BACHELOR PAPER

Term paper submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Information and Communication Systems and Services

Measuring Password Quality with Natural Language encoders

By: Benjamin Akhras

Student Number: 1310258008

Supervisor: Dipl-Ing. Mag. Dr. Priv.-doz. Edgar Weippl

Vienna, July 21, 2017

FH University of Applied Sciences TECHNIKUM WIEN

Declaration

Vienna, July 21, 2017 Signature

Kurzfassung

In dieser Arbeit wird eine neue Methode zur Messung der Qualität von Passwörtern vorgestellt. Die dafür verwendete Methode basiert auf Natural language processing. Berücksichtigt werden aber ebenfalls simple hybride Wörterbuchattacken, bis hin zu gezielten Attacken auf einen einzelnen Benutzer. In der Einleitung wird eine Übersicht über die die verschiedensten Angriffe auf Passwörter gegeben und möglichkeiten aufgezeigt passwörter gegen diese zu stärken. Weiters werden weit verbreitete Fehlkonfigurationen welche solche attacken ermöglichen diskutiert. Im Rahmen dieser Arbeit wurde ein Proof of Concept in golang entwickelt, um einen Benutzer eine Bewertung seines Passwörter über Entropie hinaus zur verfügung gestellt. Ein "proof of concept" in form einer golang Applikation wurde im Rahmen dieser Arbeit entwickelt um zu zeigen, dass Passwortqualitäts Rückmeldungen nicht ausschließlich Entropie basierend sein muss. Diese Software wurde gegen zwei Datensets aus Passwörtern, welche öffentlich zur verfügung stehen, getestet. Die Ergebnisse dieser Arbeit können verwendet werden, um bestehende Software dahingehend anzupassen und Benutzern eine bessere Rückmeldung bezüglich der Qualität ihrer Passwörter zu geben, damit nicht nur Angreifer, sondern auch Benutzer von Natural Language encodern profitieren können.

Abstract

This paper describes a new way to measure the quality of passwords. This method accounts for attacks based on Natural language processing. These attacks can range from simple hybrid dictionary attacks, to targeted attacks towards a specific user with an old known password. But no matter how sophisticated a password cracking attack is in the end it is just some sort of guessing. The goal behind new methods of password cracking is always to decrease the number of password guesses needed. Therefore in the introduction an overview of different password attacks and how to strengthen a password against them is given. But to strengthen a password one can not only increase it's entropy. Unfortunately most people follow common patterns while creating their passwords, and stick to those patterns when it comes to increasing entropy. Furthermore common misconfigurations that additionally empower these attacks as password policies are discussed. A proof of concept in the form of a golang application was written to show that password quality feedback to a user is not solely possible through entropy. It was tested against two Datasets of passwords which were publicly available. The results in this paper can be used to alter tools giving users better feedback about the quality of their passwords, to have not only attackers benefit from Natural Language encoders but the user.

Keywords: password, nlp, quality, dawg, PCFG, CFG, grammar, security

Contents

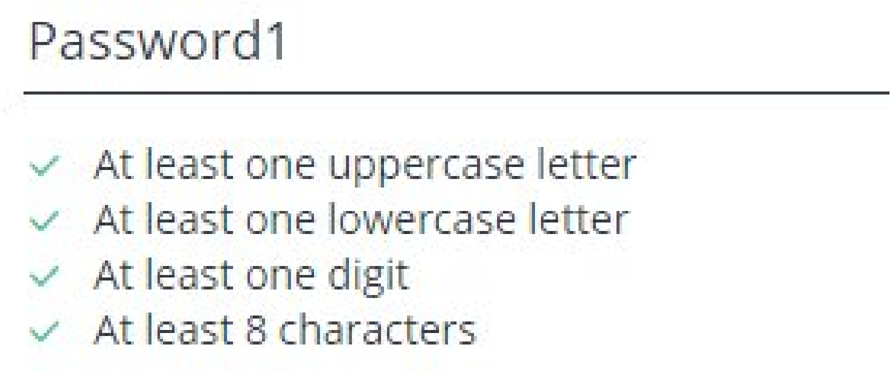
Contents

1 Introduction & Background

Authentication is an important task most computer systems need to solve. Nowadays authentication is done through the use of at least two of the factors knowledge, ownership and inherence. This paper will only address one of these factors, knowledge. This factor usually is implemented in the form of a password, but it is necessary for the security of a password that it remains secret. To use passwords in a secure manner multiple problems must be addressed. The password has to be stored in a secure way, a salted hash for example. A cryptological hash function is a special form of the hash function, which should be collision-resistant and, by definition, always a one-way function.

A hash function is a function that maps a string of any length to a fixed-length string. Applications of cryptological hash functions are mainly data processing, integrity checking of files or messages. Or as in our use case, they are used to conceal password files to increase security if the password file gets compromised. If a password is going to be transmitted via a network this has to happen in a secure manner to ensure it remains a secret. A salt is a randomly selected string in cryptography that is appended to a given cleartext before use as an input of a hash function to increase the entropy of the input. It is used for storing computer passwords, the salt is generated and stored together with the resulting hash value in a database. For example if the password is used as authentication on a website, TLS encryption has to be used to ensure that your password actually will be transmitted to the correct website, since TLS not only encrypts your message but also ensures authenticity. The issues in this paper do not only apply to passwords for web services but to any computer system. Nowadays security experts recommend the use of password vaults sometimes called wallets.[2]

But even if these preconditions are met, the secret password itself can still be an attack vector. In this work a new approach to whether passwords quality can be determined on basis of how modern password attacks work is given. A final implementation of this could happen inside a password vault like keepassx or pwgen.


Password1

✓ At least one uppercase letter

✓ At least one lowercase letter

✓ At least one digit

✓ At least 8 characters

Figure 1: A common password quality meter

1.1 Password quality

The quality of a password is represented in a function of length, complexity and unpredictability. Usually we scale passwords via length and complexity every few years. This must happen as the rate at which systems can attack (guess) passwords are getting higher every year.

A problem with passwords is the so called „Snake Oil“ industry. [10] There is an whole industry that provides pseudo-mathematical nonsense in an effort to sell their products. Users are often overwhelmed with information and advertising from an industry trying to sell security products. Strong passwords can not be sold, so they are not part of this information and therefore often forgotten about. Companies and even private households spend money on Anti-virus software and Firewalls that can not protect them if their passwords are insecure. If for example they want to know if their password 'Password1' is secure, they enter it on the website they want to use it on, and the website confirms that it is secure. Most 'passwordquality' checks nowadays check it like this:

But as explained later on, under dictionary attack this is one of the worst passwords possible.

Another misconception is that rotating passwords more often makes them more secure. The National Cyber Security Centre, a UK Goverment Organization already stated this misbehavior.[8] Password rotations do not increase security. Similar passwords will be used substrings get rotated, numbers increased or the one special character moves. These patterns are the main attack surface on passwords nowadays and should be avoided, not promoted.

In which situations does a forced-password-change actually increase security? If an attacker actually got hold of a password, but is not able to either:

- extract all data for example via a database dump
- is not able to install any kind of backdoor

- is not able to retrieve the password again

For the last point the question of how the attacker got the password in the first place is important. He may have gathered the password directly via phishing or via a Keylogger. A paper from 2010 by Yinqian Zhang is one of the most complete works concerning password reuse.[18] He was able to proof that password reuse hurts security more than it benefits. But there was proof of that this is a bad idea since 1999: A paper with the perfectly matching title 'The User is not the Enemy' by Adams and Sasse[1].

> To date, research on password security has focused on designing technical mechanisms to protect access to systems; the usability of these mechanisms has rarely been investigated. [...] Since security mechanisms are designed, implemented, applied and breached by people, human factors should be considered in their design. It seems that currently, hackers pay more attention to the human link in the security chain than security designers do, e.g. by using social engineering to obtain passwords.

Unfortunately this is still a problem. For example administrators forcing password policies on users they do not have to obey themselves. In the last years practices as infrastructure as Code (IaC) and Infrastructure as a Service IaaS made it possible to analyze and eliminate these behaviors.

So configuration management tools are a major tool not only in increasing usability but in increasing security.

1.2 Password cracking

Password cracking is as old as passwords themselves. There are numerous papers describing various techniques for password cracking. The problem is the lack of information a typical user is able to get about how to protect himself from them. From the view of the password cracker the weakest link is always the approach to crack the password. The weakest link usually isn't the entropy. A Brute-force approach usually is the last thing an attacker resorts to.

1.2.1 Brute-force attack

> (I) A cryptanalysis technique or other kind of attack method involving an exhaustive procedure that tries a large number of possible solutions to the problem. (See: impossible, strength, work factor.)
>
> Tutorial: In some cases, brute-force involves trying all of the possibilities. For example, for cipher text where the analyst already knows the decryption algorithm, a brute-force technique for finding matching plain text is to decrypt the message with

Table 1: Maximum calculationtime for a brute-force attack at 1 billion keys per second

codespace	10 [0-9]	26 [a-z]	52 [A-z]	62 [A-z;0-9]	96 +special chars
4 characters	<1m second	<1 second	<1 second	<1 second	<1 second
5 characters	<1m second	<1 second	<1 second	<1 second	8 seconds
6 characters	1m seconds	<1 second	20 seconds	58 seconds	13 minutes
7 characters	10m seconds	8 seconds	17 minutes	1 hour	21 hours
8 characters	100m seconds	4 minutes	15 hours	3 days	84 days
9 characters	1 second	2 hours	33 days	159 days	22 years
10 characters	10 seconds	2 days	5 years	27 years	2108 years
11 characters	2 minutes	42 days	238 years	1649 years	202k years
12 characters	17 minutes	2 years	12.4k years	102k years	19M years

> every possible key. In other cases, brute-force involves trying a large number of possibilities but substantially fewer than all of them. For example, given a hash function that produces an N-bit hash result, the probability is greater than 1/2 that the analyst will find two inputs that have the same hash result after trying only 2**(N/2) randomly chosen inputs. [13]

To protect a password from a brute-force attack one can just increase the password's length. This will increase the password's entropy and therefore makes it harder to crack via brute-force.

In the following table 1 you can see how easily when a brute-force attack is applicable and when not.

From this table it is easy to see that it is quite easy to create a password that is not vulnerable to a Brute-force attack. But since the brute-force is always the **last** approach from the view of an attack it is absolutely irrational to see it as the first approach to measure the quality of your password.

1.2.2 Dictionary attack

The Internet Security Glossary describes a Dictionary attack as:

> (I) An attack that uses a brute-force technique of successively trying all the words in some large, exhaustive list. Examples: Attack an authentication service by trying all possible passwords. Attack an encryption service by encrypting some known plaintext phrase with all possible keys so that the key for any given encrypted message containing that phrase may be obtained by lookup.[13]

So it is very similar to a brute-force attack, but the passwords used for the brute-force aren't generated but read from a database. This database is called the dictionary. The first dictionary attacks used just standard dictionaries. But nowadays it is more common to use passwords from password leaks. As a thesaurus does not provide nearly as good results.

1.2.3 Hybird attack

A hybrid attack is a combination of these two attacks. On the one hand you have a dictionary but you extend it with the result of a brute-force attack on the other hand. So usually you take a word from the dictionary and append or prepend a character from the Brute-Force keyspace. Even common replacements of characters are possible as seen for example in leetspeak. So the word 'eleet' from the dictionary would be replaced by 31337 or 3l33t. [5]

1.3 Letter frequency

Letter frequency analysis is one of the first methods[12] described to break classical ciphers.

The letter frequency is a statistical quantity indicating either the absolute number of appearances of a specific letter in a text or this number in relation to the total number of the letters in a text. Counts on the frequency of letters or sounds in texts or text corpora have been detected since the early 19th century at the latest. While earlier assumptions generally thought to predict the statistical distribution of the frequency of letters by the Zipf's law. The frequency distribution of the letters depends on the natural language of the text. Passwords have their own artificial language, this language will be analyzed below.

For these analysis it could also be interesting to see how often a letter appears at the beginning, in the middle or at the end of a password.

In the following two tables the letter frequency of the English language is compared with the letter frequency within the analyzed passwords. First you will notice that the frequency on the password site does not add up to 100% The probability that a character is a letter is only about 55%. But for generating a PCFG to generate passwords the resulting grammar will have to consider a lot more information. Not only single letters but tuples and triples, have to get analyzed. But furthermore the position of each of these. For example the probability for the first letter of an English word to be an a is 11.6%. Whereas the probability for an a overall the word is only 8.167%.

Table 2: Relative frequencies in the english language[11]

letter	frequency english
e	12.702%
t	9.056%
a	8.167%
o	7.507%
i	6.966%
n	6.749%
s	6.327%
h	6.094%
r	5.987%
d	4.253%
l	4.025%
c	2.782%
u	2.758%
m	2.406%
w	2.360%
f	2.228%
g	2.015%
y	1.974%
p	1.929%
b	1.492%
v	0.978%
k	0.772%
j	0.153%
x	0.150%
q	0.095%
z	0.074%

letter	frequncy password
a	6.6006274 %
e	6.2969117 %
i	4.2088804 %
o	4.2038164 %
r	4.1757946 %
n	4.0893664 %
s	3.938884 %
l	3.3826761 %
t	3.2035873 %
m	2.5830574 %
c	2.314984 %
d	2.2931213 %
h	2.092243 %
u	1.8772438 %
b	1.8341757 %
y	1.8046929 %
g	1.6083426 %
p	1.5811483 %
k	1.5549033 %
w	1.0565172 %
f	0.9466438 %
j	0.91134226 %
v	0.82878506 %
z	0.4604551 %
x	0.40925556 %
q	0.28185317 %

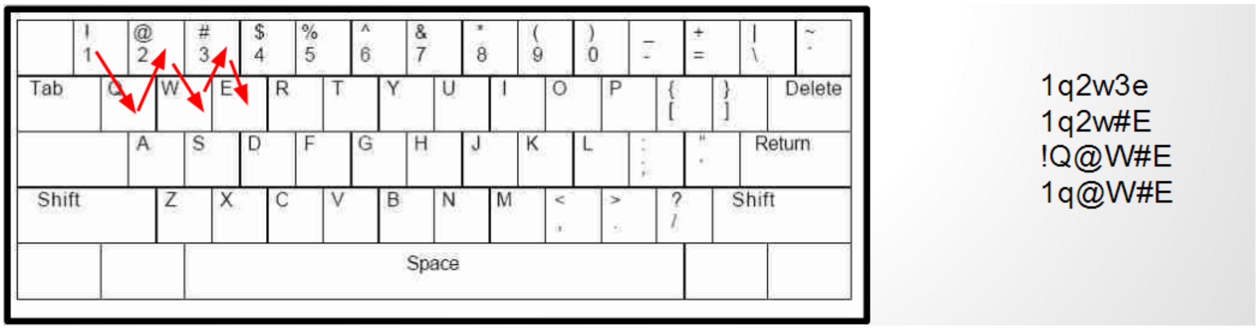


Figure 2: Keyboard walk on QWERTY[9]

1.4 Keyboard layout

The keyboard layout describes the coding of the individual keys as well as their location on the keyboard of a computer. In principle, a distinction must be made between the physical allocation which usually is nothing more than the printing on the keys, and the variable keyboard layout which can be adapted via software, and therefore easily switched.

Keyboard layouts often use the QWERTY-labeled arrangement of the letter buttons. There are certain keyboard layouts as Neo-layout for the German language and the Dvorak layout the for English language that considered these frequencies while placing their keys. But QUERTY or QUERTZ does not, it was designed in 1886 by the American printer and newspaper publisher Christopher Latham Sholes. One of its design goals was that it's easy to type the word 'Typewriter'. The other more important design goal was to separate tuples apart to decrease the likelihood of the typewriter to jam. Which makes it especially hard to learn and to type on keyboard layout for computers. Due to the popularity of QUERTY-labeld keyboards a lot of common password patterns naturally derive from the keyboard layout on which they are created. These patterns don't have to be English language they follow only the layout. It is common that passwords follow a specific pattern on the keyboard layout they are created on. But this can also just manifest in the fact that certain characters that are hard to reach aren't used as often. These patterns have been already used for attacks, and are generated by all common password crack tools.

So since passwords are an artifical language generated on and for a specific keyboard layout the letter frequency of passwords shifts towards easy to reach characters on this layout.

Table 3: Relative frequencies in the english language

letter	frequency	letter	frequency
th	1.52 %	ar	0.04 %
en	0.55 %	nt	0.56 %
ng	0.18 %	ti	0.34 %
he	1.28 %	ve	0.04 %
ed	0.53 %	ha	0.56 %
of	0.16 %	as	0.33 %
in	0.94 %	ra	0.04 %
to	0.52 %	es	0.56 %
al	0.09 %	te	0.27 %
er	0.94 %	ld	0.02 %
it	0.50 %	st	0.55 %
de	0.09 %	et	0.19 %
an	0.82 %	ur	0.02 %
ou	0.50 %		
se	0.08 %		
re	0.68 %		
ea	0.47 %		
le	0.08 %		
nd	0.63 %		
hi	0.46 %		
sa	0.06 %		
at	0.59 %		
is	0.46 %		
si	0.05 %		
on	0.57 %		
or	0.43 %		

2 Natural Language Processing

Natural language processing (NLP) describes methods and algorithms to help computers understand human languages. Modern approaches to achieve this are using a method called 'machine learning'. It is a field in computer science that is closely related to statistics. The idea is that a 'learner' is able to generalize from a specific experience. This concept can be applied to software via various ways. For example artificial neural networks modeling the way the human brain solves problems using a collection of neural units. In this paper the approach of an Bayesian network will be used.

Specifically a Markow network which is an undirected and possibly cyclic version of an Bayesian network.

2.1 Context-free grammar

A grammar can be regarded as a device that enumerates the sentences of a language.[4]

Noam Chomsky

But can it be hacked to enumerate the possibilities of a password? Recent attacks as these in 2011 on lastpass showed it is not only possible but already happened. But what if we can use these grammars not to attack passwords but to make them more secure?

A context-free grammar (CFG) is a term used in formal language theory to describe a certain type of formal grammar. A context-free grammar is a set of production rules that describe all possible strings in a given formal language. Production rules are simple replacements.

In context-free grammars, all rules are one to one, one to many, or one to none. These rules can be applied regardless of context. The left-hand side of the production rule is also always a nonterminal symbol. This means that the symbol does not appear in the resulting context-free language (CFL). Rules can also be applied in reverse to check if a string is grammatically correct according to the grammar. [4]

2.1.1 Suffix automaton

A suffix tree is a way of structuring data for later processing. It structures the information in a tree where every leaf represents a suffix of the input. A DAWG (deterministic acyclic word graph) derived from the String 'password' for example is an automaton able to recognize all the substrings of 'password'. 'passwor', 'passwo', 'passw', 'pass', 'pas', 'pa', 'p' are all seen als valid by this automaton. Every other string is invalid and would return false.[7]

An example in a visualized form with the input 'pass1':

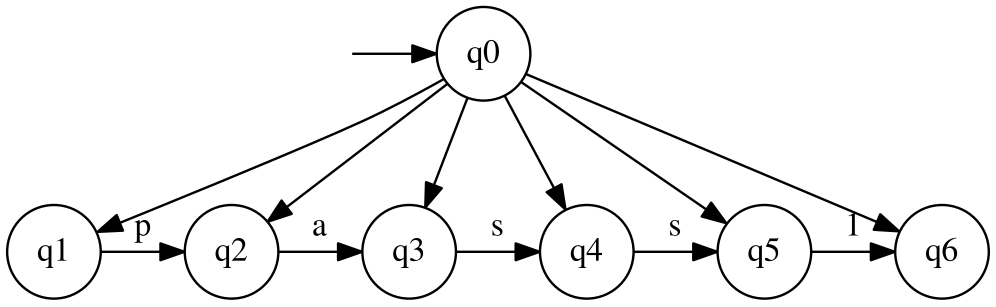


Figure 3: A Suffix tree

2.2 Probabilistic context free grammars

Probabilistic context free grammars (PCFG) add probabilities to context free grammars.

Grammar theory to model symbol strings originated from work in computational linguistics aiming to understand the structure of natural languages. Probabilistic context free grammars (PCFGs) have been applied in probabilistic modeling of RNA structures almost 40 years after they were introduced in computational linguistics.

PCFGs extend context-free grammars similar to how hidden Markow models extend regular grammars. Each production is assigned a probability. The probability of a derivation (parse) is the product of the probabilities of the productions used in that derivation. These probabilities can be viewed as parameters of the model, and for large problems it is convenient to learn these parameters via machine learning. A probabilistic grammar's validity is constrained by context of its training dataset.

PCFGs have application in areas as diverse as natural language processing to the study the structure of RNA molecules and design of programming languages. Designing efficient PCFGs has to weigh factors of scalability and generality. Issues such as grammar ambiguity must be resolved. The grammar design affects results accuracy. Grammar parsing algorithms have various time and memory requirements.

Derivation: The process of recursive generation of strings from a grammar.
Parsing: Finding a valid derivation using an automaton.
Parse Tree: The alignment of the grammar to a sequence.

An example of a parser for PCFG grammars is the pushdown automaton. A more efficient alternative to the pushdown automaton is the Cocke–Younger–Kasami (CYK) algorithm (CYK) algorithm which is described in detail later on in this paper.

A PCFG can be defined by following quintuple: $G = (M, T, R, S, P)$
where
M is the set of non-terminal symbols
T is the set of terminal symbols
R is the set of production rules
S is the start symbol
P is the set of probabilities on production rules

2.2.1 Markow chain

The Russian mathematician Andrey Markow defined a special stochastic process that satisfies the Markow property. A Markow chain (Markow chain, Markof chain, Markoff chain) is a special stochastic process. A Markow chain is defined by the fact that knowledge of a limited history is just as good as predictions of future development as well as knowledge of the entire history of the process. Markow chains of different order are distinguished. In the case of a first-order Markow chain, this means that the future of the system depends only on the present (the current state) and not on the past. The mathematical formulation in the case of a finite state quantity requires only the notion of discrete distribution as well as the conditional probability, whereas the concepts of filtration and conditional expectation are needed in the case of time. The aim of using Markow chains is to specify probabilities for the occurrence of future events. The terms Markow chain and Markow process are generally used synonymously.

To some extent, however, processes are defined in a discrete time (discrete state space) for the delimitation with Markow chains, and processes with steady process time (steady state space) with Markow processes.

2.3 Cocke–Younger–Kasami algorithm

The Cocke-Younger-Kasami algorithm (CYK-algorithm) is an algorithm that was created in the 1960er years by Itiroo Sakai, John Cocke, Tadao Kasami, Jacob Schwartz and Daniel Younger. With this algorthim it is possible to determine if a specific String in our case a password is part of a CFL. The algorithm in its standard form requires the grammar to be in Chomsky normal form.

2.4 Chomsky normal form

In formal language theory, a context-free grammar G has to fullfil a set rules to be considered as Chomsky normal form. These rules, described by Noam Chomsky in 1959 require all of its production rules are of the form:

$A \to BC$, or
$A \to$ a, or
$S \to \varepsilon$,
where A, B, and C are nonterminal symbols, a is a terminal symbol (a symbol that represents a constant value), S is the start symbol, sometimes also called sentence symbol, and ε denotes the empty string. Also, neither B nor C may be the start symbol, and the third production rule can only appear if ε is in LG, namely, the language produced by the context-free grammar G. Every grammar in Chomsky normal form is context-free, and conversely, every context-free grammar can be transformed into an equivalent one which is in Chomsky normal form and has a size no larger than the square of the original grammar's size.
First generate a new Startsymbol S_0 an the rule $S \to S_0$. This guarantees that the startsymbol does not appear anywhere on the righthandside. Second replace all rules with more than one terminal on the right side with a nonterminal.
$A \to$ ab$C$ becomes
$A \to$ aBC
$B \to$ b
Third righthandsides with more then 2 nonterminals get replaced. Again an example: $A \to$ aBCD becomes
$A \to$ aBE
$E \to CD$

Now all ε-rules have to be eliminated. A ε-rule is in the form:
$A \to \varepsilon$
Where A is not the Startsymbol. Hopcroft and Ullman (1979) call such nonterminals nullable, and compute them as follows: If a rule$A \to \varepsilon$ exists, then A is nullable. If a rule $A \to X_1$ ... X_n exists, and each X_i is nullable, then A is nullable, too.
Last remove all so called UNIT rules. These are rules as: $A \to B$
IF B for example is:
$B \to$ b
you would replace it both of the rules above with:
$A \to$ b

Chomsky is one of the most prominent American linguists of the present, who, through the combination of the scientific disciplines of linguistics, cognitive science and computer science, played a major role in the development of the latter half of the 20th century. His contributions to the general linguistic sciences as well as his models of the generative transformation grammar altered the predominant American structuralism. His criticism of behaviorism promoted the rise of cognitive science.

From the 1960s and 1970s, Chomsky was often present in political and scientific discourse. Since Chomskys criticism of the Vietnam War, he has repeatedly appeared as a keen critic of

US foreign and economic policy and has become known worldwide as a critic of capitalism and globalization. According to the Arts and Humanities Citation Index of 1992, Chomsky was the most cited living person in the world between 1980 and 1992. His contributions are the basis even to this paper.

3 Data

3.1 Password lists

To bring all that knowledge to a use we need passwords. It is easy to get access to password dumps nowadays. The media only covers such releases if it is millions of datasets big. There are bots that crawl the internet for passwords leaks one of the most popular beeing dumpmon[17]. Of course most of the data returned by these bots isn't useful. But the amount of actual passwords is actually still shockingly high. In this paper two different sources were used: 1. The RockYou database taken from https://wiki.skullsecurity.org/Passwords[16]. This dataset was leaked in 2009, and contains over 30 million passwords. Since all passwords were stored in cleartext 100% of these passwords are useable for research. This leak is very well researched, it revolutionized password research back in 2009 since it was the biggest single leak of clear text passwords till this day.

Appendix A

2. The 10 Million Passwords List released by Mark Burnett on xato.net[3] It was collected over multiple years, featuring password dumps from various sources. Including twitter bots, google search alerts. This list is not from a single leak so its passwords are not restricted to any specific password creation rule. It is more of a meta list since it contains multiple passwords leaks over multiple years.

Appendix B

4 Proof of Concept

As a final result a proof of concept application in golang was written.

As a proof of concept a cyk-parser with a context free grammar was created. Its purpose is to show that password quality can be measured in more then just its entropy. I created password masks from the two analyzed sets. At first I tried to give a score on tuples and triples of non terminals. But it showed that this was not as useful, it was always to specific. Getting out that "one master grammar" where every non terminal has a weight would most probably be a very powerful tool since analyzing could be much faster, but creating so would be have been a manual task. So rather than this approach chose a different one, creating multiple password "masks".

A single password mask represents a set of passwords with a specific entropy.

For example "?l?l?l?l?l?d?d" would be 5 letters followed by two digits a very common mask. The syntax I choose is from the very popular and powerful password cracking tool hashcat. So the first part is 5 times ?l, it represents a keyspace from aaaaa to zzzzz. The two ?d would represent a number from 00 to 99.

Probabilities are used on password masks themselves and not on the terminals in the mask. Since it is that bad behavior of using that mask that is to be punished in the password score. The usage of common patterns are especially tuples are common for any random generated string.

- ?l = abcdefghijklmnopqrstuvwxyz
- ?u = ABCDEFGHIJKLMNOPQRSTUVWXYZ
- ?d = 0123456789
- ?s = «space»!"()*+,-./:;<=>?@[]‘|’& % $ # _ { } ~^\
- ?a = ?l?u?d?s
- ?b = 0x00 - 0xff

If generating a new password, one can test it against that tool to get feedback of its quality. But be aware that in the end this tool won’t give you any new insights in most cases. The only secure way is to generate an entirely new passwords if rotating them. This is only applicable for an user if he uses a password manager.

One could summarize all these problems in one sentence:

> Humans are lousy random-number generators

Wrapping it all up, we can see that a password safe is the one most important tool when it comes to password safety. But it is important to actually generate random passwords. At the moment there are a lot of misconceptions out there amplifying the problems. On the one hand IT Managers not understanding how attacks on their infrastructure actually happens. Therefore creating nonsense rules for password length and rotation. On the other hand developers giving the user a wrong feeling about what is secure. There should be less talk about the single broken password ’123456’ but more about broken processes and broken tooling. The "top 10" passwords lists serve for a good laugh but no insight in the strength of the passwords on a specific site at all. One should try helping the users to set one strong password or even better try to use more secure methods as public key authentication or two factor authentication wherever possible. The User is not the enemy but quiet often the system administrator or the developer is, by creating these rules. Pass the Hash attacks[14] show the additional issues that arrive with the use of passwords. Therefore Public-key authentication should be used where possible.

4.1 Statistical Analysis

A. Password length
B. Character Distribution
C. Password Mask

4.2 Password reuse

Another major problem of passwords nowadays is password reuse. One major amplifier for that issue is the wrong use of password rotation policies. You can stop users to reuse the same password they had before again by saving old password hashes. But it is impossible to determine if just one single character in the password changed. Usually this character is a digit that gets incremented. This reduce the whole rotation policy to absurdity, if anyone compromised the account it is still compromised since, the change will be obvious even for an automated attack.

5 Results

Some of the results this paper were already shown by others as [15] It is important to remember that the user or his password can be the weak link in a computer system, but this does not make him the enemy. If attackers think about the "weakpoint user" first when attacking, why don't we think about "users first" when designing security software. The password analysis in this paper showed that most of the "tricks" to generate secure easy to remember passwords won't work out. Why are password policies enforced but not the use of a password manager. There are multiple papers proving the benefits of password managers and multiple proving the disadvantages of password policies.[6]

Attacks on IT infrastructure are already fact based and have a measurable output, as long as security doesn't do the same the attackers will always be one step ahead.

Bibliography

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

[2] Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 553–567, Washington, DC, USA, 2012. IEEE Computer Society.

[3] Mark Burnett. 10 million passwords list. https://xa.to/10m, 2015.

[4] Noam Chomsky. Information and control. pages 137—-167, 1959.

[5] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and Xiaofeng Wang. The tangled web of password reuse.

[6] Zhiwei Li, Warren He, Devdatta Akhawe, and Dawn Song. The emperor's new password manager: Security analysis of web-based password managers. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 465–479, San Diego, CA, 2014. USENIX Association.

[7] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001.

[8] NCSC. password expiry. https://www.ncsc.gov.uk/articles/problems-forcing-regular-password-expiry, 2015. 2016-12-01.

[9] Rich. Keyboard walks. http://bytesdarkly.com/2014/08/generating-keyboard-walks/, 2014.

[10] Bruce Schneier. Snake oil. https://www.schneier.com/crypto-gram/archives/1999/0215.html, 1999. 2016-12-01.

[11] Thomas Schürmann and Peter Grassberger. Entropy estimation of symbol sequences. Technical report. http://arxiv.org/ftp/cond-mat/papers/0203/0203436.pdf.

[12] Claude E. Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. http://doi.acm.org/10.1145/584091.584093.

[13] R. Shirey. Internet security glossary, version 2. RFC 4949, RFC Editor, August 2007. http://www.rfc-editor.org/rfc/rfc4949.txt.

[14] Daniel Stirnimann. Gain enterprise admin privileges in 5 minutes. https://www.hacking-lab.com/misc/downloads/event_2010/daniel_stirnimann_pass_the_hash_attack.pdf, 1997.

[15] M. Weir, S. Aggarwal, B. d. Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *2009 30th IEEE Symposium on Security and Privacy*, pages 391–405, May 2009.

[16] wiki.skullsecurity.org. Rockyou password dump. http://downloads.skullsecurity.org/passwords/rockyou-withcount.txt.bz2, 2009.

[17] Jordan Wright. Information dump monitor. https://github.com/jordan-wright/dumpmon.

[18] Yinqian Zhang, Fabian Monrose, and Michael K. Reiter. The security of modern password expiration: An algorithmic framework and empirical analysis. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS '10, pages 176–186, New York, NY, USA, 2010. ACM.

List of Figures

List of Tables

Abbreviations

NLP Natural Languge Processing

EBNF Extended Backus–Naur Form

CNF Chomsky normal form

CFG Context-free grammar

PCFG Probabilistic context free grammar

DAWG directed acyclic word graph

IaC Infrastructure as Code

IaaS Infrastructure as a Service

A Appendix

Table 4: The length of the passwords rockyou

character length	percent of passwords	number of passwords
8	20%	2966037
7	17%	2506271
9	15%	2191039
10	14%	2013695
6	13%	1947798
11	06%	866035
12	03%	555350
13	02%	364174
5	01%	259169
14	01%	248527
15	01%	161213

Table 5: Simple Masks rockyou

charceter class	percent of passwords	number of passwords
stringdigit	37%	5339556
string	28%	4115314
digit	16%	2346744
digitstring	04%	663951
othermask	04%	576324
stringdigitstring	03%	450742
stringspecialstring	01%	204441
stringspecialdigit	01%	167816
stringspecial	01%	148328

Table 6: Advanced Masks rockyou

mask	percent of passwords	number of passwords	keyspace
?l?l?l?l?l?l?l?l	04%	687991	208827064576
?l?l?l?l?l?l	04%	601152	308915776
?l?l?l?l?l?l?l	04%	585013	8031810176
?l?l?l?l?l?l?l?l?l	03%	516830	5429503678976
?d?d?d?d?d?d?d	03%	487429	10000000
?d?d?d?d?d?d?d?d?d?d	03%	478196	10000000000
?d?d?d?d?d?d?d?d	02%	428296	100000000
?l?l?l?l?l?l?d?d	02%	420318	30891577600
?l?l?l?l?l?l?l?l?l?l	02%	416939	141167095653376
?d?d?d?d?d?d	02%	390529	1000000
?d?d?d?d?d?d?d?d?d	02%	307532	1000000000
?l?l?l?l?l?d?d	02%	292306	1188137600
?l?l?l?l?l?l?l?d?d	01%	273624	803181017600
?l?l?l?l?l?l?l?l?l?l?l	01%	267733	3670344486987776
?l?l?l?l?d?d?d?d	01%	235360	4569760000
?l?l?l?l?d?d	01%	215074	45697600
?l?l?l?l?l?l?l?l?d?d	01%	213109	20882706457600
?l?l?l?l?l?l?d	01%	193097	3089157760
?l?l?l?l?l?l?l?d	01%	189847	80318101760
?l?l?l?l?l?l?l?l?l?l?l?l	01%	189355	95428956661682176
?l?l?l?d?d?d?d	01%	178304	175760000
?l?l?l?l?l?d?d?d?d	01%	173559	118813760000
?l?l?l?l?l?l?d?d?d?d	01%	160592	3089157760000
?l?l?l?l?l?l?l?l?d	01%	160054	2088270645760
?l?l?l?l?l?d?d?d	01%	152400	11881376000

B Appendix

Table 7: The length of the passwords 10million

charcater length	percent of passwords	number of passwords
8	29%	2980863
6	25%	2543976
7	16%	1662849
9	06%	680815
5	04%	494992
10	04%	471289
4	03%	345137
11	02%	263466
12	01%	190997
13	01%	135587

Table 8: Simple Masks 10million

charceter class	percent of passwords	number of passwords
string	41%	4185383
stringdigit	21%	2173421
digit	20%	2035160
digitstring	05%	549645
othermask	05%	507869
stringdigitstring	03%	363760
digitstringdigit	01%	107776

Table 9: Advanced Masks 10million

mask	percent of passwords	number of passwords	keyspace
?l?l?l?l?l?l	11%	1175666	308915776
?l?l?l?l?l?l?l?l	08%	897374	208827064576
?l?l?l?l?l?l?l	07%	749491	8031810176
?d?d?d?d?d?d?d?d	07%	739507	100000000
?d?d?d?d?d?d	07%	702944	1000000
?l?l?l?l?l	03%	314438	11881376
?d?d?d?d?d?d?d	02%	208532	10000000
?l?l?l?l?l?l?d?d	01%	199591	30891577600
?l?l?l?l?l?l?l?l?l	01%	196953	5429503678976
?l?l?l?l	01%	173532	456976
?d?d?d?d	01%	142092	10000
?l?l?l?l?l?l?l?d	01%	138451	80318101760
?l?l?l?l?l?d	01%	124788	118813760
?l?l?l?l?l?l?l?l?l?l	01%	120658	141167095653376
?l?l?l?l?l?d?d	01%	119001	1188137600
?l?l?l?l?d?d?d?d	01%	110167	4569760000
?l?l?l?l?l?l?d	01%	107369	3089157760
?l?l?l?l?d?d	01%	104593	45697600