analysing survey results and popular passwords. Daniel Miessler also has a list of password mutations using *"leet speak"*, a spelling system of modified spellings used primarily on the internet for informal communications, available on GitHub (Miessler, 2017) (Dictionary.com, n.d). A function can then be created to both apply and reverse mutations and generate a list of potential passwords at each position the original character occurs. To cover all potential mutations, each mutation should also be applied to all previously generated mutations.

```
if length(word) < 16:
        max_words = 2500

for character in word:
        if character has mutations:
                add_mutations_to_list

        positions = get_positions(character)
        for position in positions:
                apply_mutation_at_position
                apply_mutation_at_each_position_forward
                apply_to_previous_mutations

        if items_generated > max_words:
                return
```

Figure 4 - Character Mutation Pseudocode

For words over 16 characters in length, a limit is applied to the number of words generated by the function. This is to prevent the script from generating wordlists too long to use while also helping to keep system resource usage relatively low. Without this limit, especially long password will generate thousands or millions of combinations which will slow down the script substantially and consume a large amount of the host systems memory.

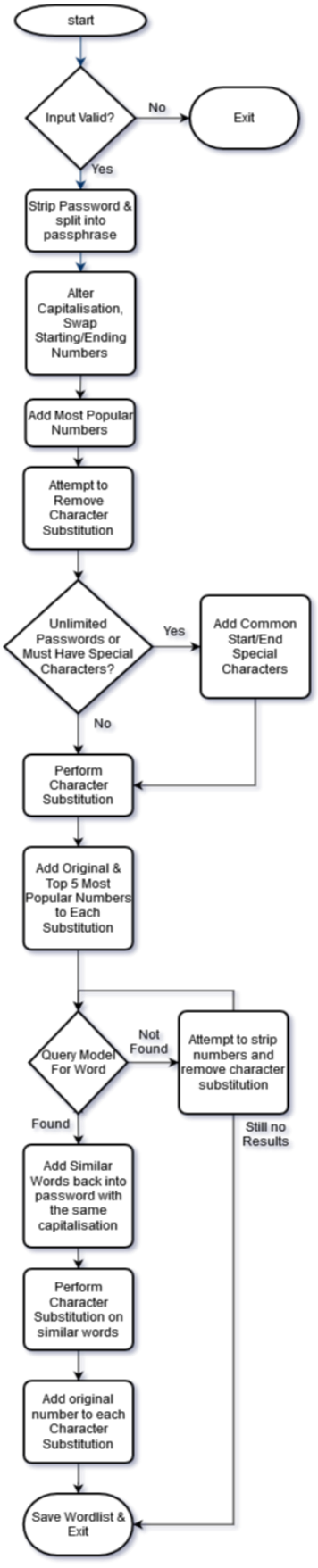

Figure 5 - Process Diagram of Script

By capitalising the letter at the start of each word in a password or passphrase and converting the password to full capital or lowercase letters, the most common capitalisations can be applied to keep the generated wordlist short. Optional settings can also be implemented to allow the user to choose to capitalise each letter or generate every possible password using capitalisation, however, this will substantially increase the number of passwords generated.

A full process diagram of the script to illustrate how each section functions within the script can be seen in Figure 5.

3.2.6 User Settings

Optional flags can also be implemented to improve ease-of-use for the user when running the script. These flags allow the user to specify password rules which the script must adhere to, such as specifying a password must have or must not include special characters. This will also help to further improve the efficiency of the wordlists generated by the scripts as they can eliminate passwords which cannot be used due to password rules.

3.2.7 Data Types

Initially, a set was used to store generated passwords, which is *"an unordered collection of hashable items"* (Python Software Foundation, 2020). By storing the generated passwords in a set, Python can automatically filter out any duplicate passwords which are generated, while also increasing efficiency when iterating through or

searching the wordlist. The drawback of this datatype is that items are not stored in the order which they were added, meaning the final wordlist is not in the order in which they were generated. As wordlist order is important for efficient wordlist generation, a list was used instead, which resulted in approximately 9% longer run times over 3 runs, as can be seen in Appendix E – List versus Set Performance.

3.3 Testing

To measure the efficiency of the wordlists generated by the script, which for an 8 character password is usually around 5000 passwords in length, it can be tested against other popular wordlists. The RockYou wordlist provides a suitable comparison for testing as it is a popular and comprehensive wordlist, comprising of 14,343,898 unique passwords. The second wordlist used was the top 304,000 Probable Passwords v2 dataset (berzerk0, 2019). This wordlist was constructed by analysing multiple password breaches and ordered passwords from most to least frequently occurring. The top 304,000 wordlist contains every password of the over 2 billion analysed that occurred more than 75 times during this analysis.

An appropriate testing set of passwords can be created using passwords from existing wordlists and survey data. Success is measured by the number of passwords cracked and the attempts taken compared to other wordlists. If the end of the password list is reached before the password is found, then the password can be marked as not found.

```
for line in testpasswords:
        if not line.startswith('-') and length(line) > 1:
                old_password = line.split(':')[0]
                new_password = line.split(':')[1]

                generate_wordlist_for(old_password)

                if new_password in custom_wordlist:
                        add line number to results
                else:
                        add 'NF' to results

                if new_password in RockYou:
                        add line number to results
                else:
                        add 'NF' to results

                if new_password in Probable:
                        add line number to results
                else:
                        add 'NF' to results
```

Figure 6 - Test Script Pseudocode

Performance testing of the script will also be completed. These tests will be run on a virtual machine, executed in succession immediately after boot. This will outline the system resource requirements of the script and its associated natural language processing model to generate customised wordlists. There are no success criteria for this testing other than a pass or fail dependant on if the script successfully generated a wordlist, however, time taken to execute will be measured and compared. This testing will ensure that the script can be run in low-resource environments such as virtual machines, where both memory and processing power can be very limited.

4 Results

This chapter will provide an analysis of results from testing the customised wordlists outputted from the Natural Language Passwords script to the popular RockYou wordlist and the top 304,000 Probable Passwords wordlist (berzerk0, 2019). Success is measured by how many passwords were cracked, and how many attempts were taken to successfully crack the password.

4.1 Wordlist Comparison Results

Testing was completed using default settings for the script. This meant that the script was generating as many passwords as possible with no password policy rules such as requiring numbers, special characters, or minimum length, using the top 5 most similar words from the natural language processing model.

Using the test set of 61 password pairs comprised of passwords submitted to the survey and passwords from wordlists chosen by the researcher, the script was given a fictional old password and tasked with cracking the new password. If the password was not found, then the result is marked with *"NF"* for *"Not Found"*. A copy of this password test set can be found in Appendix F – Password Test Set. The number of attempts was then compared to the number of attempts taken to find the new password in the RockYou and Probable Passwords wordlists.

The wordlist generated by the Natural Language Passwords script successfully cracked 26% more of the test data set compared RockYou, as can be seen in Table 1, with a median of just 0.132% of the attempts taken when both wordlists successfully cracked the password. It also cracked 45% more passwords from the test data set than the Probable Passwords wordlist. The raw data from testing can be seen in Appendix G – Test Results.

	Custom Wordlist	RockYou Wordlist	Probable Wordlist
Passwords Cracked	21	11	4
Cracked (%)	55.3	28.9	10.5
Average Attempts	1261	1715670	42560

Table 1- Processed Test Results

4.1.1 Trends in Result Data

As can be seen below in Figure 7, the customised wordlist generated by the script cracked the test set of passwords more reliably and in far fewer attempts than the other wordlists used for testing. The results are clear, once a targets password is known, or even if the attacker has researched interests and hobbies of the target, it becomes far easier to crack any passwords they use when a smarter approach is used by utilising natural language processing to help generate a customised wordlist against the target. Discussion on how the script settings could be tweaked to generate a more diverse wordlist can be found in section 5.4.

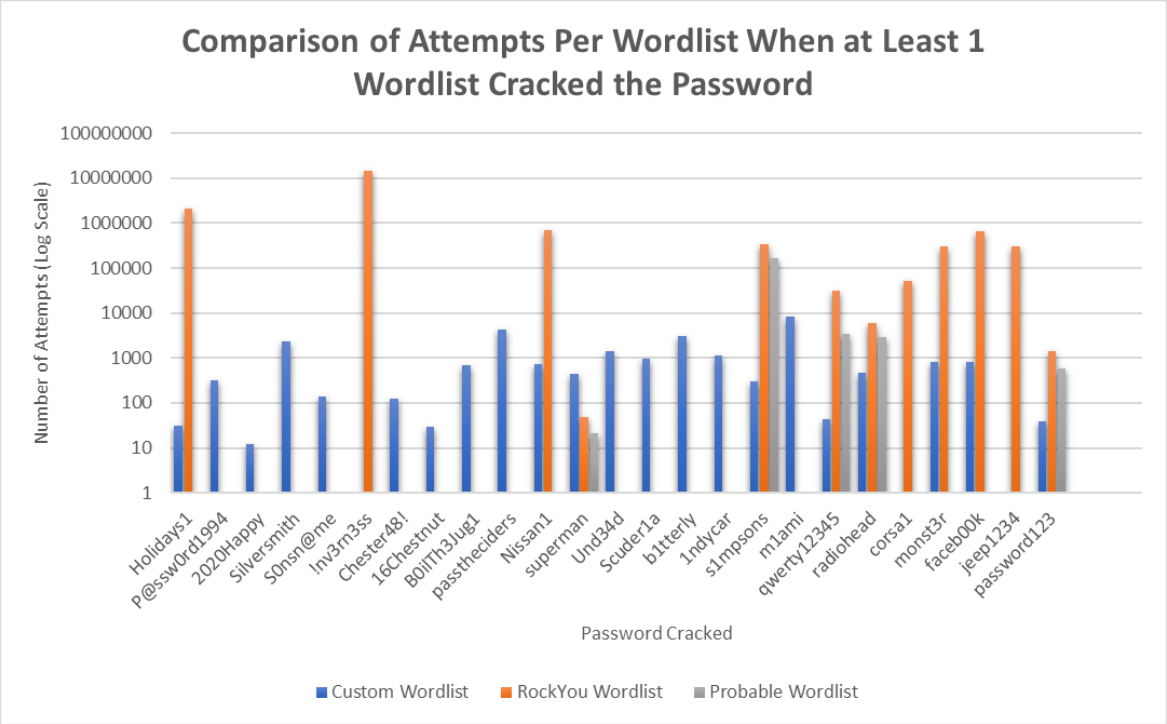

Figure 7 - Comparison of Attempts Per Wordlist When at Least 1 Wordlist Cracked the Password

4.2 Performance Testing

The performance was measured as an average over 5 runs executed in succession immediately after boot. Testing was completed on a virtual machine with 2 cores running PopOS_! 19.10, an Ubuntu-based Linux distribution.

With 6GB of RAM dedicated to the virtual machine, the script took on average 15 seconds to generate a wordlist for the password *"password123"* across 5 runs. This time increased to 19 seconds when the amount of RAM was reduced to 4GB due to the natural language processing model filling system memory and loading

into the page file, a file held on the systems hard disk to store items when no more RAM is available at the cost of substantially lower performance. This issue was further exacerbated when the amount of RAM was reduced to 2GB, after which the runtime of the script increased substantially to 102 seconds on average, as can be seen below in Figure 8. It was also observed that the low memory of the system resulted in the script taking substantially longer to save the wordlist, which is likely also caused by the variable used to hold the wordlist being loaded into the page file. Raw data from performance testing can be seen in Appendix H – Performance Metrics.

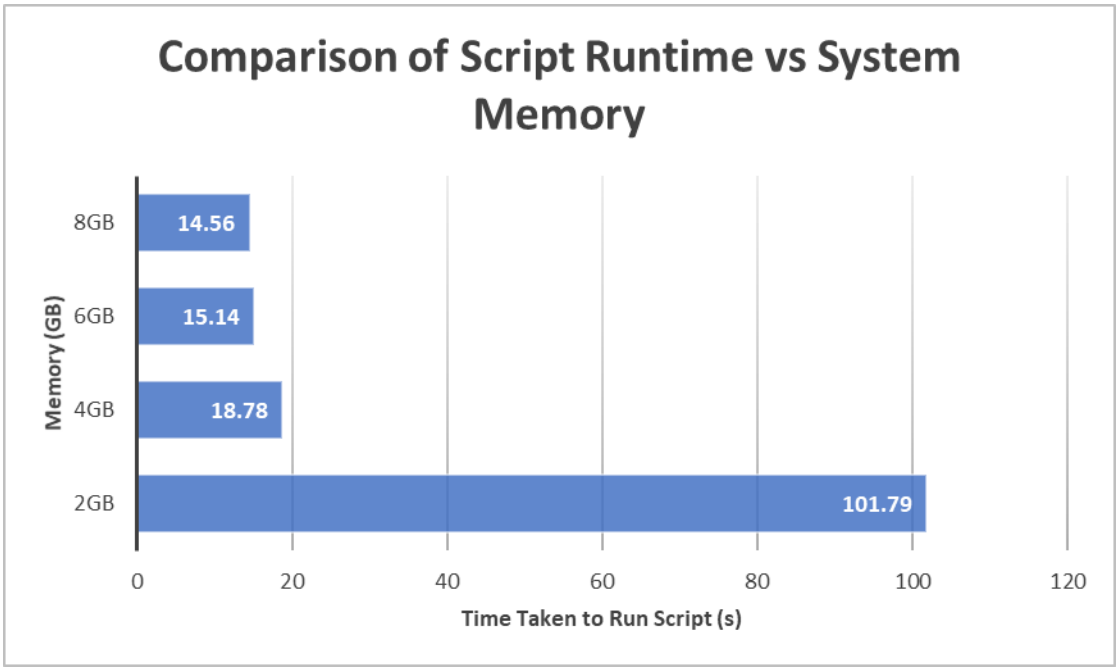

Figure 8 - Comparison of Script Runtime vs System Memory

5 Discussion

This chapter will discuss the rationale behind the settings used to train the natural language processing model, before moving on to analyse survey results, discussing challenges faced during development, and analysis of any abnormalities in test results. The chapter then rounds off with a discussion on how to improve password security and demonstrates how easily short passwords can be cracked using modern graphics cards.

5.1 Natural Language Model

Each of the trained test models were queried for the test set of words, as described in the previous chapter, section 3.2.3. Upon analysis of these results, it became clear that the FastText algorithm would not be suitable for use with an unfiltered dataset as it rated common misspellings as the most similar words. Its design also allows it to guess the vector of out-of-dictionary words, meaning it would always return a result and therefore could not be used to test words stripped of numbers and special characters for a base word.

Testing the Word2Vec models using various settings and based on results from previous papers, it became clear that the SG algorithm would be the most suitable for the script. This is supported by Mikolov's findings, which found that it performed comparably to the CBOW algorithm in syntactic tasks and far better in semantic tasks, as shown below (Mikolov, et al., 2013b). The SG algorithm also appeared to recognise the names of brands and towns more reliably than CBOW based on researcher observations. Outputs from testing as described in the previous chapter can be seen in Appendix D – NLP Test Outputs.

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
3 epoch CBOW	300	783M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783M	50.0	55.9	53.3	3
1 epoch CBOW	300	783M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783M	56.7	54.5	55.5	2.5

Figure 9 - Comparison of Models Trained on the Same Data

The final model was trained to discard any words that occurred less than 10 times, using Hierarchical SoftMax sampling, and a size of 150. The chosen sampling method meant that while training time was significantly higher, the model would be more likely to accurately identify uncommon words in the corpora. The higher dimensionality means that the model should provide more accurate results when queried for the most similar words with such a large set of text to be trained on, without increasing the models size significantly (Pennington, et al., 2014). On a system with an AMD Ryzen 3600 processor utilising 12 threads and 16GB of RAM, the model took approximately 10 hours to train.

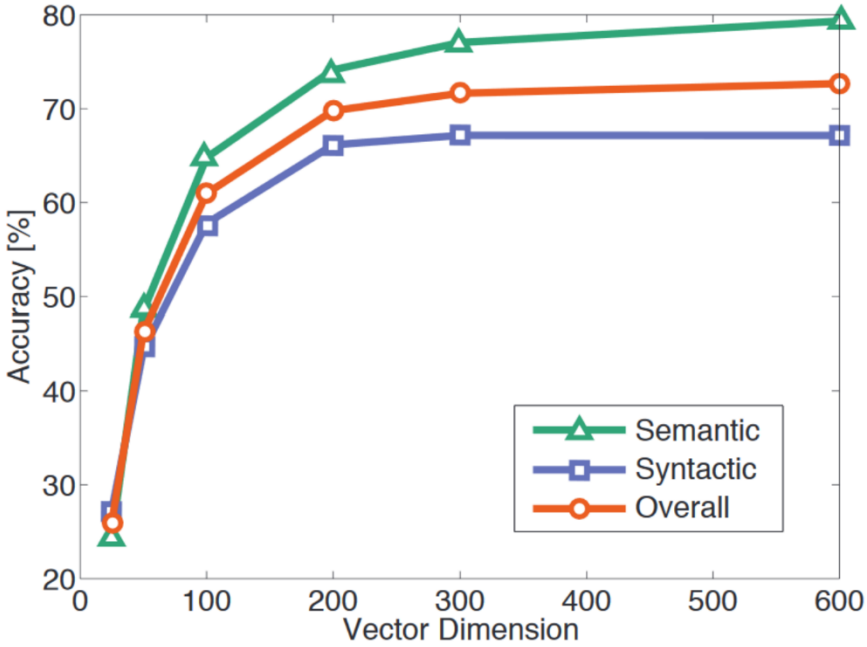

Figure 10 - Accuracy of Model vs Vector Dimension (Size)(Pennington, et al., 2014)

5.2 Survey Results

The survey had a total of 59 participants, 58 of which answered all questions. Most participants were 18-24 years old and rated themselves as technically competent. Two-thirds of participants said they used completely different or random passwords for important online accounts such as bank accounts, suggesting people are understanding the risk data breaches may pose to them as individuals. This is further supported in a following question in which 84.7% of participants said that they believed data breaches pose a threat to them individually. 88.2% of participants also said they create completely new or

random passwords when notified of a data breach on a website they have an account with.

When it came to accounts which participants deemed less important such as social media accounts however, over half of participants admitted to using the same or similar passwords across multiple accounts. Over 54% also admitted to not using any type of password management service, including web-browser auto-fill services, the majority of whom rated themselves as technically competent. This data suggests that while people will change their passwords when notified of a data breach, it leaves many of their other online accounts which used the same or similar passwords vulnerable, allowing an NLP-based password cracker to potentially crack the passwords of other accounts.

Technical Competency	**Number of Participants That Did Not Use a Password Manager**
Basic	6
Competent	24
Advanced	2

Table 2 - Number of Survey Participants That Did Not Use a Password Manager, Sorted by Technical Competency

The survey was distributed using the social media of the researcher, consequently, most respondents were likely students with interests in cyber security as this is the main demographic of people who follow the researcher on Twitter. This is backed up by 44% of participants being aged 18-24, meaning they are likely students. This, combined with the relatively small sample size of the survey, means that the data could be biased in favour of better security practices. In an effort to combat this bias the survey was also distributed on the researcher's personal Facebook, as well as among a class of students with varying academic backgrounds taking an Ethical Hacking elective at Abertay University.

5.2.1 Survey Password Trends

Almost a quarter of passwords submitted to the survey were pseudo-random or consisted of long passphrases made of random words. This again could be caused by the cyber-security background of several participants, however, does show that password practises are improving and would render a dictionary attack nigh on impossible.

Analysis of non-randomly generated passwords submitted to the survey shows that people often follow the same patterns when creating new passwords. This pattern may be a name of a person or town followed by numbers, or the same number moved from the start to the end of their password. Perhaps the most interesting finding was that people tended towards favouring the same capitalisation style of their passwords. Those who submitted completely lower or upper-case passwords, or those who capitalised the first letter of the password or each word in a passphrase, tended to stick to this pattern throughout their answers. This was used in the script to prioritise passwords which stick to similar capitalisation styles of the original password to improve efficiency when using a wordlist generated by the script.

5.3 Testing

When factoring in password attack mitigations, discussed in section 2.4 - Password Attack Mitigations, the efficiency of the wordlists generated could help to drastically reduce the amount of time spent on password cracking – requiring just a hundredth of the attempts in some scenarios.

However, one challenge faced in testing was that there is no appropriate test data which explicitly shows the evolution of peoples' passwords which exists at the time of writing. This meant the researcher had to combine survey answers with appropriate passwords chosen from wordlists that were deemed as likely alternative passwords based on research into the topic. Should such a dataset be created, it could be used to improve and test the script more thoroughly.

Another challenge faced was that there are no comparable tools. Tools such as CeWL, as covered in chapter 2, can create customised wordlists based on website data – but there are currently no mainstream tools to create wordlists based on previous passwords. Such a tool did once exist, known as the Associative Word List Generator (AWLG) – this tool would use search engines such as Google to query for a subject and generate a list of words based on its findings (Darknet, 2015). However, very little documentation on this tool has been created since its

creation in 2010 and according to the Internet Archive's Wayback Machine, the host domain has been offline since approximately 2013.

5.4 Result Abnormalities

Most passwords were cracked in far fewer attempts using the script compared to using RockYou, apart from *"superman"* which was cracked in 48 attempts by RockYou but requiring 431 in the custom wordlist when given the old password *"b4tman"*. This occurred as *"superman"* is a common and simple password with no numbers or special characters while being a completely different word compared to the original password. Since the wordlist generated by the script first prioritises character substitution and adding popular numbers, the new password was added later in the list, not accounting for it being a more common password.

The password *"Ab3rd33n123"* was not cracked by the script given the original password *"Dundee12"*. This happened because the script, in order to keep wordlists relatively short, does not add all frequently occurring numbers to every substitution. Instead, the script only adds any numbers from the end of the original password which in this case is 12. This means that some passwords will inevitably be missing from the password list, otherwise, the generated wordlists would contain too many passwords to be of use.

The password *"!nv3rn3ss"* also was cracked by RockYou but not by the script when supplied with the original password of *"U11apool000"*. This password was not generated by the script as it does not class *"Ullapool"*, a town in Scotland, as similar enough to *"Inverness"*, which is a city. Sometimes this can be addressed by increasing the number of similar words using the *"num-of-words"* flag, such as the case with the password *"Carlsberg94"*. *"Carlsberg"* is the 8th most similar word compared to the original password base word *"Tennents"*, however, by default the script only uses the 5 most common words given by the model.

5.5 Creating Secure Passwords

One of the key reasons this tool was created is to educate users on the potential implications a data breach or phishing campaign can have on them even if they only use similar passwords across accounts. As such, it is appropriate to recommend some password practises to keep accounts secure.

5.5.1 Password Managers

Mainstream password managers such as LastPass, the most popular password manager from the project survey with 20.3%, allow users to generate random passwords and specify the rules these must follow. They will then store these using encryption that is nigh on impossible to crack and automatically fill out login forms in websites once the user logs in with their master password. Some also offer services which automatically change passwords for services if a data breach is reported or at a scheduled interval (Gott, 2014). However, these still present the issue of the user must have at least 1 secure password that is not used anywhere else, and some services (such as logging into work accounts) may not support password managers.

5.5.2 Creating a Secure Password

Instead of creating a password, users should focus on creating a passphrase or sentence. Common phrases from books, films, and songs should also be avoided, and numbers and special characters will help to further the resistance of the password to brute-forcing. No password is invulnerable to hacking, and as such passwords should be changed every so often. The time interval between each change can increase dependant on the longer the password is, as the effort required to crack a password increases exponentially with length. This is illustrated in Figure 11 which shows the maximum number of hours it would take to crack a password, consisting of all 96 printable characters on the UK QWERTY keyboard, on a single AMD RX 480 GPU - a mid-range graphics card released in 2016. Researchers and attackers alike tend to use multiple high-end graphics cards in offline brute-force attacks which significantly reduces the time required, meaning they can crack passwords far faster than shown below. The time required could be drastically reduced using a dictionary attack,

theoretically even more so using the Natural Language Passwords tool assuming the targets password is not random or pseudo-random.

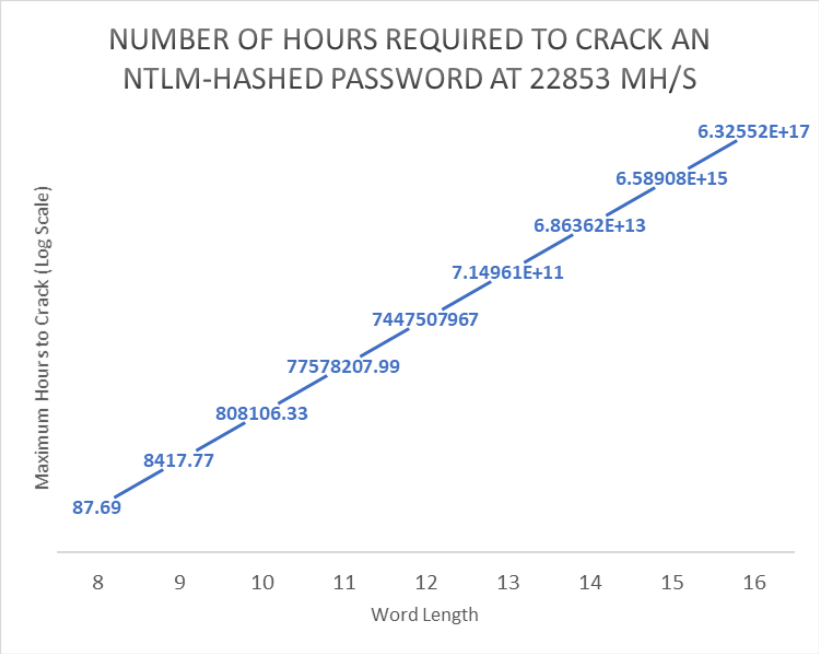

Figure 11 - Number of Hours Required to Crack an NTLM-Hashed Password

5.5.3 Multi-Factor Authentication

Wherever possible, users should be encouraged or forced to utilise multi-factor authentication tools. These are tools which send a one-use code to users through e-mail, SMS, or through a smartphone application which they must enter before being logged into their account. By enabling this feature, if a password is cracked and an attacker tries to log in, the user will be notified of a new login to their account immediately which will allow them to act swiftly to secure their account without the attacker successfully logging in. The services which utilise an application such as Authy or Google Authenticator can usually be set up by simply scanning a QR barcode and are inter-compatible, meaning it does not matter which application the user chooses to use.

6 Conclusion

To conclude, this project has successfully shown that it is possible to generate an efficient customised wordlist by utilising natural language processing models and common password mutation techniques to predict a target's password – provided their password is not pseudo-random or randomly generated by a computer. The project has also provided an insight into how passwords are commonly chosen and mutated, and the current understanding of cyber-security issues relating to passwords through a survey. The Natural Language Passwords tool could be used by penetration testers during security testing or by educational bodies to illustrate the potential implications a data breach can have on users using non-random passwords.

Regarding the research questions mentioned during the introductory chapter, this project has proven that natural language processing can be used to create a more diverse customised wordlist when a password or interest of the target is known. The wordlists often contained passwords that otherwise might not have been tested. Even using a relatively simple algorithm, Word2Vec, the trained model was able to retrieve the most similar words to a given word using word vectors with surprising accuracy using the settings outlined in the previous chapter. Utilising the Gensim Python library, training the model was easy and the final model was both memory efficient and quick to execute query operations.

By analysing the survey data, clear patterns and preferences could be identified when looking at how people create passwords. It was found that people often iterated their passwords using character substitutions and switching the position numbers or special characters found at the start or end of their password. The survey also found that people commonly stuck to the same capitalisation technique throughout all passwords supplied to the survey, such as typing passwords in all lower or upper case, or by capitalising the first letter of the password or each word in a passphrase.

Finally, the proof-of-concept script proved that it is possible to use these factors, in conjunction with a natural language processing model, to generate a more

reliable and efficient wordlist for a dictionary attack when compared to RockYou and the Probable Passwords data sets during testing. The wordlist generated by the script managed to crack 26% more password than the RockYou wordlist and 45% more than the Probable Passwords wordlist in significantly fewer attempts. Some passwords that were not found could be found by tweaking a variety of settings to expand the number of passwords generated by the script. The obvious limitation of this approach to cracking passwords is that a previous password must be known to generate a list of potential passwords. To combat this, a security researcher or penetration tester may wish to enumerate hobbies or interests of the target using OSINT techniques and then build a wordlist based on their findings by tweaking the script settings to include more words from the NLP model at the expense of generating a longer wordlist.

By using a combination of corpora from Westbury Labs' Wikipedia corpus and a filtered collection of Reddit comments, the NLP model had a broad enough understanding of human language to be useful when cracking passwords without adding so much overhead that it would be slow or unable to run on environments with lower memory, such as a virtual machine. Although, when executed on a system with less than 4GB of RAM, the script still executed but with significantly longer runtime due to the NLP model loading into the system's page file.

The results of this project show how even a simple natural language processing algorithm such as Word2Vec can be used to create a more varied password wordlist using word vectors to retrieve the most similar words to a given word. As research advances, it may be possible to add further enhancements which will improve model accuracy and thereby wordlist reliability, or even use a new algorithm entirely which could be more suited to this type of application.

6.1 Future Work

With advances in technology and machine learning occurring regularly, it may be possible to create a more appropriate machine learning model from new algorithms for such an application. Improvements could also be made by utilising supervised learning to improve the accuracy of the Word2Vec model or using

spelling checked corpora to train an appropriate FastText model while retaining an understanding of common shorthand expressions and internet slang.

Further research could also be carried out into features typically found in passwords to enhance the efficiency of generated wordlists and to add features which generate passwords that previously are not generated by the script with its current feature set.

7 References

1Password, 2019. *How PBKDF2 strengthens your Master Password.* [Online]
Available at: https://support.1password.com/pbkdf2/
[Accessed 10 March 2020].

berzerk0, 2019. *Probable Wordlists - Version 2.0.* [Online]
Available at: https://github.com/berzerk0/Probable-Wordlists
[Accessed 9 March 2020].

Bischoff, P., 2020. *US property and demographic database of 200 million records leaked on the web.* [Online]
Available at: https://www.comparitech.com/blog/vpn-privacy/200-million-us-database-leaked/
[Accessed 22 April 2020].

Blocki, J., Harsha, B. & Zhou, S., 2018. On the Economics of Offline Password Cracking. In: *2018 IEEE Symposium on Security and Privacy (SP).* San Francisco: IEEE, pp. 853-871.

Bojanowski, P., Grave, E., Joulin, A. & Mikolov, T., 2017. *Enriching Word Vectors with Subword Information.* [Online]
Available at: https://arxiv.org/pdf/1607.04606.pdf
[Accessed 9 March 2020].

Burnett, M., 2006a. Is Random Really Random?. In: D. Klieman, ed. *Perfect Passwords.* Rockland: Syngress Publishing, pp. 26-37.

Burnett, M., 2006b. Password Length: Making It Count. In: D. Kleiman, ed. *Perfect Passwords.* Rockland: Syngress Publishing, pp. 60-61.

Burnett, M., 2006c. Time: The Enemy Of All Secrets. In: D. Kleiman, ed. *Perfect Passwords.* Rockland: Syngress Publishing, pp. 70-71.

Burnett, M., 2006d. *Perfect Passwords.* Rockland: Syngress.

Cheng, Z., Caverlee, J. & Lee, K., 2010. *You Are Where You Tweet: A Content-Based Approach toGeo-locating Twitter Users.* [Online]
Available at: http://faculty.cs.tamu.edu/caverlee/pubs/cheng10cikm.pdf
[Accessed 25 January 2020].

Cubrilovic, N., 2009. *RockYou Hack: From Bad To Worse.* [Online]
Available at: https://techcrunch.com/2009/12/14/rockyou-hack-security-

myspace-facebook-passwords/
[Accessed 9 March 2020].
CySecurity, 2011. *What is the Difference between Brute Force vs Dictionary Attack.* [Online]
Available at: http://breakthesecurity.cysecurity.org/2011/05/what-is-the-difference-between-brute-force-vs-dictionary-attack.html
[Accessed 22 April 2020].
Darknet, 2015. *The Associative Word List Generator (AWLG) – Create Related Wordlists for Password Cracking.* [Online]
Available at: https://www.darknet.org.uk/2009/01/the-associative-word-list-generator-awlg-create-related-wordlists-for-password-cracking/
[Accessed 17 April 2020].
Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K., 2019. *BERT: Pre-training of Deep Bidirectional Transformers forLanguage Understanding.* [Online]
Available at: https://arxiv.org/pdf/1810.04805.pdf
[Accessed 6 April 2020].
Dictionary.com, n.d. *What are the most-used words in English?.* [Online]
Available at: https://www.dictionary.com/e/commonwords/
[Accessed 16 February 2020].
Dictionary.com, n.d. *leetspeak.* [Online]
Available at: https://www.dictionary.com/browse/leetspeak
[Accessed 3 April 2020].
Gott, A., 2014. *Introducing Auto-Password Changing with LastPass.* [Online]
Available at: https://blog.lastpass.com/2014/12/introducing-auto-password-changing-with.html/
[Accessed 24 April 2020].
Henriquez, M., 2019. *The Top 12 Data Breaches of 2019.* [Online]
Available at: https://www.securitymagazine.com/articles/91366-the-top-12-data-breaches-of-2019
[Accessed 22 April 2020].
Horsch, M., Schlipf, M., Braun, J. & Buchmann, J., 2016. Password Requirements Markup Language. In: J. Liu & R. Steinfeld, eds. *Information Security and Privacy*. Melbourne: Springer International Publishing Switzerland, pp. 426-439.

Identity Theft Resource Center, 2019. *2019 End-of-Year Data Breach Report,* s.l.: s.n.

Kaggle, 2019. *May 2015 Reddit Comments.* [Online]
Available at: https://www.kaggle.com/reddit/reddit-comments-may-2015
[Accessed 25 January 2020].

Kelley, P. G. et al., 2011. *Guess Again (and Again and Again): Measuring Password Strength by Simulating Password-Cracking Algorithms.* [Online]
Available at: https://www.cylab.cmu.edu/_files/pdfs/tech_reports/CMUCyLab11008.pdf
[Accessed 9 March 2020].

Kotadia, M., 2004. *Gates predicts death of the password.* [Online]
Available at: https://www.cnet.com/news/gates-predicts-death-of-the-password/
[Accessed 22 April 2020].

Kuranda, S., 2016. *The 10 Biggest Data Breaches Of 2016.* [Online]
Available at: https://www.crn.com/slide-shows/security/300083246/the-10-biggest-data-breaches-of-2016.htm/11
[Accessed 22 April 2020].

Lennon, M., 2010. *Survey Reveals How Stupid People are With Their Passwords.* [Online]
Available at: https://www.securityweek.com/survey-reveals-how-stupid-people-are-their-passwords
[Accessed 16 April 2020].

Leskin, P., 2018. *The 21 scariest data breaches of 2018.* [Online]
Available at: https://www.businessinsider.com/data-hacks-breaches-biggest-of-2018-2018-12?op=1&r=US&IR=T
[Accessed 22 April 2020].

Lord, N., 2017. *The Top 10 Biggest Data Breaches of 2015.* [Online]
Available at: https://digitalguardian.com/blog/top-10-biggest-data-breaches-2015
[Accessed 22 April 2020].

Miessler, D., 2017. *SecLists/Passwords/Permutations/1337speak.txt.* [Online]
Available at: https://github.com/danielmiessler/SecLists/blob/master/Passwords/Permutations/1337speak.txt
[Accessed 4 March 2020].

Miessler, D., 2018. *10-million-password-list-top-500.txt.* [Online]
Available at:
https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-500.txt
[Accessed 16 February 2020].
Miessler, D., 2019. *SecLists/Passwords/darkweb2017-top10000.txt.* [Online]
Available at:
https://github.com/danielmiessler/SecLists/blob/master/Passwords/darkweb2017-top10000.txt
[Accessed 6 March 2020].
Mikolov, T., Chen, K., Corrado, G. & Dean, J., 2013a. *Efficient Estimation of Word Representations in Vector Space.* [Online]
Available at: https://arxiv.org/pdf/1301.3781.pdf
[Accessed 9 March 2020].
Mikolov, T., Chen, K., Corrado, G. & Dean, J., 2013b. Comparison of models trained for three epochs on the same data and models trained forone epoch. Accuracy is reported on the full Semantic-Syntactic data set. In: *Efficient Estimation of Word Representations inVector Space.* s.l.:s.n., p. 8.
Novinson, M., 2017. *The 10 Biggest Data Breaches Of 2017.* [Online]
Available at: https://www.crn.com/slide-shows/security/300096951/the-10-biggest-data-breaches-of-2017.htm/11
[Accessed 22 April 2020].
Offsec Services, n.d. *crunch Package Description.* [Online]
Available at: https://tools.kali.org/password-attacks/crunch
[Accessed 9 March 2020].
OWASP Foundation, 2020. *Blocking Brute Force Attacks.* [Online]
Available at: https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks
[Accessed 10 March 2020].
Pennington, J., Socher, R. & Manning, C., 2014. The GloVe Model. In: *GloVe: Global Vectors for Word Representation.* s.l.:s.n., p. 7.
Python Software Foundation, 2020. *Set Types.* [Online]
Available at: https://docs.python.org/3.8/library/stdtypes.html#set-types-set-

frozenset
[Accessed 12 February 2020].
Python Software Foundation, n.d. *Regular Expression HOWTO.* [Online]
Available at: https://docs.python.org/3/howto/regex.html
[Accessed 14 April 2020].
Řehůřek, R., 2019. *gensim.utils.simple_preprocess.* [Online]
Available at: https://radimrehurek.com/gensim/utils.html
[Accessed 14 April 2020].
Řehůřek, R., n.d. *parsing.preprocessing – Functions to preprocess raw text.* [Online]
Available at: https://radimrehurek.com/gensim/parsing/preprocessing.html
[Accessed 14 April 2020].
Řehůřek, R. & Sojka, P., 2011. *Gensim—Statistical Semantics in Python.* [Online]
Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.695.4286&rep=rep1&type=pdf
[Accessed 9 March 2020].
Schneier, B., 2000. The Human Factor. In: *Secrets and Lies: Digital Security In A Networked World, 15th Anniversary Edition.* Indianapolis: John Wiley & Sons, Inc., pp. 255-269.
Shauol, C. & Westbury, C., 2010. *The Wesbury Lab Wikipedia corpus (2010).* [Online]
Available at: http://www.psych.ualberta.ca/~westburylab/downloads/westburylab.wikicorp.download.html
[Accessed 25 February 2020].
Verizon, 2019. *2019 Data Breach Investigations Report,* s.l.: s.n.
Vickery, R., 2019. *Securi-Tay 2019: Using Natural Language Processing Techniques To Crack Passwords - Robin Vickery.* [Online]
Available at: https://youtu.be/YJfYWuTzHIQ
[Accessed 6 March 2020].
Weidman, G., 2014. Password Attacks. In: A. Law, ed. *Penetration Testing: a Hands-on introduction to Hacking.* San Francisco: No Starch Press, pp. 199-201.

Wood, R., 2016. *CeWL - Custom Word List generator*. [Online]
Available at: https://digi.ninja/projects/cewl.php
[Accessed 9 March 2020].

8 Appendices

Appendix A – Largest Breach by Year

2015 – 191 million affected (Lord, 2017)

2016 – 1 billion affected (Kuranda, 2016)

2017 – 143 million affected (Novinson, 2017)

2018 – 1.1 billion (Leskin, 2018)

2019 – 885 Million (Henriquez, 2019)

2020 (So Far) – 200 Million (Bischoff, 2020)

Appendix B – Test Words

time

person

year

way

day

thing

man

world

life

hand

password

dragon

baseball

superman

secret

computer

shadow

master

Appendix C – Number Occurrences

Number	Occurrences
1	2260
123	456
2	252
12	218
3	129
123456	89
4	88
11	71
13	68
5	67
7	66
12345	66
1234	47
23	41
01	40
9	39
10	39
123456789	38
8	35
22	33
6	31
69	28
21	25
14	25
0	21
666	17
101	17
15	16
09	16
08	15
16	14
1234567	13
24	12
18	12
07	11
420	8
20	8
19	8
17	8
123123	7
12345678	6
88	5
777	5
77	5
55	5
321	5
1234567890	5
123321	5
111111	5
11111	5
111	5
007	5
00	5

Appendix D – NLP Test Outputs

i. Word2Vec Results

```
Word: time
Most similar:
[('way', 0.648678183555603), ('day', 0.6252382397651672),
('then', 0.5829129219055176), ('first', 0.5588756799697876),
('week', 0.5467789173126221), ('again', 0.5463563799858093),
('place', 0.5412317514419556), ('while', 0.5402042269706726),
('thing', 0.533696174621582), ('minute', 0.525600790977478)]

Word: person
Most similar:
[('people', 0.7161005735397339), ('some1',
0.7046341896057129), ('ppl', 0.6466778516769409), ('woman',
0.6425032615661621), ('thing', 0.6300520896911621), ('guy',
0.6257038712501526), ('sum1', 0.6107431650161743), ('someone',
0.6069379448890686), ('reason', 0.5989629626274109), ('no1',
0.583400547504425)]

Word: year
Most similar:
[('years', 0.7855926752090454), ('month', 0.740210235118866),
('yrs', 0.6925792098045349), ('decade', 0.6925646066665649),
('yr', 0.6922121047973633), ('5yrs', 0.6422821879386902),
('months', 0.6315730810165405), ('decades',
0.6254290342330933), ('week', 0.6219742298126221), ('days',
0.606185793876648)]

Word: way
Most similar:
[('time', 0.648678183555603), ('place', 0.6126850247383118),
('and', 0.5767316222190857), ('things', 0.5687756538391113),
('thing', 0.5679728984832764), ('far', 0.5610794425010681),
('places', 0.5573731064796448), ('ways', 0.5531235933303833),
('situations', 0.5441452264785767), ('but',
0.542748212814331)]

Word: day
Most similar:
[('weekend', 0.7264079451560974), ('week',
0.7111625671386719), ('today', 0.6672411561012268), ('time',
0.6252382397651672), ('afternoon', 0.6135393977165222),
('weekends', 0.6035586595535278), ('morning',
0.5877408385276794), ('evening', 0.5864670872688293),
('sunday', 0.581837415695190 4), ('days', 0.5714130401611328)]

Word: thing
Most similar:
[('things', 0.720630407333374), ('idea', 0.6604040861129761),
('one', 0.6341578960418701), ('reason', 0.6319546699523926),
('person', 0.6300519704818726), ('situation',
0.6099085807800293), ('but', 0.5999799966812134),
('everything', 0.5980021357536316), ('actually',
0.5885758399963379), ('probably', 0.5824246406555176)]

Word: man
```

```
Most similar:
[('woman', 0.7629948854446411), ('dude', 0.7317678928375244),
('mans', 0.6899942755699158), ('guy', 0.6719633340835571),
('girl', 0.5805158615112305), ('lady', 0.5770869851112366),
('boy', 0.5672240853309631), ('kid', 0.564288854598999),
('son', 0.5624884366989136), ('nigga', 0.560688853263855)]

Word: world
Most similar:
[('worlds', 0.7465701103210449), ('earth',
0.6589276790618896), ('life', 0.5731104612350464),
('universe', 0.5725148916244507), ('nations',
0.5510287284851074), ('america', 0.5301198959350586),
('wolrd', 0.5280536413192749), ('country',
0.5269243717193604), ('nation', 0.5209254622459412), ('lives',
0.5203665494918823)]

Word: life
Most similar:
[('lifes', 0.72854083776474), ('lives', 0.7147356271743774),
('friendships', 0.6914500594139099), ('relationship',
0.6491398215293884), ('ambition', 0.6252471804618835),
('relationships', 0.6200249791145325), ('knowing',
0.6184813976287842), ('situations', 0.6154676079750061),
('mindset', 0.614814281463623), ('circumstances',
0.6129956841468811)]

Word: hand
Most similar:
[('hands', 0.7277333736419678), ('arm', 0.7127677202224731),
('finger', 0.6627435088157654), ('eyebrow',
0.6516595482826233), ('sanitizer', 0.6445428133010864),
('forearm', 0.6319881677627563), ('elbow',
0.6224849224090576), ('thumb', 0.6098898649215698), ('napkin',
0.6051280498504639), ('nostril', 0.6035719513893127)]

Word: password
Most similar:
[('pw', 0.8374300599098206), ('passwords',
0.8367894291877747), ('username', 0.7758256793022156),
('settings', 0.7450624704360962), ('login',
0.7231239080429077), ('acct', 0.7052409052848816), ('account',
0.6679548025131226), ('hacked', 0.6553182601928711), ('gmail',
0.6487231254577637), ('pword', 0.6442725658416748)]

Word: dragon
Most similar:
[('origins', 0.6961507797241211), ('dragons',
0.5746784806251526), ('samurai', 0.5689435005187988),
('demon', 0.5566211342811584), ('xmen', 0.543816328048706),
('jungle', 0.5429788827896118), ('butterfly',
0.5428361892700195), ('monkey', 0.5382314324378967), ('sword',
0.5342062711715698), ('mutant', 0.530684769153595)]

Word: baseball
Most similar:
[('football', 0.7473307847976685), ('mlb',
0.7059385776519775), ('basketball', 0.703383207321167),
```

```
('cubs', 0.7008551359176636), ('hockey', 0.6982263326644897),
('league', 0.6789563298225403), ('fball', 0.676206648349762),
('sports', 0.6723508834838867), ('royals',
0.6708768606185913), ('soccer', 0.6678846478462219)]

Word: superman
Most similar:
[('batman', 0.7060868740081787), ('deadpool',
0.6630048751831055), ('spiderman', 0.6508046388626099),
('villain', 0.6389768123626709), ('joker',
0.6219347715377808), ('thor', 0.5733839273452759),
('thundercats', 0.5647870302200317), ('superhero',
0.5631474852561951), ('reeves', 0.5566315650939941),
('invincible', 0.5532715320587158)]

Word: secret
Most similar:
[('secrets', 0.7190839052200317), ('hidden',
0.5150153636932373), ('formula', 0.4873487949371338),
('reveal', 0.44558119773864746), ('uncover',
0.4393852949142456), ('attraction', 0.43880122900009155),
('purpose', 0.42713621258735657), ('revealing',
0.4251580536365509), ('discover', 0.4242616891860962),
('untold', 0.41781526803970337)]

Word: computer
Most similar:
[('laptop', 0.8093208074569702), ('computers',
0.7780020236968994), ('pc', 0.7621933817863464), ('cpu',
0.7046529054641724), ('router', 0.6864368915557861),
('desktop', 0.6802523136138916), ('modem',
0.6701671481132507), ('keyboard', 0.6671475172042847),
('reinstall', 0.6571248769760132), ('phone',
0.653498113155365)]

Word: shadow
Most similar:
[('shadows', 0.6449943780899048), ('spiral',
0.555783748626709), ('dark', 0.5483183264732361), ('darker',
0.5269689559936523), ('grim', 0.5174442529678345), ('gleam',
0.5146036148071289), ('claw', 0.513247013092041), ('liner',
0.5108810067176819), ('moonlight', 0.5093792676925659),
('gray', 0.5051147937774658)]

Word: master
Most similar:
[('mastery', 0.5658203363418579), ('jedi',
0.5426878929138184), ('perfecting', 0.5199397802352905),
('beginner', 0.5188783407211304), ('essential',
0.5098149180412292), ('composition', 0.5000872611999512),
('iii', 0.4952048063278198), ('basics', 0.4832035005092621),
('analytical', 0.48024865984916687), ('reinventing',
0.4781031906604767)]
```

ii. FastText Results

```
Word: time
Most similar: [('time2time', 0.9217326641082764), ('1time',
0.8843765258789062), ('ttime', 0.8679197430610657), ('2time',
0.8640760183334351), ('time4', 0.861491858959198), ('time2',
0.8590903282165527), ('cptime', 0.8245218992233276),
('itstime', 0.823288083076477), ('firsttime',
0.8123944401741028), ('nexttime', 0.8042607307434082)]

Word: person
Most similar: [('persone', 0.9216907620429993), ('personn',
0.9037349820137024), ('1person', 0.8875747919082642),
('personel', 0.8817798495292664), ('personnage',
0.8792098760604858), ('personeel', 0.8761124014854431),
('persons', 0.8746722936630249), ('personne',
0.8508207201957703), ('imthatperson', 0.844006359577179),
('sameperson', 0.8422012329101562)]

Word: year
Most similar: [('9year', 0.9570554494857788), ('3year',
0.9565057754516602), ('5year', 0.9562569856643677), ('6year',
0.9552361369132996), ('1year', 0.9542660713195801), ('8year',
0.9527437686920166), ('2year', 0.9499192833900452), ('4year',
0.9495000839233398), ('18year', 0.9037007689476013), ('yearz',
0.901720404624939)]

Word: way
Most similar: [('wayrt', 0.8079869151115417), ('way2',
0.8063687682151794), ('wayside', 0.7972642779350281),
('waybe', 0.7947698831558228), ('wayt', 0.7838908433914185),
('waying', 0.750969409942627), ('everyway',
0.7460411190986633), ('someway', 0.7435370683670044), ('3way',
0.7408084869384766), ('2way', 0.7275089621543884)]

Word: day
Most similar: [('day2day', 0.9102061986923218), ('dayday',
0.8944153785705566), ('day3', 0.8814074397087097), ('5day',
0.8794251680374146), ('6day', 0.875942587852478), ('4day',
0.8748103976249695), ('day4', 0.8736667633056641), ('7day',
0.8720911145210266), ('dday', 0.8714820742607117), ('t0day',
0.8708271980285645)]

Word: thing
Most similar: [('thingrt', 0.9208962917327881), ('thingd',
0.9131098985671997), ('soemthing', 0.9097210168838501),
('thingss', 0.9092444181442261), ('evything',
0.9085795879364014), ('1thing', 0.9083523750305176),
('smthing', 0.9050476551055908), ('onething',
0.9027617573738098), ('somwthing', 0.9017852544784546),
('nthing', 0.89905846118927)]

Word: man
Most similar: [('woman', 0.8047046661376953), ('yeoman',
0.7974767684936523), ('madwoman', 0.7915624976158142),
('manñ', 0.7748119831085205), ('batwoman',
0.7721189856529236), ('maná', 0.7677620649337769),
('catwoman', 0.7672019004821777), ('homan',
```

```
0.7648884057998657), ('realwoman', 0.7595033645629883),
('ooman', 0.756003737449646)]

Word: world
Most similar: [('worldy', 0.9570191502571106), ('worldd',
0.9204071760177612), ('vizworld', 0.8982473611831665),
('itworld', 0.8976403474807739), ('wdworld',
0.8954460620880127), ('worldly', 0.8949019908905029),
('mpworld', 0.8886868953704834), ('worldcon',
0.8873156905174255), ('inworld', 0.8872780203819275),
('vapeworld', 0.8858392238616943)]

Word: life
Most similar: [('life123', 0.8937889933586121), ('1life',
0.8826805949211121), ('lifey', 0.8755849599838257), ('lifes',
0.8724992871284485), ('4life', 0.8450573682785034), ('lifee',
0.8395850658416748), ('lifeofaboss', 0.820716142654419),
('lifemate', 0.818347692489624), ('truelife',
0.8154309988021851), ('life101', 0.812211275100708)]

Word: hand
Most similar: [('byhand', 0.8597666025161743), ('handz',
0.8554303646087646), ('shand', 0.8340389728546143), ('hands',
0.8338788747787476), ('handsme', 0.8322467803955078),
('handss', 0.8217894434928894), ('handknit',
0.8213232159614563), ('offhand', 0.8211010098457336),
('lefthand', 0.8128443956375122), ('handd',
0.8071460723876953)]

Word: password
Most similar: [('1password', 0.9813928008079529),
('passwords', 0.9482752084732056), ('pasword',
0.8260963559150696), ('passwd', 0.8083299398422241),
('passwrd', 0.8047693967819214), ('username',
0.7924185991287231), ('newcussword', 0.7879974842071533),
('pw', 0.7754055261611938), ('gmailfail', 0.7623568773269653),
('hackedu', 0.7470141649246216)]

Word: dragon
Most similar: [('gdragon', 0.9570474028587341), ('snapdragon',
0.9176701903343201), ('dragonfly', 0.9096890091896057),
('dragoncon', 0.9093260765075684), ('dragons',
0.9032028317451477), ('dragonage', 0.9003623723983765),
('dragoon', 0.894172191619873), ('dragonette',
0.8833568692207336), ('dragones', 0.8761351704597473),
('dragonblight', 0.8682958483695984)]

Word: baseball
Most similar: [('batsbaseball', 0.9453513622283936),
('fantasybaseball', 0.9309583306312561), ('baseballs',
0.9243345856666565), ('fooseball', 0.8853707909584045),
('dodgeball', 0.88434898853302), ('basebal',
0.8811429738998413), ('baskeball', 0.8772920966148376),
('baseballcards', 0.8769434690475464), ('dirtball',
0.8683424592018127), ('greaseball', 0.8680351972579956)]

Word: superman
```

```
Most similar: [('supermans', 0.8704308271408081), ('piderman',
0.8703302145004272), ('shimerman', 0.8679547905921936),
('imerman', 0.8571472764015198), ('starman',
0.8567804098129272), ('lerman', 0.8562725782394409),
('patman', 0.8535439968109131), ('wierman',
0.8530476093292236), ('kupperman', 0.8530313372612),
('halderman', 0.851237952709198)]

Word: secret
Most similar: [('secrete', 0.9236711263656616),
('tellmeasecret', 0.8932634592056274), ('secreto',
0.8875601291656494), ('secretive', 0.882801353931427),
('secrets', 0.8802083134651184), ('secre',
0.8751596212387085), ('blogsecret', 0.852465271949768l),
('secreted', 0.8512417078018188), ('secrecy',
0.8479081988334656), ('360dabestkeptsecret',
0.8475340604782104)]

Word: computer
Most similar: [('computerr', 0.9607019424438477),
('computeruser', 0.9507483243942261), ('compute',
0.948625385761261), ('computerless', 0.9413051605224609),
('computerrr', 0.9369449019432068), ('computers',
0.9284563064575195), ('computerlove', 0.9229307174682617),
('computerized', 0.9201228618621826), ('computes',
0.9167788028717041), ('supercomputer', 0.9088626503944397)]

Word: shadow
Most similar: [('shadowy', 0.955434262752533), ('shadowrun',
0.9063811898231506), ('shadows', 0.9036986827850342),
('shadowbox', 0.8726943731307983), ('foreshadow',
0.8639175891876221), ('shadowed', 0.8394898176193237),
('shadowroom', 0.8301563262939453), ('shadowpress',
0.8160924315452576), ('eyeshadow', 0.8112516403198242),
('shadowland', 0.809562087059021)]

Word: master
Most similar: [('masterpeice', 0.9497573375701904),
('mastery', 0.9482516050338745), ('masterplan',
0.9282899498939514), ('masterd', 0.9277714490890503),
('scrummaster', 0.909203827381134), ('loremaster',
0.9011173844337463), ('masterbation', 0.8996954560279846),
('remaster', 0.8960347175598145), ('bassmaster',
0.8933050632476807), ('funkmaster', 0.8928113579750061)]
```

Appendix E – List versus Set Performance

List:	Time(s)		
1	11.65		Average:
2	12.09		11.86
3	11.84		
Set:			
1	10.85		Average:
2	10.95		10.88667
3	10.86		
		Diff(%)	108.9406

Appendix F – Password Test Set

```
-from Survey
0KiU7FMWbwwn5qXu:0KiU7FMWbwwn5qXu
#8qeqz8Qp@wT2b:ptC8U%kQLTr&!w
Yinjbiovucy986:p8bbcicjnY5f
12Bl1ndMice$:M&mtwentyfour4#
234RampantLizards!:PiratesAreGone!76
Youpe3ple45:Defer3ytl
Ashtanga20:Change it
Cowdengelly:Cowden632
Ca10duenco&#;= pq8h3:Iwhdos7h22ody99&Ã—;=("Â£Ã·;'
TulipsforSpring2028:ManagemyPassword3991
Gartocharn1:Rhodes123
Freelands18300:QueryTime1066
Stewart191074:Gillian24011975
=@tpItLfaw:=@tpLdapaap
18Germany!!:19CroatiaÂ£@
ThankYou8:Yes12335
Winter19@!:Summer20##
Filledin:Infilled
4thyear2020!!:Smarties98
AbertayHackers:Abertay2020
Tennents94:Carlsberg94
Sensa2019:Sensa20!9
Holidays:Holidays1
Password1994:P@ssw0rd1994
Happy2020:2020Happy
Goldsmith:Silversmith
S0nsname:S0nsn@me
U11apool000:!nv3rn3ss
Supernova27:Supernov@17
Seabike58:Landbike58
Chester48:Chester48!
Dundee12:Ab3rd33n123
Richmond01:Richm@nd@1
Chestnut16:16Chestnut
0KiU7FMWbwwn5qXu:3ne4^#DxGHERh61D
Shakira1:Shskira1
Winter11:Winter_11!
Abigail07:Evie2018

-phrases
Boil,The,Jug2:B0ilTh3Jug1
pass,the,beers:passtheciders
delicate-,housetop-,peroxide-,guild:curtain-profane-cutesy-
balk
Football,Chicken,Youtube:TigerBananaChair
Global,Study,Deploy:LevelSocialQuote
6D0lly, The, 27Horse:12 Gid Gramp 10
Sandwich,Butterfly:MakeupLaptop23


-from
https://github.com/danielmiessler/SecLists/blob/master/Passwor
ds/Permutations/1337speak.txt
Honda8:Nissan1
4PPl3:Banana0
```

```
-from
https://github.com/danielmiessler/SecLists/blob/master/Passwor
ds/xato-net-10-million-passwords-1000.txt
b4tman:superman
Z0mb13:Und34d
Ferrari:Scuder1a
b1tter:b1tterly
nascar1234:1ndycar
futurama:s1mpsons
florida:m1ami

-from
https://github.com/danielmiessler/SecLists/blob/master/Passwor
ds/Common-Credentials/100k-most-used-passwords-NCSC.txt
qwerty123:qwerty12345
greenday:radiohead
vauxhall:corsa1
dragon1:monst3r
linkedin123:faceb00k
cherokee:jeep1234

-from researcher
password:password123
```

Appendix G – Test Results

Old Password	New Password	Custom Wordlist	RockYou Wordlist	Probable Wordlist
0KiU7FMWbwwn5qXu	0KiU7FMWbwwn5qXu	NF	NF	NF
#8qeqz8Qp@wT2b	ptC8U%kQLTr&!w	NF	NF	NF
Yinjbiovucy986	p8bbcicjnY5f	NF	NF	NF
12Bl1ndMice$	M&mtwentyfour4#	NF	NF	NF
234RampantLizards!	PiratesAreGone!76	NF	NF	NF
Youpe3ple45	Defer3ytl	NF	NF	NF
Ashtanga20	Change it	NF	NF	NF
Cowdengelly	Cowden632	NF	NF	NF
Ca10duenco&#;= pq8h3	Iwhdos7h22ody99&Ã—;=("Â£Ã·;'	NF	NF	NF
TulipsforSpring2028	ManagemyPassword3991	NF	NF	NF
Gartocharn1	Rhodes123	NF	NF	NF
Freelands18300	QueryTime1066	NF	NF	NF
Stewart191074	Gillian24011975	NF	NF	NF
=@tpItLfaw	=@tpLdapaap	NF	NF	NF
18Germany!!	19CroatiaÂ£@	NF	NF	NF
ThankYou8	Yes12335	NF	NF	NF
Winter19@!	Summer20##	NF	NF	NF
Filledin	Infilled	NF	NF	NF
4thyear2020!!	Smarties98	NF	NF	NF
AbertayHackers	Abertay2020	NF	NF	NF
Tennents94	Carlsberg94	NF	NF	NF
Sensa2019	Sensa20!9	165	NF	NF
Holidays	Holidays1	31	2134470	NF
Password1994	P@ssw0rd1994	315	NF	NF
Happy2020	2020Happy	12	NF	NF
Goldsmith	Silversmith	2285	NF	NF
S0nsname	S0nsn@me	141	NF	NF

U11apool000	**!nv3rn3ss**	NF	14338175	NF
Supernova27	**Supernov@17**	NF	NF	NF
Seabike58	**Landbike58**	NF	NF	NF
Chester48	**Chester48!**	123	NF	NF
Dundee12	**Ab3rd33n123**	NF	NF	NF
Richmond01	**Richm@nd@1**	NF	NF	NF
Chestnut16	**16Chestnut**	30	NF	NF
0KiU7FMWbwwn5qXu	**3ne4^#DxGHERh61D**	NF	NF	NF
Shakira1	**Shskira1**	NF	NF	NF
Winter11	**Winter_11!**	NF	NF	NF
Abigail07	**Evie2018**	NF	NF	NF
BoilTheJug2	**B0ilTh3Jug1**	683	NF	NF
passthebeers	**passtheciders**	4257	NF	NF
delicate-housetop-peroxide-guild	**curtain-profane-cutesy-balk**	NF	NF	NF
FootballChickenYoutube	**TigerBananaChair**	NF	NF	NF
GlobalStudyDeploy	**LevelSocialQuote**	NF	NF	NF
6D0lly The 27Horse	**12 Gid Gramp 10**	NF	NF	NF
SandwichButterfly	**MakeupLaptop23**	NF	NF	NF
Honda8	**Nissan1**	712	705836	NF
4PPl3	**Banana0**	NF	NF	NF
b4tman	**superman**	446	48	21
Z0mb13	**Und34d**	1457	NF	NF
Ferrari	**Scuder1a**	939	NF	NF
b1tter	**b1tterly**	3113	NF	NF
nascar1234	**1ndycar**	1126	NF	NF
futurama	**s1mpsons**	306	335911	163983
florida	**m1ami**	8329	NF	NF
qwerty123	**qwerty12345**	44	31970	3386
greenday	**radiohead**	466	6133	2848
vauxhall	**corsa1**	NF	52563	NF

dragon1	**monst3r**	824	298934	NF
linkedin123	**faceb00k**	837	662942	NF
cherokee	**jeep1234**	NF	305391	NF
password	**password123**	40	1383	595

Appendix H – Performance Metrics

RAM	**Run 1 (s)**	**Run 2 (s)**	**Run 3 (s)**	**Run 4 (s)**	**Run 5(s)**
2GB	99.13	102.76	105.18	102.34	99.56
4GB	33.55	15.03	14.68	14.37	16.26
6GB	33.31	10.33	10.42	10.87	10.76
8GB	35.39	10.44	8.9	9.08	8.97