Universiteit Leiden
The Netherlands

Opleiding Informatica

PassGAN and Hashcat

A look at how traditional password crackers can be used to complement PassGAN

Bas Treffers

Supervisors:
Eleftheria Makri
Nusa Zidaric

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

4/3/2025

Abstract

In 2018 a new method of password cracking was proposed. This method, called PassGAN, made use of Generative Adversarial Networks trained on a wordlist consisting of leaked passwords in order to generate a wordlist that could be used to crack password hashes. It was able to deliver promising results, matching the performance of existing password cracking methods when allowed to create a large enough wordlist. It also showed that PassGAN was able to generated passwords not present in the original dataset. Included in the study was a section examining the performance of an attack making use of both PassGAN and an existing rule-based attack. The results were promising, using the two methods in tandem resulted in a larger number of passwords being cracked as compared to when only doing a rule-based attack.

This thesis seeks to further examine the merits of using an attack which makes use of both traditional methods, such as masks and rule-based methods as executed by Hashcat, and PassGAN

Contents

1 Introduction

Passwords have been the most common form of user authentication for a long time and will likely stay the most common form of authentication for the foreseeable future [HVO12]. This ubiquity along with the fact that users tend to create passwords that are easy to remember (and thus easy to crack) [AS99], and reuse those passwords across several accounts[DBC+14, FH06] means that passwords are a point of vulnerability in any security infrastructure. Passwords are usually turned into a hash using a hashing algorithm in order to prevent passwords from being easily misused when stolen. However, these hashes can be cracked and tools have been developed specifically to crack password hashes. These tools, like Hashcat [Hasj] and John the Ripper[Joh], make use of several different techniques to crack passwords. All of these techniques generate potential passwords, which are then hashed using a hashing algorithm selected before the attack was started and the resulting hash is compared to the hashes that the tool is trying to crack. If the two hashes match then the password is considered cracked. Both Hashcat and John the Ripper have near identical capabilities when it comes to cracking passwords, but Hashcat was chosen to be used for this thesis as it is simpler to install and use. Hashcat also includes an option to perform attacks on a list of plaintext passwords. This means that rather than generating a plaintext password, hashing said password, and comparing the generated hash to a target hash, instead a plaintext password is generated and compared to a list of plaintext passwords. And a target password is considered "cracked" if the same plaintext password is generated by Hashcat. This option was used during this thesis in order to save time. Hashing passwords would take time during the attack and not doing so would not affect the results of any tests.

In recent years the field of neural networks has advanced and now offers a new avenue for password cracking. By training neural networks to generate passwords using old password leaks as input, it is possible to have it generate passwords that are indistinguishable from those generated by humans. The potential of using neural networks for password cracking has been explored in a 2018 paper. In this paper the authors create and test a password generating tool called PassGAN [HGAPC19]. PassGAN makes use of Generative Adversarial Networks (GANs) to generate passwords.

The paper that originally proposed PassGAN compared the performance of PassGAN to that of existing tools, including attacks executed using Hashcat. The paper also included a section about the effectiveness of using PassGAN in tandem with an attack executed by Hashcat. The results of this section indicate that there was increased password coverage when these tools are used together. This thesis aims to further explore the potential of using PassGAN in tandem with existing password cracking techniques, such as rule-based attacks, Markov-model based attacks, and mask attacks. The tool used to execute these attacks is Hashcat.

2 Background

2.1 Password Cryptography

Passwords were first introduced in the 1960's. They were initially used on time sharing systems in order to authenticate users, and to protect those users data[Hask]. However, at this time all passwords were stored in plaintext. This meant that if the list of passwords somehow leaked, the

entire list of accounts would be compromised. In order to combat this security flaw, a hashing algorithm was introduced, crypt[Hask]. Hashing algorithms take input from the user, their password, and generate a fixed length string based on that user input, meaning that every user generated string corresponds with a specific string generated by the hashing algorithm. This string generated by the hashing algorithm is referred to as a hash. This means that instead of storing the plaintext password, it is possible to store the string generated by the hashing algorithm instead. And when a user tries to log in and fills in a password, that password is hashed and the generated hash is compared to the one that is stored on the computer for the account the users is trying to log in to. If the strings match, the user can log in to that account. And hashing algorithms have another property, they are very hard to reverse. This is due to three reasons. First is preimage resistance; for any given hash it is computationally infeasable to find the input that produced it[sec]. Next is second preimage resistance. This property means that for any given input it is computationally infeasible to find another input which produces the same hash[sec]. Lastly there is collision resistance, which means it is impossible to find any pair of inputs that result in the same hash[sec].
Crypt was originally developed for Unix, but after its introduction, some flaws were quickly identified. Specifically, if two users were to coincidentally share a password, they will also share a hash. The reason for this is that hashing algorithms when given a specific output will always produce the same output. But this also means that, when two users share a password, cracking one hash could result in multiple accounts being compromised. At the time, computers were capable of hashing a password using crypt in 1.25 milliseconds[Hask], meaning that it did not take much time to check a large number of potential passwords by hashing them and comparing the stored hashes.
To alleviate the first flaw, salts were introduced in the late 1970's. These were short random strings, 12-bits long at the time, that were appended onto the end of a password. This meant that even if two users shared a password, they would not necessarily share the same hash because of the users having different salts. Another improvement that was made was that crypt started iterating a cypher called the Data Encryption Standard (DES) cypher create a hash from a user's password[Pro].
These two additions increased the amount of time necessary to execute any attack as every password tested now needed to be hashed with 2^12 different salts appended at the end and it became impractical to prepare an encrypted dictionary in advance due to the sheer number of possible hashes[Hask].

However, as technology advanced and computers became faster, this alone was no longer enough. It became possible to check too many passwords too quickly. So, in 1997, a new hashing algorithm was introduced, bcrypt[Hask]. Bcrypt had a key innovation, an adjustable cost factor[Pro]. This factor introduced adaptive hashing, meaning that the function could be set to iterate a certain number of times to increase the time required to compute the hash. This causes any attack to become far more resource intensive. Aside from introducing the cost factor, bcrypt also used a 128-bit salt value, further increasing the number of possible salts that could be added to the end of a password.
The 2000s saw the introduction of Password-Based Key Derivation Function 2 (PBKDF2), which also featured an adjustable cost factor, but added two other parameters. One was the ability to set the length of the derived key, and the other the introduction of key stretching, a method whereby a cryptographic function is run repeatedly on a password and hash. This was done to increase the time necessary to execute brute-force attacks as in order to get the correct hash for any given password and salt it is necessary to also perform these time consuming operations. This made passwords even harder to crack[Hask]. But a new avenue would soon start to open for password

crackers.
The late 2000's would see the first use of GPUs to crack passwords. By making use of GPUs ability process a large amount of data in parallel, it became possible for GPUs to outperform CPUs when it came to password cracking[Hask].

Over time hashing algorithms have kept changing in order to keep up with the ever increasing computing power that computer systems get access to. According to the National Institute of Standards and Technology (NIST), a US government agency that regularly puts out security standards as guidance for organizations to follow, proper security can be provided by the SHA-2 family of hashing algorithms[NIS]. Although it should be noted that this family of hashing algorithms is in the process of being replaced by the SHA-3 family of hashing algorithms. The SHA-2 and SHA-3 families of hashing algorithms posses the 3 key properties discussed earlier, preimage resistance, second preimage resistance, and collision resistance. These properties mean that the these families of hashing algorithms are secure.

2.2 GANs

Neural networks are a type of program that attempts to mimic how a human brain learns. It does this through nodes, which consist of an input function and an activation function, and is capable of taking inputs and producing output. Neural networks consist of three layers of these nodes: the input layer, hidden layer, and output layer[RN09]. Notable here is the hidden layer because, unlike the input and output layers, the hidden layer can consist of multiple layers of nodes. The input nodes take a given input and send the produced output to the nodes in the hidden layer. The nodes in the hidden layer take the output from the previous layer, either input or hidden, and send the produced output to either the next layer of hidden nodes, or the output layer of nodes. The output node takes the output from the hidden nodes as input and produces an output based on that. Each input received by a node has a weight associated with it. This weight is what gets adjusted during the training of a neural network.
To train a neural network a dataset is split in two, the training set and the test set. The training set is used to train the network, it is used to define what the desired output looks like. The test set is used to verify the performance of the network after training. This is done to prevent the network from only recreating the training set[RN09].
During training the output of the output nodes is compared to the data in the training set. And after determining how well a given ouput matches the training set the weights of the various layers are adjusted based on how close the output was to the data in the training set[RN09].

Generative Adverserial Networks (GANs) consist of two neural networks. Neural network G, a generative model that captures the data distribution, and Neural network D, a discriminative model that estimates the probability that a sample came from the training data rather than G[GPAM+14]. By doing this the two neural networks help train each other, one gets better and better at generating output similar to the training set, the other improves at distinguishing between the training set and the generated output of the other network.
PassGAN makes use of Improved training of Wasserstein GANs[HGAPC19]. This method of training GANs iterates on a method of training neural networks that makes use of Wasserstein distance to define how good the output of a network is. This has the effect of improving the stability of learning,

gets rid of problems such as mode collapse, and provides a meaningful learning curve that is useful for debugging and hyperparameter searches[ACB17]. The Improved training of Wasserstein GANs attempts to solve a problem that occurs in the original version of Wasserstein GANs (WGANs) that causes models to fail to converge. It does this by penalizing the norm of the gradient of the discriminator with respect to its input[GAA+17a].

3 Related Work

Formal research into passwords and how to create hard to crack passwords dates back at least to 1967[Dhi67] and continues until this very day. And passwords are more pervasive today than ever before with users having 25 accounts with passwords on average[FH06].
These passwords are usually stored in plaintext, but rather a hash is generated using a hashing algorithm and this hash is stored. It is also common to append a salt to the end of a password before hashing it to make it more time consuming to crack the password and to make each password unique. A password guessing attack attempts to crack this hash by generating passwords, hashing them, and comparing the generated hash to the password hash they want to crack. If the hashes match the passwords is considered cracked.

One popular password guessing tool is Hashcat[Hasj]. This tool is capable of executing various types of password guessing attacks and was the tool used to execute any traditional password attack in this thesis.

The most basic type of attack is a **brute-force attack**, also called an exhaustive search attack. A brute-force attack generates all possible passwords within a given password space and compares these generated passwords to the list of passwords which the tool is attempting to crack. It should be noted here that in this thesis the term keyspace will be used as a term synonymous to password space, the space of all possible passwords that can be generated given a specific alphabet, and search space, the set of all possible solutions to a problem. The time it takes to crack a password using this method increases based on the length of the password. A variation of brute-force attack was developed which executes a more precise version of brute-force attack. This type of attack is referred to as a **mask attack**[Hasf]. A mask attack is a more precise form of a brute-force attack. A mask attack reduces the keyspace to a more efficient one by defining what set of characters can appear on a specific position[Hasf]. This type of attack has no disadvantage when compared to a pure brute-force attack, it merely allows for the keyspace of password candidates to be reduced to a more efficient set by creating masks that mirrors the ways in which humans create passwords[Hasf]. Hashcat defines 8 different sets of characters which are shown in Table 1. These symbols are combined to create a mask, which defines the keyspace over which a mask attack is performed. An example of a mask would be ?u?l?l?l?l?d?d?d. Passwords generated from this mask would consist of a capital letter, 4 lowercase letters, and 3 digits in that order. All passwords generated using this mask will have a length of 8.

Symbol	Set
?l	abcdefghijklmnopqrstuvwxyz
?u	ABCDEFGHIJKLMNOPQRSTUVWXYZ
?d	0123456789
?h	0123456789abcdef
?H	0123456789ABCDEF
?s	<space>!"#$%&'()*+,-./:;<=>?@{\}
?a	?l?u?d?s
?b	0x00 - 0xff

Table 1: Character sets used to create a mask. Most symbols code for a specific set of characters. The exceptions are ?a which defines a combined character set which includes all upper case letters, lower case letters, digits, and special characters, and ?b which defines characters based on their hexadecimal code.

Hashcat also allows for the creation of 4 custom character sets. An example of a custom character set being defined would be "–custom-charset1=?l123". This codes for set consisting of all lower case letters and the digits 1,2, and 3.
Hashcat also has a setting to iterate a mask which is called using -i when running Hashcat. When using this option the mask ?u?l?l?l?l?d?d?d would first be run as the mask ?u, then ?u?l, then ?u?l?l, and so forth until eventually running ?u?l?l?l?l?d?d?d. This allows for multiple mask attacks to be executed in one run without needing to initiate multiple mask attacks.

A **Markov-model based attack**[NS05] seeks to further optimize passwords guessing by making use of the fact that it is unlikely for passwords generated by people are distributed uniformly in the space of possible characters. So in order to model the distribution of symbols , a Markov-models can be used. Markov-models originate from natural language processing and are used to define the probability distribution over a sequence of symbols [NS05]. By modeling the distributions of symbols of a list of passwords generated by humans, a Markov-model can be created that can be used to generate passwords based on the distribution of symbols in the original password list.
Hashcat itself does not have the capability to execute a Markov-model based attack, however it does offer a tool for generating passwords using a Markov-model on its website. This tool is called statsprocessor[Hase] and by piping the output form this tool into Hashcat it is possible to perform a Markov-model based attack. Any time a password generated by the Markov-model matches a password in the target wordlist, the password in the wordlist is considered cracked.

A **dictionary attack**[Hasc] is a type of attack which seeks to exploit the fact that people have an easier time remembering words as compared to remembering random sequences of characters and are thus likely to use full words in their passwords[BSB18]. So by using a large dictionary of words and comparing those against the target passwords list, it is possible to crack passwords by attempting to match the target passwords with entries in the dictionary. If the two passwords match, the target passwords is cracked.
This type of straight dictionary attack is less effective today due to the introduction of password policies. The exact rules of a password policy vary but an example would be to require passwords to have a minimum length of 10 characters and to include a number, a capital letter, and a

special character. How humans choose to modify their passwords based on these restrictions makes passwords more varied and harder to guess[SKD+16]. This policy increases the time necessary to execute a brute-force attack as it makes the keyspace that a brute-force attack would need to cover larger due to there being more spaces where a number or special character could potentially appear, and reduces the number of passwords cracked by a dictionary attack as users will modify words in a meriad of ways, which decreases the chances that those passwords are present in the dictionary used in the attack.

However, a variation of dictionary attacks has been developed which allow for effective guesses to be made without needing to change existing dictionaries. This was done by introducing so called **mangling rules**[Hash]. These mangling rules are used to modify the words in the dictionary. This is effective for the same reason that dictionary attacks are effective, people are predictable. And when made to abide by password policies they tend to take a dictionary word and modify it in some predictable way[BSB18]. By knowing how people tend to modify their passwords rules can be created which modify words in the dictionary the same way that people tend to modify their passwords.

For example a person using the password "password" might change their password to "Passw0rd!" when made to abide by a password policy which forces them to include a number, a capital letter, and a special character. A rule can be created which modifies a word in the dictionary in a way which mimics how people tend to modify words. To match the example a rule could be created which would capitalize the first letter, change all 'o' to '0', and append a '!'. This would allow for the transformation of "password" into "Passw0rd!". However, one rule would not cover all the ways in which people modify their passwords. To solve this mangling rulesets were created, which are files which includes multiple mangling rules. For example one rule could capitalize the first letter and append a '1' at the end of words, and another rule could capitalize the first letter and append a '2' at the end of words. These rules are written using something resembling a programming language. This programming language is partially the same even when using different password crackers. This study makes use of Hashcat, but John the Ripper, another password cracking tool, is also capable of executing an attack using these rules.

Table 2 contains the Hashcat version of functions that are shared between Hashcat and John the Ripper, though there may be differences in how arguments are defined for the rules[Hash]. And Table 3 shows rules unique to Hashcat. Lastly Table 4 shows the list of mangling rules that will be used in this thesis.

Name	Symbol	Description
Nothing	:	Do nothing.
Lowercase	l	Lowercase all letters.
Uppercase	u	Uppercase all letters.
Capitalize	c	Capitalize the first letter and lower the rest.
Invert Capitalize	C	Lowercase first found character, uppercase the rest.
Toggle Case	t	Toggle the case of all characters in a word. So any uppercase character becomes lower case and vice versa.
Toggle at	TN	Toggle the case of characters at position N.
Reverse	r	Reverse the entire word.
Duplicate	d	Duplicate entire word.
Duplicate N	pN	Append duplicate of word N times.
Reflect	f	Append reversed duplicate of word.
Rotate Left	{	Rotate word left. So the first letter becomes the last with 1 rotation.
Rotate Right	}	Rotate word right. So the last letter becomes the first with 1 rotation.
Append Character	$X	Append character X to the end of word.
Prepend Character	^X	Prepend character X to start of word.
Truncate left	[	Delete first character.
Truncate right	]	Delete last character.
Delete at position	DN	Delete character at position N.
Extract range	xNM	Extract M characters, starting at position N. Those characters then form the new word.
Omit range	ONM	Delete M characters, starting at position N. The remaining characters then form the new word.
Insert a position	iNX	Insert character X at position N.
Overwrite at position	oNX	Overwrite character at position N with X.
Truncate at position N	'N	Truncate word at position N.
Replace character(s)	sXY	Replace all instances of X with Y
Purge character(s)	@X	Purge all instances of X.
Duplicate first N	zN	Duplicate first character N times.
Duplicate last N	ZN	Duplicate last character N times.
Duplicate all	q	Duplicate every character.
Memorize	M	Memorize current word.
Extract Memory	XNMI	Insert substring of length M starting from position N of word saved to memory at position I.
Append Memory	4	Append the word saved to memory to the current word.
Prepend Memory	6	Prepend the word saved to memory to the current word.

Table 2: Mangling rules shared between Hashcat and John the Ripper. Each of these individual rules modifies words in a wordlist to generate new words when executing a rule-based attack. These rules can also be combined to perform multiple modifications on a single word. This is done by defining multiple rules in sequence on a single line. An example would be uexampl$e which would first set all characters of a word to uppercase and then append the string "example" to the end of the word. A file which contains multiple lines of rules is called a mangling ruleset.

Name	Symbol	Description
Swap Front	k	Swap first two characters
Swap Back	K	Swap last two chracters
Swap At N	*NM	Swap character at position N with character at position M
Bitwise shift left	LN	Bitwise shift left character at N
Bitwise shift rigth	RN	Bitwise shift right character at N
ASCII increment	+N	Increment character at N by 1 ASCII value
ASCII decrement	-N	Decrement character at N by 1 ASCII value
Replace N + 1	.N	Replace character at N with value at N+1
Replace N - 1	,N	Replace character at N with value at N-1
Duplicate block front	yN	Duplicate first N characters
Duplicate block back	YN	Duplicate last N characters
Title	E	Lower case the whole line, then upper case the first letter and every letter after a space
Title with separator	eX	Lower case the whole line, then upper case the first letter and every letter after a custom separator character
Toggle with Nth separator	3NX	Toggle case the letter after the Nth instance of a separator character X

Table 3: Mangling rules unique to Hashcat.

Name	Example
best64	$0
D3ad0ne	*03
dive	c
OneRuleToRuleThemStill	+6
T0XlC	u$z

Table 4: Mangling rules used in this thesis. The examples make the following modifications; best64 appends a 0, D3ad0ne swaps the first and 4th characters, dive capitalizes the first letter, OneRuleToRuleThemStill increments the character at postition 6 by 1 ascii value, and T0XlC changes all letters to uppercase and appends a z.

More recently methods which make use of Artificial Intelligence to crack passwords have been proposed. The one relevant to this thesis makes use of Generative Adversarial Networks (GANs) in an attempt to generate passwords which hard to distinguish from human passwords[GAA+17b, HGAPC19]. GANs consist of two neural networks that are working against each other[GPAM+14]. In the case of PassGAN one network is attempting to generate passwords while the other is trained to guess if a password was generated by the other neural network or whether it belonged to the training set. This training set consists of a large list of human generated passwords. By pitting these two networks against each other both are pushed to improve at their respective jobs. Competition between the neural network results in the ability for one of the neural networks to generate passwords that are indistinguishable from human passwords and this network can then be tasked with generating passwords for use in password cracking.
The research which proposed PassGAN it was shown that it can match the effectiveness of traditional password guessing tools given a large enough number of guesses.

4 Methodology

4.1 Setup

For this thesis PassGAN was trained using the RockYou dataset[Roc]. This passwords dataset originates from a leak from the company of the same name which used to develop widgets for the social media site MySpace. Something notable about this leak, and part of why the number of passwords present in it is so large, is that all of the passwords were stored as plaintext. This means none of them needed to be cracked and the full list of plaintext passwords users used is known. The leak occured in 2009 and included 32.503.388 passwords of which 14.344.391 are unique. Of this unique set of passwords 9.926.276 (69,2%) are length 10 or less. PassGAN was trained on unique passwords of length 10 or less originating from the RockYou dataset.
This dataset was split 70/30 with 10.042.020 being used to train the PassGAN model and the remaining 4.302.371 being the test set.
Entries of more than 10 characters were not removed from the dataset as the PassGAN program itself allows for limiting of the length of entries used for training. This decision was made as the workstation used to train the program did not have enough memory to train the model using the full dataset.
This list of unique passwords of length 10 or less used to train PassGAN was also used as the

wordlist for Hashcat in experiment 1.

The second dataset used in this experiment consists of 60.525.521 passwords. This dataset was obtained from the PassGAN github[Lin]. These passwords are referred to in the original paper as the LinkedIn dataset. Of this set 43.354.871 are both unique and of length 10 or less. A dataset was created which includes all passwords of length 10 or less and this dataset will henceforth be referred to as the LinkedIn dataset. According to the original paper this list consists of hashes which were cracked using John the Ripper[Joh] and Hashcat, this could cause disproportionate performance of attacks which can be executed using these tools[HGAPC19].
A clear overview of the number of passwords, number of unique passwords, and the number of unique passwords of length 10 or less of both datasets can be seen in Table 5. Relevant to the experiments will be the number of unique passwords of length 10 or less.

	Rockyou	**LinkedIn**
Total passwords	32503388	60525521
Unique passwords	14344391	60516688
Unique passwords length 10 or less	**9926276**	**43354871**
Percentage of unique passwords length 10 or less	**69,20%**	**71,60%**

Table 5: Overview of how passwords in RockYou and LinkedIn dataset. This table shows how many passwords there were in total on the first row, how many unique passwords each dataset contained in the second row, how many unique passwords had a length of 10 or less, and the precentage of unique passwords that were length 10 or less.

The exact details of the password leak from which these passwords originate is not stated, nor was it stated how this dataset was obtained for the original study as the link in the original study no longer functions. It is possible that this set of passwords originates from a 2012 leak[Nga12] but little information is given about this dataset in the original study.

This thesis will not make use of the original PassGAN implementation, which was implemented in Python 2.7 [Pasa], and will instead use a version of PassGAN which has been implemented in Python 3.8.19 [Pasb] as there was difficulty in getting the version implemented on Python 2.7 running. Python version 3.8.19 was used to train PassGAN and the version of the Tensorflow library used for the training was 2.5.1. All experiments were run on a workstation running Ubuntu version 22.04.4 with 16GB of RAM, a i7-8700 3.20GHz Intel CPU, and an NVIDIA GeForce GTX 2070 with 8GB of memory.

4.2 PassGAN training

The PassGAN model was trained using the RockYou dataset described above. This dataset was split 30/70 with 4302371 (29,99%) being used as the testing set and the remaining 10042040 (70,01%) being used for training. This split was chosen to balance training the model properly with not having the model use too much memory. The original paper used a 20/80 split with 80% being used to train PassGAN, however this was not possible as the workstation used to train the model trained for this thesis does not have enough memory to train the PassGAN model using 80% of the

RockYou dataset. When attempted the workstation would freeze up and require a forced shutdown. The following parameters were used to train the model. They are set in the train.py file. The specific values of the parameters originate from the original paper[HGAPC19] or were the standard value in the train.py file. These values were chosen as they were found to be effective values in the original PassGAN paper. The descriptions of the parameters originate from the original paper[HGAPC19].

- Iterations: 200000. This number indicates how many times the GAN invokes its forward set and its back-propagation step.
- Batch-size: 64. This represents the number of passwords from the training set that propagate through the GAN at each step of the optimizer.
- Maximum sequence length: 10. Maximum password length.
- Hidden layer dimensionality: 128. This represents the number of dimensions for each convolution layer.
- Number of discriminator iterations per generator update: 10. This indicates how many iterations the discriminator performs in each GAN iteration.
- Gradient penalty coefficient (noted as lamda in code): 10. This sets the penalty applied to the norm of the gradient of the discriminator with respect to its input. Increasing this parameter leads to a more stable training of the GAN[HNLP18].

The training of the model took around 12 hours.

After the training of the model was completed 2.074.625.024 passwords were generated (of which 1.125.512.894(54,25%) were unique) over a period of 12 hours. This number is close to the number of passwords required to match the performance of the gen2 mangling ruleset ($2,1 * 10^9$) tested in the original PassGAN paper, which was able to crack 15,54% of the LinkedIn database according to the original PassGAN paper. The list of unique passwords was used for all experiments performed.

4.3 Experiments

For this thesis, 5 experiments were performed. The first experiment focuses on comparing the Python 3.8 version of PassGAN used in this thesis to the 2.7 version of PassGAN used in the original paper. With the exception of the first, all experiments have the goal of evaluating whether there is any benefit to using PassGAN alongside a more traditional password cracking method.
The number of uncracked passwords was kept track of for each individual step for each experiment that had multiple steps.

4.3.1 Experiment 1

The first experiment focused on recreating an experiment done in the paper which proposed PassGAN. In the original experiment Hashcat using the best64 mangling ruleset was used in order to generate passwords from their training set, which consisted of 80% of the RockYou dataset. This experiment instead uses 70% of the RockYou dataset as a dictionary for Hashcat using the best64 mangling ruleset. This is because 70% of the RockYou dataset was used as the training set to train

the Python 3.8 version of PassGAN due to the workstation used in training not having enough memory to use 80% of the RockYou dataset for training.

First the 70% of the Rockyou dataset, the dataset that was also used to train PassGAN, was used as a dictionary for a rule-based attack executed by Hashcat made use of the best64 ruleset. This attack will produce a list of cracked and uncracked passwords. The cracked passwords are put aside; meanwhile, another attack will get executed on the list of uncracked passwords. This second attack is a dictionary attack that uses the passwords generated by PassGAN as a dictionary for the attack.

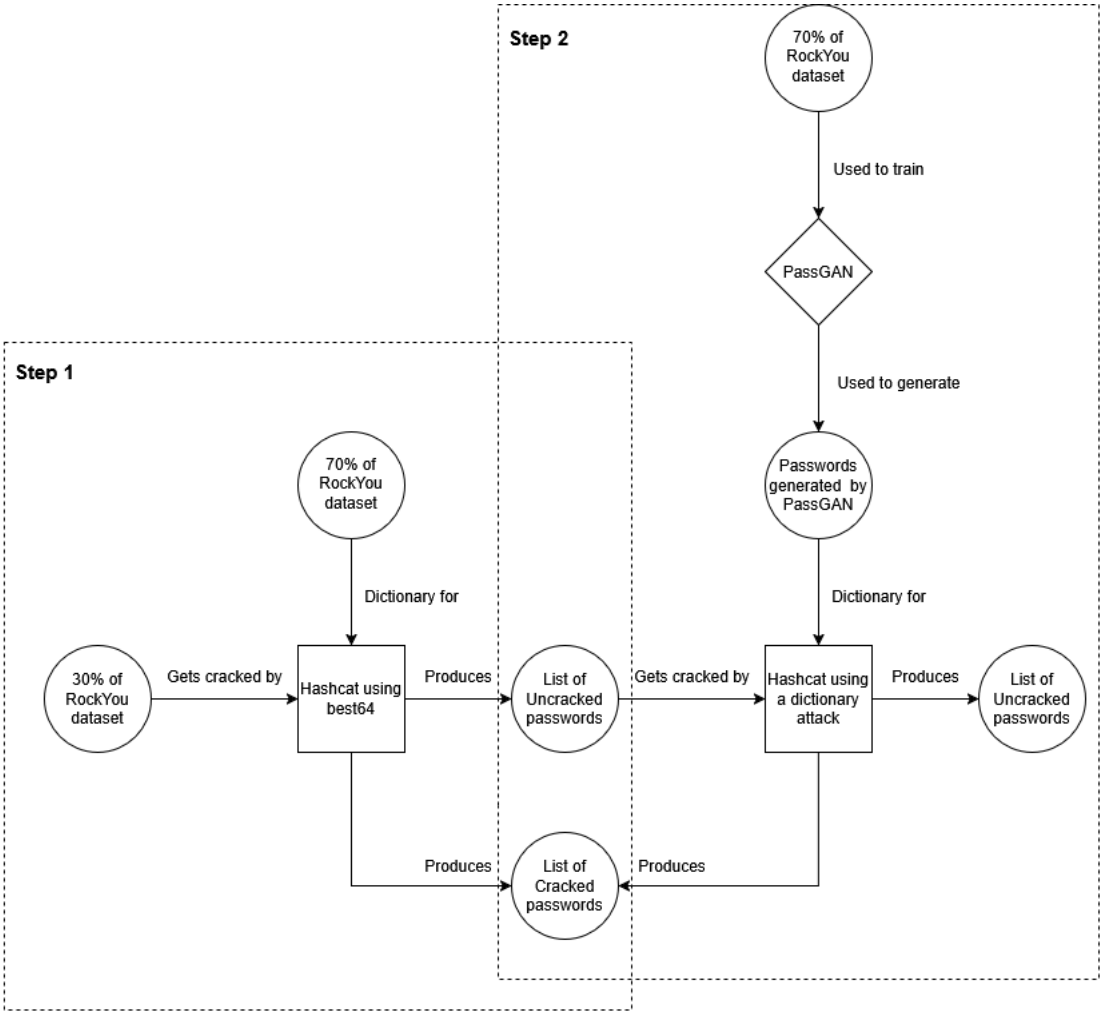

Figure 1: Graph showing an overview of how the tools and datasets used in experiment 1 fit together.

4.3.2 Experiment 2

The goal of this experiment was to evaluate the effectiveness of a dictionary attack using PassGAN when used alongside a mask attack.

First hashcat was used to execute a dictionary attack using the passwords generated by PassGAN. This attack produced a list of cracked and uncracked passwords. The cracked passwords were set aside; meanwhile, more attacks were executed on the list of uncracked passwords. Each of these attacks was a mask attack using a different mask, and each attack was executed in parallel. The results of each attack, the lists of cracked and uncracked passwords, were stored separately for each attack.
For this experiment several masks were used to attempt to crack the LinkedIn dataset. A total of 5 masks were tested and they are show in Table 6.

For this experiment two masks were run with Hashcats -i option for mask attacks which iterates the length of the mask, starting with only the first character of the mask and adding the next character when all options of the current set are exhausted, while 3 were run without this option. The results of the masks that made use of the -i option will be discussed separately from the masks, that did not make use of the -i option. The attacks which made use of the -i option were stopped if an iteration of a mask took longer than 6 hours as the number of passwords generated by the next iteration of the mask is equal to the number generated by the current mask multiplied by the number of characters in the character set that is added on in the next iteration. The smallest character set that can be added is ?d with a total of 10 characters. This means that the number of passwords generated increases tenfold at a minimum as compared to the previous set and the time necessary to check the next set likely increases at an approximate rate, though not exactly the same rate due to the number of passwords decreasing over time. If a mask is stopped early during the experiment the mask used will be changed in the results table to the last iteration of the mask that was run.
The two masks that were run with the -i option were the masks ?1?l?l?l?l?l?l?d?d?d and ?1?2?2?2?2?2?2.
These masks were chosen to be ran this way because they generate passwords which include dictionary words.
The full mask ?1?l?l?l?l?l?l?d?d?d generates strings that start with either a lower or upper case letter and ends in a string of 3 digits. This mimics a simple way that users might modify passwords when forced to include a capital letter and 3 digits[SKD+16]. With the -i option it will also generate strings that start with a capital letter, have up to 6 lowercase letters after the first character, and up to 3 digits after a string of 7 characters. This mask does not generate shorter strings that end in up to 3 digits and it does not include special characters. For this reason three additional mask attacks were run with the masks ?1?l?l?l?l?2?2?2, ?1?l?l?l?2?2?2, ?1?l?l?2?2?2 to cover a part of the password space not covered by this first mask and give some idea of how many passwords exist that were not covered by the first mask.
The second mask that was ran using the -i option to generate passwords was the mask ?1?2?2?2?2?2?2?2?2?2, which includes any string of length 10 or less that starts with either a lowercase or upper case letter and then includes either a lower case letter or number at every other space in the string. This will mask will generate passwords that include random numbers or substitute letters for numbers.

Mask used	-i?	?1	?2
?1?l?l?l?l?l?l?d?d?d	✓	?l?u	
?1?2?2?2?2?2?2?2?2?2	✓	?l?u	?l?d
?1?l?l?l?l?2?2?2	X	?l?u	?s?d
?1?l?l?l?2?2?2	X	?l?u	?s?d
?1?l?l?2?2?2	X	?l?u	?s?d

Table 6: Masks used in Experiment 2

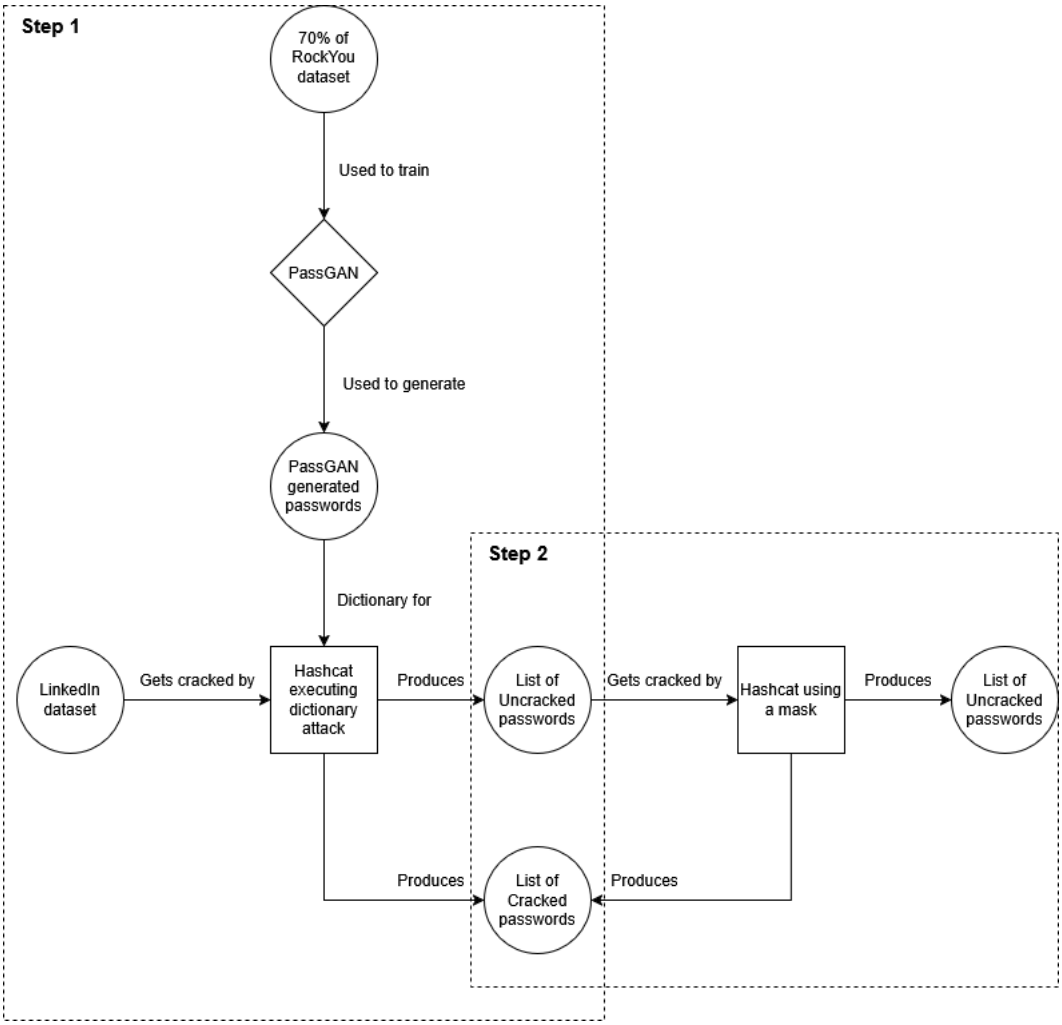

Figure 2: Graph showing an overview of how the tools and datasets used in experiment 2 fit together.

4.3.3 Experiment 3

The goal of this experiment was to evaluate the effectiveness of a dictionary attack using PassGAN when used alongside an attack which makes use of a Markov-model.
For this experiment a Markov-model was trained using the statsprocessor tool[Hase] using the

RockYou dataset as the training data. This model was then used to generate a list of passwords which were then used in a dictionary attack using Hashcat in one of the steps of this experiment.

For the first step of this experiment, hashcat was used to execute a dictionary attack using the passwords generated by PassGAN. This attack produced a list of cracked and uncracked passwords. The cracked passwords were set aside; meanwhile, an attack which made use of a Markov-model was executed using hashcat on the list of uncracked passwords.

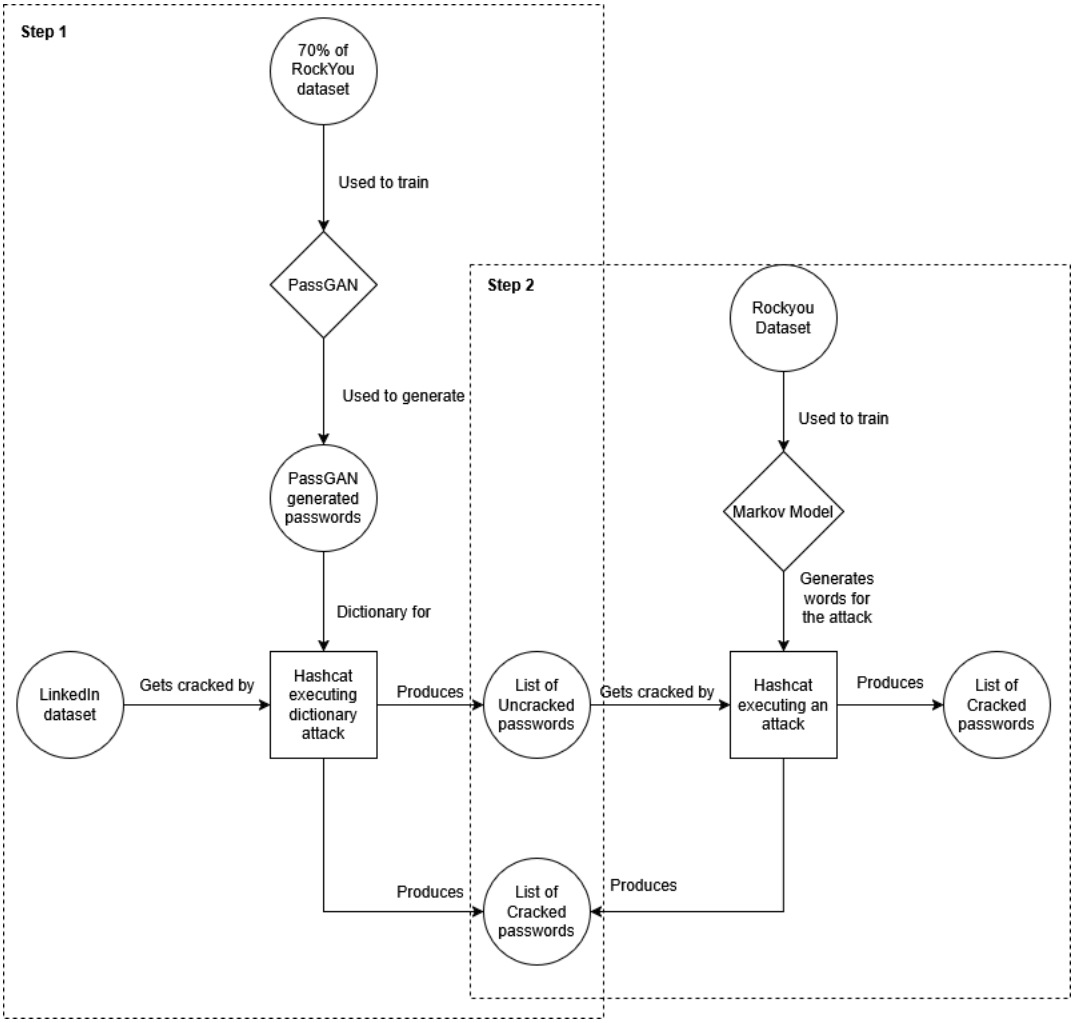

Figure 3: Graph showing an overview of how the tools and datasets used in experiment 3 fit together.

4.3.4 Experiment 4

The goal of this experiment was to compare several mangling rulesets and their performance when used alongside PassGAN.
The rulesets used were best64[Hasa], D3ad0ne[Hasb], dive[Hasd], OneRuleToRuleThemStill[Hasg], and T0XlC[Hasi]. These rulesets were chosen as they come standard with Hashcat and John the Ripper which are state of the art password cracking tools and are thus likely to include the most

effective available mangling rulesets available.

First hashcat was used to execute a dictionary attack using the passwords generated by PassGAN. This attack produced a list of cracked and uncracked passwords. The cracked passwords were set aside; meanwhile, more attacks were executed on the list of uncracked passwords. Each of these attacks was a rule-based attack using a different mangling ruleset, and each attack was executed in parallel.

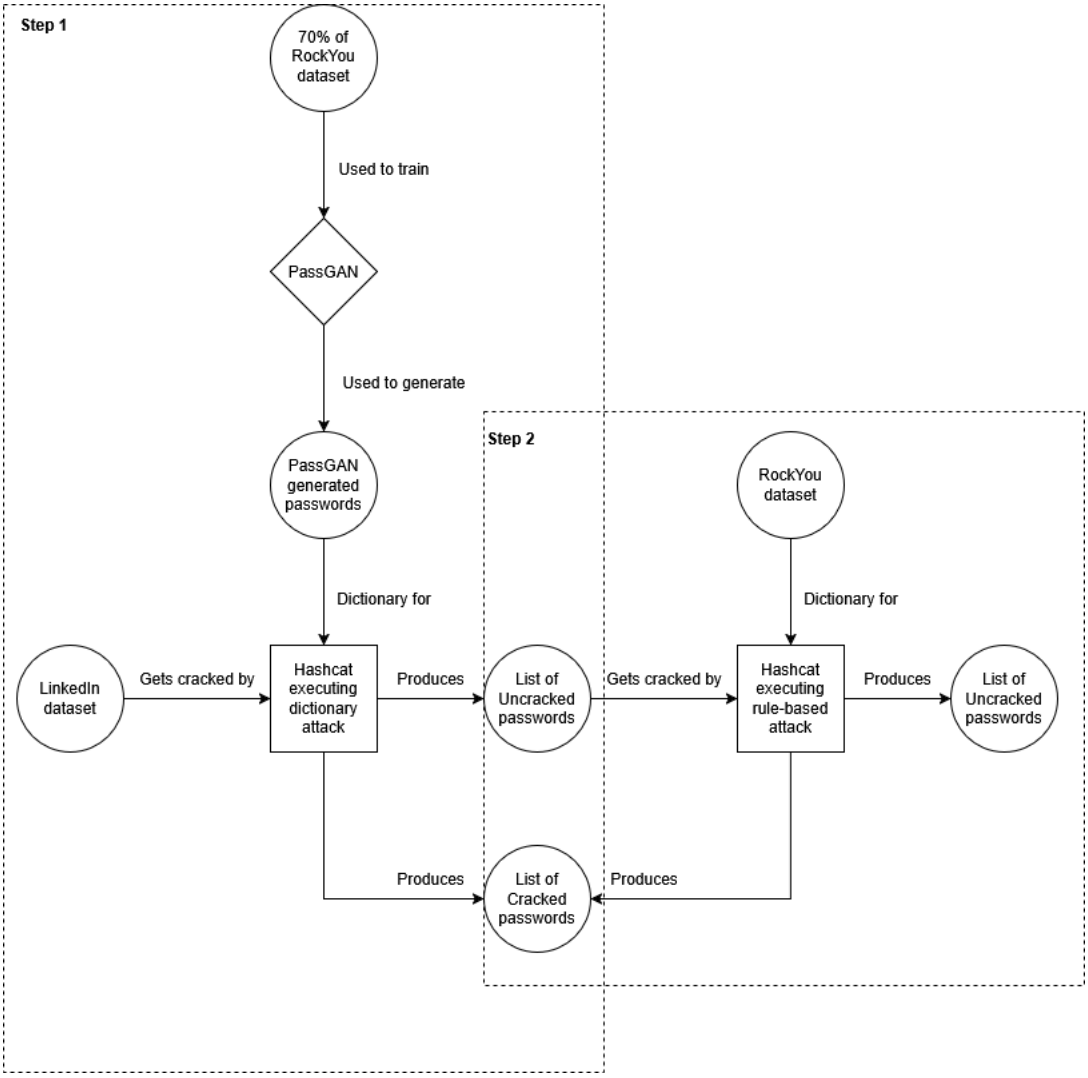

Figure 4: Graph showing an overview of how the tools and datasets used in experiment 4 fit together.

4.3.5 Experiment 5

The final experiment aims to explore how it is viable for a wordlist generated by PassGAN to be used for a rule-based attack.
This experiment involves the execution of two parallel rule-based attacks executed by hashcat,

one making use of a wordlist of 200.000.000 generated by PassGAN, and another which uses the Rockyou dataset as the wordlist. The lists of cracked and uncracked passwords were stored seperatly for each experiment.

This experiment is important as while using passwords from known password leaks is effective, it is ethically questionable to share these passwords lists, especially if the leaks are more recent, as the passwords in this list can include names, dates of births, and other potentially sensitive information. While this thesis did not make use of more recent password leaks, however leaks as recent as 2023 were trivial to locate. Were PassGAN able to generate password lists with similar performance to existing password leaks then it could discourage the sharing of these password lists as there would be a viable alternative to use in research.
The mangling ruleset used for this experiment was dive.

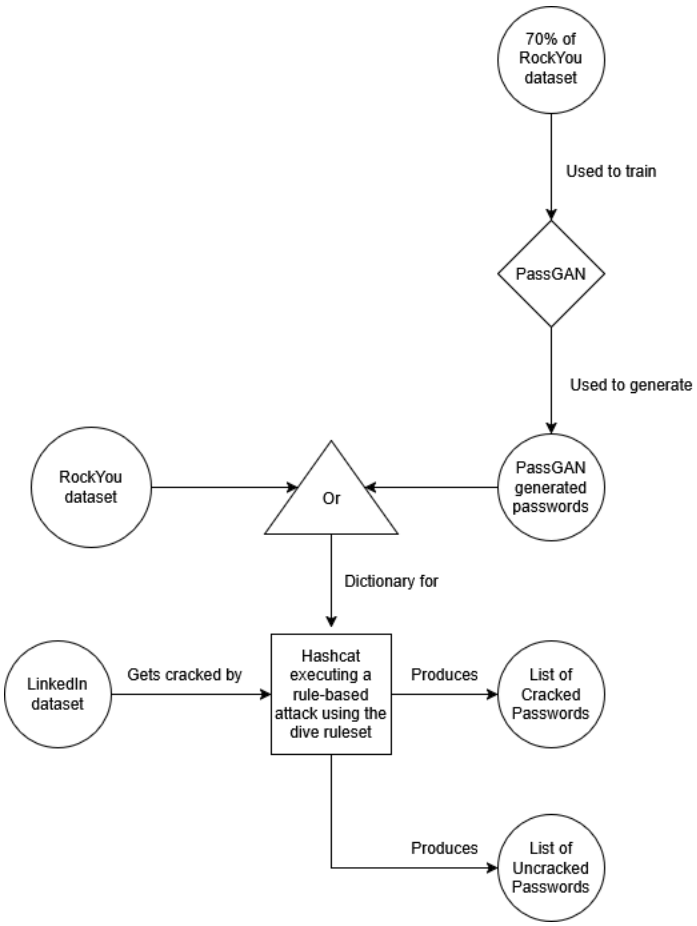

Figure 5: Graph showing an overview of how the tools and datasets used in experiment 5 fit together.

5 Results

5.1 Experiment 1

The goal of this first experiment is to gauge the performance of both the Python 3.8 version of PassGAN when used in conjunction with the current performance of Hashcat cracking with the best64 mangling rule. The dictionary for Hashcat was the training set that was used by PassGAN. The reason why the full RockYou dataset was not used in this test was in order to compare the results to the results of a similar experiment done in the original PassGAN study[HGAPC19]. The full RockYou dataset was not used in a similar experiment in the original paper hence why the full dataset was not used for this first experiment. It should also be noted that the number of unique passwords is higher in the experiment performed for this thesis as compared to the experiment performed in the original paper.

First Hashcat was allowed to attempt to crack the test dataset using best64 and using the training datset as its input. It was able to crack 1362155(31,66%) of the 4302371 passwords in the test set. After this, the remaining list was compared to the list of passwords generated by PassGAN and any matching password was removed from the list. After this test 1680682 (39,06%) of the passwords were cracked.

This is a decrease in number of passwords cracked, as compared to the original study. Hashcat performed similarly (31,85% cracked compared to this thesis' 31,66%). However PassGAN cracked a lower number of passwords as compared to the original study. In the original the combination of both attacks was able to crack 48,04% of the target dataset whereas here it only managed to crack 39,06% of the target dataset.

	Original	**This Thesis**
Total unique passwords	1978367	4302371
Hashcat cracked	630068	1362155
PassGAN cracked	320365	318527
Total cracked	950433	1680682
Total cracked %	**48,04%**	**39,06%**

Table 7: Comparison of results from an experiment performed in the orignal PassGAN paper and the one performed in this thesis.

As can be seen in Table 7, the Python 3.8 version of PassGAN performed worse than the Python 2.7 version of PassGAN used for the original study. The Python 2.7 version was able to crack 51% of the password that remained after Hashcat executed a rule-based attack, while the Python 3.8 version of PassGAN was only able to crack 10.83% of the remaining passwords.
This decrease in the number of passwords cracked can have three different explanations; the different training data split, or the difference in the number of passwords generated, the difference in version.

The original study split the RockYou dataset 80/20 meanwhile this thesis used a 70/30 split to train PassGAN. The smaller number of passwords in the training set and the larger number of

passwords in the test set could have affected the number of passwords cracked.
There were also fewer passwords generated by PassGAN for this thesis as compared to the number of passwords generated in the original study. The orignal study used $7 * 10^9$ passwords generated using PassGAN whereas this thesis only generated $2 * 10^9$ passwords using PassGAN.
Lastly the version of PassGAN used for this thesis was implemented in Python 3.8 wheras the original paper used Python 2.7 for their implementation. This version difference could have led to different results.

5.2 Experiment 2

This experiment focuses on evaluating the use of masks alongside PassGAN. The database used for this experiment was the LinkedIn dataset. PassGAN was first used to attempt to crack the LinkedIn dataset then Hashcat was used to execute a mask attack. PassGAN was able to crack 6547141 (15,09%) of the LinkedIn dataset. Several different masks were used for this experiment. The results per mask can be seen in Table 8. Noted in Table 8 are what mask was used, how many passwords the mask cracked, the total number of passwords cracked, and the percentage of the database cracked. Masks 1 and 2 used the -i function of Hashcat meaning that the length of the mask was iterated from 1 to the full length of the mask. Custom character sets were defined for these masks. The custom character set used are shown in the ?1 and ?2 columns. These custom character sets are defined using the command "–custom-charset1=?a123" with the "1" being either 1, 2, or 3 depending on which custom character set is being defined, and "?a123" defines the characters which belong to the set. Under column ?1 in Table 8 can be seen what characters are a part of custom character set 1, and under ?2 can be seen what characters are a part of custom character set 2. Custom character set 3 was not used in this thesis. If a mask took more than 6 hours to complete it was chosen as the final iteration of that mask, as the number of guesses required for any mask that comes after would be equal to the number of guesses required to complete the previous iteration times the number of options in the added character set. This means that the number of guesses it would take to test a mask increases by at least 10 times which increases the time required to finish the next iteration of the mask attack by a similar amount of time.
Each mask was ran separately, not sequentially. The only mask that was stopped early was the second mask which made use of the -i option.

Mask used	-i?	?1	?2	Mask cracked	Total cracked	Total % cracked
?1?l?l?l?l?l?l?d?d?d	✓	?l?u	-	3223702	9770834	**22,52%**
?1?2?2?2?2?2?2	✓	?l?u	?l?d	4252237	10799378	**24,89%**
?1?l?l?l?l?2?2?2	X	?l?u	?s?d	615779	7162920	**16,5%**
?1?l?l?l?2?2?2	X	?l?u	?s?d	183101	6730242	**15,51%**
?1?l?l?2?2?2	X	?l?u	?s?d	162429	6709570	**15,46%**

Table 8: Results of mask attacks. The table shows the mask used, the definitions of the ?1 and ?2 custom char sets, how many passwords were cracked using the mask, the total number of passwords cracked using the mask and PassGAN, and what percentage of passwords was cracked using the mask and PassGAN

As can be seen in Table 8, the masks which made use of the -i option, specifically the one which had more options per character, performed better when used alongside PassGAN as compared to

the mask with fewer option per character. This is in spite of the fact that the better performing mask being limited to passwords of length 8 or less, as compared to the worse performing mask which could crack passwords of length 10.
The masks that were ran without the -i option also show that password structures that consist of a short string consisting of an upper case letter and multiple lower case letters followed by 3 digits or special characters were uncommon and unlikely to be a large factor in the underperformance of the first mask.
The results imply that masks with more options per character improve performance when used alongside PassGAN as compared to using longer masks with fewer options per character though this could also be a bias resulting from the dataset having been partially cracked as it is possible that this was the kind of password structure is easier too crack using tools like Hashcat.

5.3 Experiment 3

The goal of this experiment is to evaluate how effective a Markov attack is when combined with a PassGAN attack. For this experiment PassGAN was first used to attempt to crack the LinkedIn dataset. Then a Markov-model was trained on the RockYou dataset by using the statsprocessor tool[Hase]. Then the sp64.bin program included with the statsprocessor tool was used generate words using the Markov-model and these generated words were pipelined into Hashcat so it could execute a dictionary attack on the remainder of the LinkedIn dataset. The result of this experiment can be seen in Table 9 and Figure 6.

	Results
Total unique passwords	43354871
Cracked by PassGAN	6516419
Cracked by Markov-model based attack	3329023
Total passwords cracked	9876164
Total passwords cracked %	**22,78%**

Table 9: This table shows how many passwords were cracked by the Markov-model based attack, the total number of passwords cracked, and the total percentage of passwords cracked.

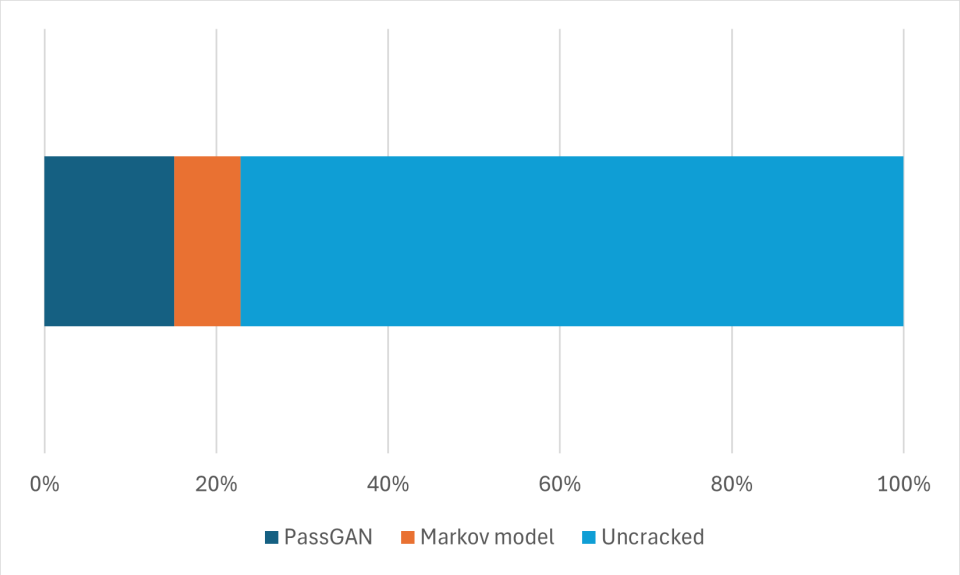

Figure 6: Graph visually showing how many passwords were cracked by PassGAN, the Markov-model based attack, and how many passwords were uncracked.

As can be seen in Table 9 PassGAN was able to crack 6547141 (15,1%) passwords from the LinkedIn dataset, and using the Markov-model another 3329023 for a total of 9876164 (22,76%) passwords cracked. Comparing this to the results from the mask attack it can be seen that an attack making use of a Markov-model can outperform a mask attack depending on the mask used.

5.4 Experiment 4

This experiment seeks to evaluate which rule set is able to crack the highest number of passwords when used along side PassGAN. Five mangling rulesets were tested during this experiment, all used the RockYou dataset as their wordlist. The mangeling rules tested were: best64, D3ad0ne, dive, OneRuleToRuleThemStill, and T0XlC. The dataset being cracked is the LinkedIn dataset.
For this experiment PassGAN was the first tool used to crack passwords in order to create a dataset of passwords from the LinkedIn dataset that PassGAN was not able to crack. PassGAN was able to crack 6516419 passwords from the LinkedIn dataset.
After this Hashcat was used to crack the dataset containing the passwords PassGAN was unable to crack. Table 10 shows how many passwords each ruleset was able to crack when used in conjunction with PassGAN.

Rule used	Rule cracked	Total cracked	Total percentage cracked
T0XlC	9512089	16028508	**37,01512159%**
OneToRuleThemStill	19374274	25890693	**59,74659606%**
dive	19050971	25567390	**59,00141091%**
D3ad0ne	16503119	23019538	**53,12883473%**
best64	5260578	11776997	**27,21576031%**

Table 10: This table shows the mangling ruleset used in the first column, and how many passwords were cracked using it in the second column. The third column shows how many passwords were cracked in total by both PassGAN and the rule-based attack. Lastly the fourth column shows the total percentage of the LinkedIn dataset that was cracked.

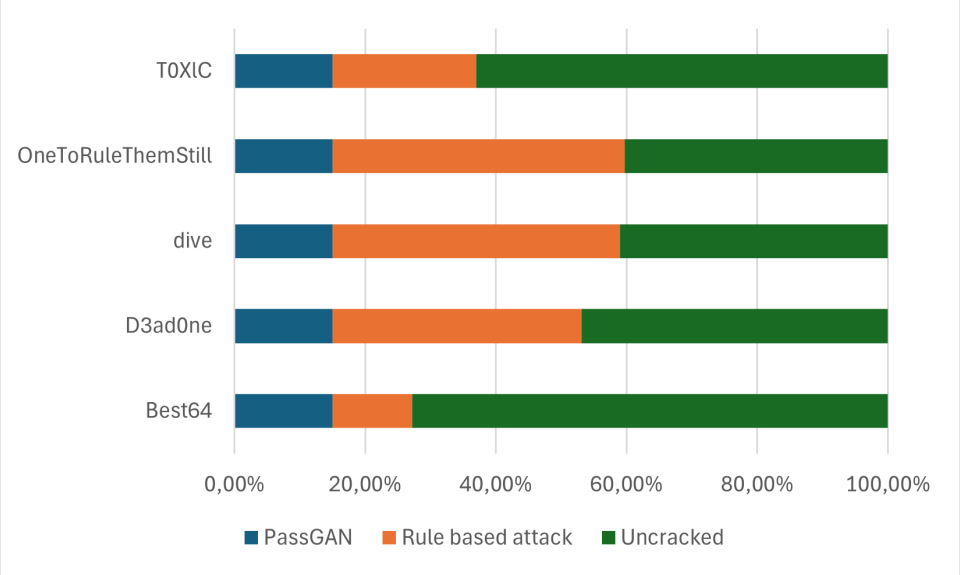

Figure 7: Graph visually showing the percentage of passwords were cracked using PassGAN, using each mangling ruleset when executing a rule-based attack using the RockYou dataset as a wordlist, and how many were left uncracked.

As can be seen in Table 10 the performance of different mangling rulesets varies. Best64 is by far the worst performing only able to crack 14,28% of the passwords PassGAN was unable to crack. The next best performing rulesets was T0XlC, then D3ad0ne which were able to crack 25,82% and 44,8% of the remaining passwords respectively. Lastly the two best performing rulesets were dive and OneToRuleThemStill which performed similarly with dive cracking 51,71% of the remaining passwords and OneToRuleThemStill cracking 52,59% of the remaining passwords.
This means that all mangling rulesets were able to outperform any individual mask and perform better than a Markov-model based attack when used alongside PassGAN.

5.5 Experiment 5

The goal of this experiment was to examine whether a wordlist generated by PassGAN could outperform a human generated wordlist. The benefit of using PassGAN to generate wordlists would

be twofold. Firstly using PassGAN in this way could eliminate the need for using leaked lists of passwords, the use of which is not the best ethically as these include real passwords used by real people and these passwords could contain sensitive information such as full names or dates of births. Second is that PassGAN does not have a limit on how many passwords it is able to generate meaning that it could allow for the creation of larger wordlists which allow for greater password coverage.

During this experiment the number of passwords cracked using a wordlist of 200.000.000 passwords, all of which unique, generated by PassGAN, was compared to the performance RockYou dataset. The performance being measured is the number of passwords cracked when these wordlists are used in a rule-based attack against the LinkedIn dataset. The mangling ruleset used was dive. Table 11 and Figure 8 show the results of this experiment.

Wordlist	**Cracked**	**Cracked percentage**
PassGAN	11308545	**26,06520787%**
RockYou	15958658	**36,78331192%**

Table 11: Number of passwords cracked per wordlist

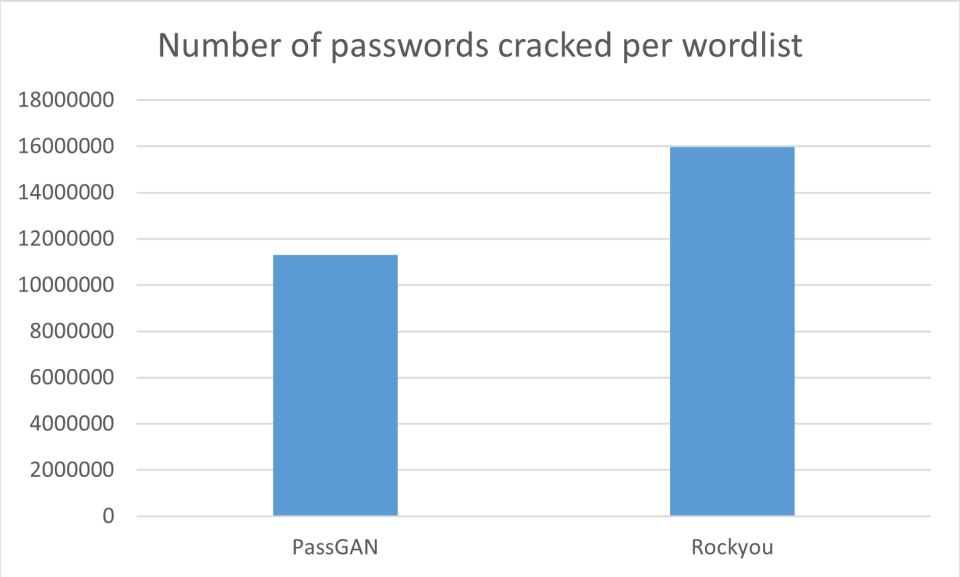

Figure 8: Graph of the number of passwords cracked per wordlist

As can be seen in Table 8, PassGAN performed far worse than the existing RockYou dataset, even though the wordlist generated by PassGAN was larger than the RockYou dataset. This means that PassGAN would likely require a larger list than this to match the performance of the RockYou dataset. Doing so has a large downside. A larger dataset would mean more time is spend cracking when compared to the RockYou dataset.

6 Discussion

6.1 Comparisons

The various methods tested had varying results when used alongside PassGAN. Best performing were the rule-based attacks, being able to crack between 27,21% and 59,75% of the target dataset when used alongside PassGAN and when using the RockYou dataset as a dictionary. Even the worst performing mangling ruleset outperformed other methods when used alongside PassGAN.
After the rule-based attacks the best performing type of attack were mask attacks. The best performing mask tested made use of Hashcat -i option and was able to crack 24,89% of the target dataset when used alongside PassGAN. This performance is quite lackluster when compared to the rule-based attacks tested. And while the mask attacks had a limit for how long they could take as every run increases the amount of time necessary at least tenfold, which limited what masks could be tested and the results of the test, it should be noted that even the high performing mangling rulesets did not take over 4 hours to execute an attack making rule-based attack options more time efficient to use as well. However it should be noted that mask based attacks are depended on the masks used and it is possible that masks can be created that can outperform the ones used in this thesis.
The attack that made use of the Markov-model trained on the RockYou dataset had performance comparable to the mask attacks which made use of Hashcat -i option, cracking 22,76%. However, compared to the mask attacks it took considerable time to achieve this result, requiring 40 hours to execute the attack. This time does not include the time required to train the model which would increase the time required for this attack further.

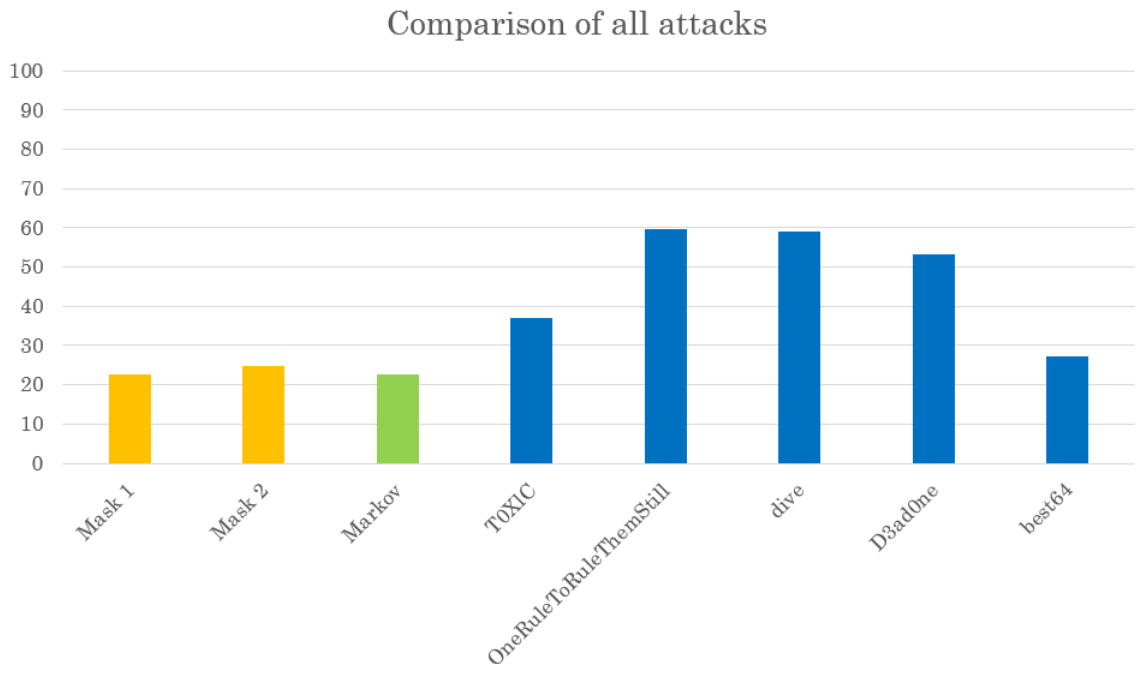

Figure 9: Graph visually showing the percentage of passwords that were cracked per attack. Includes passwords cracked by PassGAN

In short, as can be seen in figure 9 PassGAN used in tandem with rule-based attack are both more effective and faster to execute than using either a Markov-model based attack or a mask attack.

6.2 Ethical considerations

There are some ethical considerations that need to be made when using password lists that have been leaked, as is the case with both the RockYou and LinkedIn datasets. However as this data is old, not linked to any usernames, and publicly available it is unlikely that any user affected by this leak will be harmed by this use of their data as it is only used to train and test a model.

6.3 Methodological limitations

This thesis attempts to evaluate the use of three methods alongside PassGAN. Those being mask atacks, Markov-model based attacks, and rule-based attacks. There exist other methods which could replace either PassGAN or one of the methods executed by Hashcat. For example using another neural network or a method using Probabalistic Context Free Grammars instead of PassGAN.

The only PassGAN model trained for this thesis was trained using a part of the RockYou dataset. This means that performance might be improved by training PassGAN using a larger dataset like the LinkedIn dataset or a custom dataset which adds multiple datsets together to possibily increase passwords coverage. The number of passwords generated by PassGAN also affect performance [HGAPC19] meaning that PassGAN could have performed better had more passwords been generated.
A limitation presents for experiments 2, 3, and 4 is that PassGAN was always used first to crack part of the target dataset meaning that any overlap in password coverage between PassGAN and the other method being tested will be attributed to PassGAN. This order of attacks was chosen mainly to save time as PassGAN only needed to be used to crack a dataset once and this result could be used in all of the tests. The order does not affect the end results and this thesis seeks to examine combination attacks using PassGAN and one other method, which method does what part of the cracking is not relevant to this thesis.

This thesis also did not evaluate a large variety of masks nor was an attempt made to create a mask with the goal of best capturing the password space not captured by PassGAN. The masks which were tested using Hashcats -i function also did not include any special characters.
This thesis also only considered a limited number of mangling rules, specifically those that come standard with either Hashcat or john the Ripper as these tools were created with the goal of being effective passwords crackers and it is thus likely that those included with these tools are among the more effective mangling rulesets available. Though it is possible that a custom mangling ruleset could be created which better complements the passwords space covered by PassGAN.

The PassGAN model was also only trained on passwords of length 10 or less due to memory limitations of the system used to train the model, which does cover a large part of both the RockYou and LinkedIn database but not all of it. When a split of 80/20 of the training and test sets was attempted the computer would freeze and require a forced shutdown before it could function again. This means that this thesis cannot comment on performance of rule-based attacks combined

with PassGAN on passwords longer than 10 characters. This is relevant as longer passwords increase entropy which increases how secure a password is[Mas94].

This thesis also only looks at two older English datasets (RockYou from 2009, LinkedIn probably from around 2012) meaning that results might not be as applicable in a modern setting where there exist more restrictions on what passwords can be created. It is also not able to comment on the effectiveness of the type of attack tested in this thesis when the language isn't English.

6.4 Future research

Future research could focus on testing how PassGAN performs when used to crack passwords in different languages as this thesis can only comment on its effectiveness when used to crack English passwords. Research could also be done on how many passwords PassGAN requires to be an effective option to execute an attack with. This is important as datasets of leaked passwords other than those which originate from websites where English was the main language used are smaller in most cases meaning that it is essential to have a clear understanding of how well the model can perform when having access to a smaller dataset or to develop a model which performs better with a smaller dataset.

Another avenue of future research could be further looking into creating masks or mangling rulesets which better complement PassGAN with the goal of creating efficient rules and masks to be used alongside it. This could allow for even better performance than that found in this thesis.

A PassGAN model trained on a wider dataset that includes a larger variety of passwords and includes some known common passwords could improve performance of the PassGAN model. For example a model trained on both the RockYou and full LinkedIn dataset might be able to generate a larger variety of passwords as compared to the model trained for this thesis.

7 Conclusion

In this thesis PassGAN was used alongside several traditional password cracking methods to crack two specific password datasets. When used in tandem with traditional password cracking methods, PassGAN shows improved results compared to when it is used on its own. PassGAN was able to generate a large number of unique passwords that matched passwords present in the tested datasets.

PassGAN is suitable for generating large wordlists for dictionary attacks but does not appear to offer an advantage when used to attempt to generate an efficient wordlist to crack passwords. Even though the RockYou dataset was smaller than the wordlist created by PassGAN the RockYou dataset performed better when used for a rule-based attack as compared to the wordlist created by PassGAN.

There are significant advantages to using PassGAN alongside other password cracking methods in a multistage password cracking attack.

References

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[AS99] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Commun. ACM*, 42(12):40–46, dec 1999.

[BSB18] L. Bošnjak, J. Sreš, and B. Brumen. Brute-force and dictionary attack on hashed real-world passwords. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1161–1166, 2018.

[DBC+14] Anupam Das, Joseph Bonneau, Matthew C. Caesar, Nikita Borisov, and Xiaofeng Wang. The tangled web of password reuse. In *Network and Distributed System Security Symposium*, 2014.

[Dhi67] Harpreet Singh Dhillon. Second order markov model based proactive password checker. *Department of Electronics and Communication Engineering, IIT Guwahati, India., Roll*, (04010214), 1967.

[FH06] Dinei Florencio and Cormac Herley. A large scale study of web password habits. Technical Report MSR-TR-2006-166, November 2006.

[GAA+17a] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[GAA+17b] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[Hasa] Github repository containing best64 mangling ruleset (accessed 1/9/2024). https://github.com/hashcat/hashcat/blob/master/rules/best66.rule.

[Hasb] Github repository containing d3ad0ne mangling ruleset (accessed 1/9/2024). https://github.com/hashcat/hashcat/blob/master/rules/d3ad0ne.rule.

[Hasc] Hashcat website. dictionary attack page. (accessed 1/9/2024). https://hashcat.net/wiki/doku.php?id=dictionary_attack.

[Hasd] Github repository containing dive mangling ruleset (accessed 1/9/2024). https://github.com/hashcat/hashcat/blob/master/rules/dive.rule.

[Hase] jsteube, philsmd, bkerler, unix-ninja. (oct 27, 2023), github for statsprocessor (accessed 1/9/2024). https://github.com/hashcat/statsprocessor.

[Hasf] Hashcat website. mask attack page. (accessed 1/9/2024). https://hashcat.net/wiki/doku.php?id=mask_attack.

[Hasg] stealthsploit, 0xvavaldi. github repository containing onerule-torulethemstill mangling ruleset (accessed 1/9/2024). https://github.com/stealthsploit/OneRuleToRuleThemStill.

[Hash] Hashcat website. rule-based attack page. (accessed 1/9/2024). https://hashcat.net/wiki/doku.php?id=rule_based_attack.

[Hasi] Github repository containing t0xlc mangling ruleset (accessed 1/9/2024). https://github.com/hashcat/hashcat/blob/master/rules/T0XlC.rule.

[Hasj] Hashcat website (accessed 1/9/2024). https://hashcat.net.

[Hask] Password security: past, present, future (accessed 20/1/2024). https://www.openwall.com/presentations/Passwords12-The-Future-Of-Hashing/.

[HGAPC19] Briland Hitaj, Paolo Gasti, Giuseppe Ateniese, and Fernando Perez-Cruz. Passgan: A deep learning approach for password guessing. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 217–237, Cham, 2019. Springer International Publishing.

[HNLP18] Quan Hoang, Tu Dinh Nguyen, Trung Le, and Dinh Phung. Mgan: Training generative adversarial nets with multiple generators. In *International conference on learning representations*, 2018.

[HVO12] Cormac Herley and Paul Van Oorschot. A research agenda acknowledging the persistence of passwords. *IEEE Security Privacy*, 10(1):28–36, 2012.

[Joh] John the ripper website (accessed 1/9/2024).

[Lin] brannondorsey. github containing the linkedin dataset (accessed 1/9/2024). https://github.com/brannondorsey/PassGAN/releases.

[Mas94] J.L. Massey. Guessing and entropy. In *Proceedings of 1994 IEEE International Symposium on Information Theory*, pages 204–, 1994.

[Nga12] Chenda Ngak. 6.5 million linkedin passwords reportedly leaked on russian hacker site, Jun 2012.

[NIS] Nist hash functions (accessed 20/1/2024). https://csrc.nist.gov/projects/hash-functions.

[NS05] Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. pages 364–372, 11 2005.

[Pasa] beta6. github repository for python2.7 version of passgan (accessed 1/9/2024). https://github.com/brannondorsey/PassGAN.

[Pasb] beta6. github repository for python3.8 version of passgan (accessed 1/9/2024). https://github.com/beta6/PassGAN.

[Pro] Niels Provos. Bcrypt at 25: A retrospective on password security (accessed 20/1/2024). https://www.usenix.org/publications/loginonline/bcrypt-25-retrospective-password-security.

[RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009.

[Roc] Site hosting the rockyou dataset (accessed 1/9/2024). https://www.kaggle.com/datasets/wjburns/common-password-list-rockyoutxt.

[sec] Stallings, william, and lawrie brown. 2018. computer security : Principles and practice. 4th edition, global edition. new york: Pearson.

[SKD+16] Richard Shay, Saranga Komanduri, Adam L. Durity, Phillip (Seyoung) Huh, Michelle L. Mazurek, Sean M. Segreti, Blase Ur, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Designing password policies for strength and usability. *ACM Trans. Inf. Syst. Secur.*, 18(4), May 2016.