An Investigation of Semantic Patterns in Passwords

Rafael Veras Guimarães

Faculty of Science

University of Ontario Institute of Technology

A thesis submitted for the degree of

M.Sc. in Computer Science

2013

Abstract

The advent of large password leaks in recent years has exposed the security problems of passwords and enabled deeper empirical investigation of password patterns. Researchers have only touched the surface of patterns in password creation, having characterized patterns in terms of frequency, length, composition rules and, to some extent, syntactic patterns. The semantics of passwords remain largely unexplored. In this thesis, we aim to fill this gap by employing Natural Language Processing techniques to extract and leverage understanding of semantic patterns in passwords. We present the first framework for segmentation, semantic classification and semantic generalization of passwords and a model that captures the semantic essence of password samples. The results of our investigation demonstrate that the knowledge captured by our model can be used to crack more passwords than the state-of-the-art approach. In experiments limited to 3 billion guesses, our approach can guess 67% more passwords from the LinkedIn leak and 32% more passwords from the MySpace leak. Furthermore, we explore the implications of using date patterns in guessing attacks and investigate the lexical differences between standard English and the language used in passwords.

Contents

List of Figures

Chapter 1

Introduction

The exact date the password was invented is unknown. Since remote times it has been used by people as a method of authentication, not rarely in military contexts, such as by the Roman military [Paton et al., 2012] and in the Invasion of Normandy by the U.S. 101st Airborne Division during World War II [Bando, 2007]. In computing, it was introduced in the early 1960's, and has become both omnipresent, fostered by the advent of Internet, and the target of much controversy, due to its inherent security and usability problems. Nevertheless, passwords are not likely to be replaced in the near future, as alternative, more sophisticated forms of authentication are still immature or economically infeasible, including the ones based on "what you have" (e.g., tokens, cards, etc.) and on "who you are" (biometrics). Moreover, passwords offer advantages not always matched by other schemes, including usability and easy recovery from loss [Bonneau et al., 2012].

Even after half a century of use in computing, we still do not have a deep understanding of how passwords are created. As a consequence, there is no consensus on the real level of security of passwords or on the adequate metric for password strength [Bonneau, 2012]. The fact that during the past few years many security breaches in major websites (e.g., Yahoo, Sony, LinkedIn, etc.) led to the disclosure of passwords of millions of users, and the passwords that were hashed were quickly cracked, has driven researchers to try to fill this lack of understanding. These lists provide the largest samples of real-world passwords to date, offering an enormous opportunity for empirically grounded research.

It is been increasingly acknowledged that the key to solving the security problems of passwords lies on a better structural understanding of passwords [Jakobsson and Dhiman, 2013], i.e., the underlying patterns of password creation, but the community's knowledge is still restricted to superficial patterns. The literature features a wealth of investigations of distribution of characters [Castelluccia et al., 2012; Narayanan and Shmatikov, 2005] and types of mangling rules present in passwords [Chou et al., 2013; Weir et al., 2009]. Metrics of password strength consider mainly length, presence of non-alphabetic characters and character casing [Shay et al., 2010]; however, deeper patterns, in particular the ones concerning the *meaning* of passwords, remain largely unexplored.

A search of the RockYou leak, for example, reveals interesting facts about the semantics of passwords: while the most frequent passwords containing the substring *bad* predominantly contain words referring to people (e.g., *badboy*, *badgirl* and *badman*), the most frequent passwords containing *good* cooccur with a much more diverse set of semantic categories (e.g., *lifeisgood*, *goodluck* and *godisgood*). This thesis aims to address the following questions: Are there systematic preferences in the choice of concepts? If so, what is their impact on security? For example, can an attacker save time by targeting a specific semantic category, or targeting a specific sequence of them? It might be also relevant to understand the relationships between semantic categories, e.g., given a password starting with the words "I love", is it more likely to be followed by a male or female name? Another interesting object of study would be the occurrence of semantic patterns across populations of different language.

Historically, the semantics of passwords have been investigated through research instruments of social sciences, such as surveys, with small groups of participants [Brown et al., 2004; Riddle et al., 1989]. Although presenting some interesting findings, those studies lack ecological validity, as passwords are collected in controlled experiments, and direct applicability against security problems, as the evaluation is qualitative.

In this dissertation we explore the large list of passwords (over 32 million) stolen and made publicly available in 2009 from the RockYou website[1]. Our first contribution is a in-depth investigation of the date patterns in the list examined. Second, we demonstrate how Natural Language Processing (NLP) algorithms can

[1]http://rockyou.com/

be used to segment, classify and generalize semantic patterns from passwords. Our third contribution is a model that captures structural, syntactic and semantic patterns of a list of passwords. We build upon previous work on Probabilistic Context-Free Grammars to train a grammar composed of part-of-speech (POS) and semantic nonterminal symbols. Finally, the fourth contribution consists in testing the security impact of semantic patterns. We use our grammar to generate guesses in off-line attack scenarios against other leaked password lists (LinkedIn and MySpace). The results show that, in sessions bounded to 3 billion guesses, our model can guess approximately 67% more LinkedIn passwords and 32% more MySpace passwords than the state-of-the-art approach, proposed by Weir et al. [2009]. Furthermore, our approach can guess ultimately 30% more MySpace passwords than the aforementioned approach.

The thesis is structured as following: in Chapter 2 we summarize the literature on password patterns; in Chapter 3 we present an investigation of the date patterns in RockYou; in Chapter 4, we present an approach for segmenting passwords, classifying password segments by POS and semantic category and abstracting semantic categories; in Chapter 5 we build a Probabilistic Context-Free Grammar based on semantic and syntactic tags and present experimental results; finally, in Chapter 6, we review our contribution and discuss limitations and future work.

1. Introduction

Chapter 2

Related Work

Research in the field of psychology has employed qualitative research instruments to investigate the semantics of passwords. Brown et al. [2004] found through surveys that the most frequent entity in passwords authored by college students is the self, followed by family, lovers and friends; also, names were found to be the most common information used, followed by dates. Similarly, Riddle et al. [1989] found that birth dates, personal names, nicknames and celebrity names are common. However, eliciting the meaning of passwords from users may be a limited method. It is unlikely that people disclose the true theme of their passwords if it is embarrassing for them; for example, we have found that many passwords contain sexual references and profanity. Moreover, although interesting from the human point of view, the outcomes of these studies are not strong enough to inform security guidelines or proactive password checking [Bishop and Klein, 1995].

Researchers in the field of computer security have recently began breaking passwords into components and characterizing their structural patterns to develop more empirically grounded strength metrics. In general, the recent literature about passwords has focused on demonstrating that the traditional metrics of password strength, such as entropy, do not provide accurate measures in the face of real-world attacks. Several researchers have proposed methods that expose the vulnerability of the current password creation policies due to high-level patterns, including lexical (i.e., word preferences), structural (i.e., preferences in composition rules) and, to some extent, syntactic patterns (e.g., noun-verb sequences).

Weir et al. [2009] proposed a method to learn structural patterns from a password list using probabilistic context-free grammars (PCFGs) and an algorithm to generate guesses in highest probability order, which was able to crack 28% to 129% more passwords than John the Ripper, a popular password cracker, in scenarios with fixed number of guesses. Their cracking strategy has been considered the state-of-the-art technique. The main limitation of their approach is not being able to assign realistic probabilities to alphabetic words, nor capture their relationships. Nevertheless, the PCFG framework is of general applicability to learning password patterns, and has been applied in contexts beyond structural patterns [Chou et al., 2013; Rao et al., 2013]. In a follow-up paper, by performing standard password cracking attacks against real passwords, the authors devised an empirical assessment of the security provided by different creation policies and evidenced the inadequacy of the notion of entropy as a metric of password strength [Weir et al., 2010]. Bonneau [Bonneau, 2012] proposes new metrics based on guessing resistance for password strength.

Jakobsson and Dhiman [2013] propose a parser and a model for scoring password strength. Their algorithm takes a list of decomposed passwords from the parser and learns the component frequencies (including alphabetic strings, as opposed to the algorithm of Weir et al. [2009]), which are used to estimate the probability and, thus, score the strength of a password. Their approach, however, is still limited in capturing structural patterns, e.g., it makes no distinction between *password1* and *1password*. Also, it does not account for complex relationships between classes; for example, is the sequence “Ilove” most likely to be followed by a male or female name, a determiner or a noun?

A few publications have gone a step further, assuming that password creation might be influenced by *syntactic* rules, characterized syntactic patterns and lexical dependencies. Ur et al. [2013] present a study comparing the RockYou and Yahoo! leaks with several password lists obtained from participants in controlled experiments exploring varied creation policies. They performed segmentation and POS tagging of passwords and compared the distribution of POS tags between the password and natural language, concluding that passwords are more likely than English to contain nouns and adjectives, but less likely to contain verbs and adverbs. The authors also computed statistics on the presence of bigrams from the Google Web Corpus for each list, showing that knowing one piece of a password improves the probability of guessing the whole password. Finally, using a measure of corpus lexical similarity, the authors suggest that RockYou and Yahoo! are relatively similar. This relates to the finding of Bonneau [2012] suggesting that the strength of Yahoo! passwords is similar to the RockYou passwords.

More substantially, Rao et al. [2013] study the effect of grammar on vulnerability of long passwords and passphrases. Through a series of experiments, they investigate the reduction in search space resulting from following English grammar, concluding that guessing effort is not a direct function of password length, but also the syntactic structure (how many words are used and what are their POS). Some POS tags are more vulnerable than others since they can generate a smaller number of guesses (e.g., the search space of nouns is much larger than of pronouns). While not discussed in their paper, it is clear that the presence of semantic patterns could reduce even further the search space of passwords. The findings of Bonneau and Shutova [2012] suggest that the choice of people's passphrases is highly influenced by their probabilities in natural language, which has a very skewed distribution, favouring guessing attacks. In particular, they found that users strongly prefer simple noun bigrams that are common in natural language.

The above studies, however, are limited in that they assume the vulnerabilities are mainly a consequence of users choosing patterns common in natural language, represented in reference corpora, such as the British National Corpus and the Google Web Corpus. In this thesis, we present a model which, *independent* of passwords following natural language patterns, is capable of capturing their semantic and syntactic essence. In this way, we show that even if passwords do not follow the same patterns of natural language, if one is able to learn the patterns, they can be compromised.

In summary, the aforementioned studies inform extensively how mangling (composition) rules are used and their impact on security; in addition, a few studies have suggested that syntactic patterns might reduce the security of passphrases, and suggested the same of common passwords, which are used in the majority of systems. These works inspired our investigation into the role that non-uniform distributions of semantic categories, and the dependencies between them, may have on the security of passwords, and no previous work has investigated this semantic aspect to date.

Chapter 3

The Role of Dates

Recent findings indicate that numbers appear to be commonly used in passwords across language groups, nations, and other population groups [Bonneau, 2012]. For a cracker, a guessing attack based on number patterns is a straightforward way to crack a significant number of passwords, as it would not require dictionaries tailored to the target. In semantic terms, date is the most prominent concept encoded in numerical sequences. As we shall see in this chapter, patterns related to the choice of dates represent a significant vulnerability.

3.1 Processing

Passwords come in a wide variety of forms. Since our main goal is to characterize the occurrence of dates, we need to determine what will be considered as such. The everyday use of dates is supported by some important conventions and symbols meant to avoid ambiguity when a compact format is convenient. For example, separators (e.g., '/', '-', '.') are normally used to delimit the elements of a date (year, month, and day); however, perhaps due to historical constraints in some password systems, password creation rules, and factors such as usability, memorability, and even portability—it is easier to re-use them as PINs—, people tend to avoid special characters in passwords.

Not less important, the order of the elements also helps to resolve ambiguity. Notably, the way people use ordering varies deeply across countries, and is a cause of confusion even within a single country, as is the case of Canada, where both DD/MM and MM/DD formats are used. Since we do not know the country where a password was issued, deciding between formats is challenging. Furthermore, the presence of leading zeros is also a source of variation and ambiguity. Even considering the separators, the date 01/02/99 can be parsed as February 1, 1999 or January 2, 1999. If we remove the separators and the leading zero (10299), the date February 10, 1999 is also introduced as a possibility.

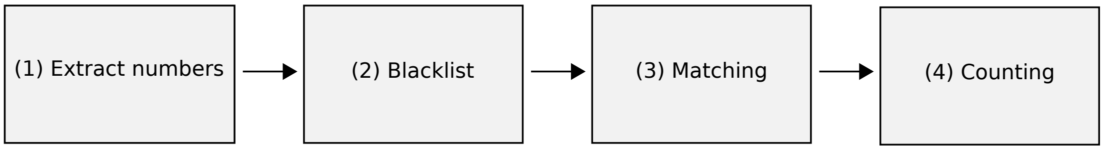

Figure 3.1: Parsing

With the aforementioned considerations, Figure 3.1 illustrates the steps of parsing, which are applied to all passwords that contain a sequence of 5 to 8 digits. Passwords containing sequences of less than 5 digits are discarded, even though a date can be represented by 4 digits; we do this because we are only seeking dates which are fully specified with day, month, and year. The first step is extract the numerical sequence from the password. After that, the most common numerical sequences are 12345, 111111, 123123, 121212 and 112233, which, intuitively, seem not to represent dates, but "pure" numerical/keyboard patterns (see Table 3.1). In (2), we remove all sequences that match any of the numerical patterns and some other highly frequent sequences not captured by the patterns.

Pattern	Examples
Repeated digits	123123, 112233, 111222
Progression	12345, 02468, 654321
Palindrome	45754, 33633, 045540

Table 3.1: List of numerical patterns

In the next step (3), the sequences are tested against a comprehensive list of date formats (Table 3.2). This list captures a broad range of formats of 5–8 digits without special characters, including variations in use of leading zero. A valid date should match at least one of them and lie between the year range [1900, 2012].

8 digits	7 digits	6 digits	5 digits
ddmmyyyy	ddmyyyy	ddmmyy	ddmyy
mmddyyyy	mddyyyy	mmddyy	mddyy
yyyymmdd	dmmyyyy	dmyyyy	dmmyy
yyyyddmm	mdyyyy	mdyyyy	mmdyy
	yyyyddm	yyyymd	
	yyyymdd		
	yyyymmd		

Table 3.2: List of date formats.

A single password can match several formats, that might translate into different or repeated dates (e.g., 030475 → mmddyy and mmddyy → April 3 and March 4, 1975). We considered different approaches for dealing with this ambiguity when building the frequency distribution of dates (4). Counting all derived dates as independent events was discarded because it would overrate ambiguous dates. Counting just the first match based on a priority list of formats turned out to be impractical since we don't have solid basis on which to prioritize them. Hence, the most reasonable strategy is to divide the count of a single event between all matched dates. In the aforementioned case, for instance, both dates would receive an increase of 0.5 in their frequency value.

3.1.1 Testing the Dates Assumption

We performed an experiment to rule out that the matched date sequences in the observed data (RockYou list) could be observed by chance.

The experiment was divided in four parts, each corresponding to one of the sequence lengths considered. For each length, we randomly generated a list containing as many numerical sequences as found in the RockYou dataset. We then ran the parsing algorithm over both samples, counting the event of a success (when a sequence is matched by at least one format). Finally, a Pearson's Chi-squared Test was performed to compare the results. The proportion of sequences that contain dates found in the random list corresponds to our expected value. The results show

Subset Description	# of Passwords	% of RY Passwords
(1) Passwords containing sequences of at least 4 digits	8,056,329	24.72
(2) Passwords from (1) that match a numerical pattern	1,346,410	4.13
(3) Passwords *containing* 5–8 consecutive digits	4,974,602	15.26
(4) Passwords from (3) that match a date	1,934,821	5.93
(5) Passwords *consisting* of 5–8 consecutive digits	3,951,852	12.13
(6) Passwords from (5) that match a date	1,469,662	4.51
(7) Passwords from (6) that match a numerical pattern	114,724	0.35
(8) Passwords that contain a date and at least one alphabetic character	358,562	1.10

Table 3.3: Table of statistics of how numbers and dates appear in the RockYou (RY) list.

that for all considered lengths, the number of dates found in the RockYou dataset is significantly higher than in the random dataset ($p < 2.2 \times 10^{-16}$). While this test does not prove that numeric passwords which match date patterns are intended to be dates, it does present intriguing evidence that the passwords may indeed represent dates.

3.1.2 Basic Statistics

Of the 32 million passwords present in the RockYou list, approximately 25% contains a sequence of 4 or more digits. Of these sequences of at least 4 digits, approximately 62% contain 5 to 8 digits (which can represent a full date consisting of a month, day, and year).

Table 3.3 summarizes some interesting statistics on this password list. When we match the sequences of 5–8 digits against our date patterns, we notice that they can explain 38% of such sequences. Dates appear to be more popular in sequences that are completely composed of digits: of the sequences that contain a date pattern, 75% are entirely numerical digits. Of all passwords that are solely composed of digits, 37% match date patterns (or 34% when we remove the ones that may be due to a numerical pattern).

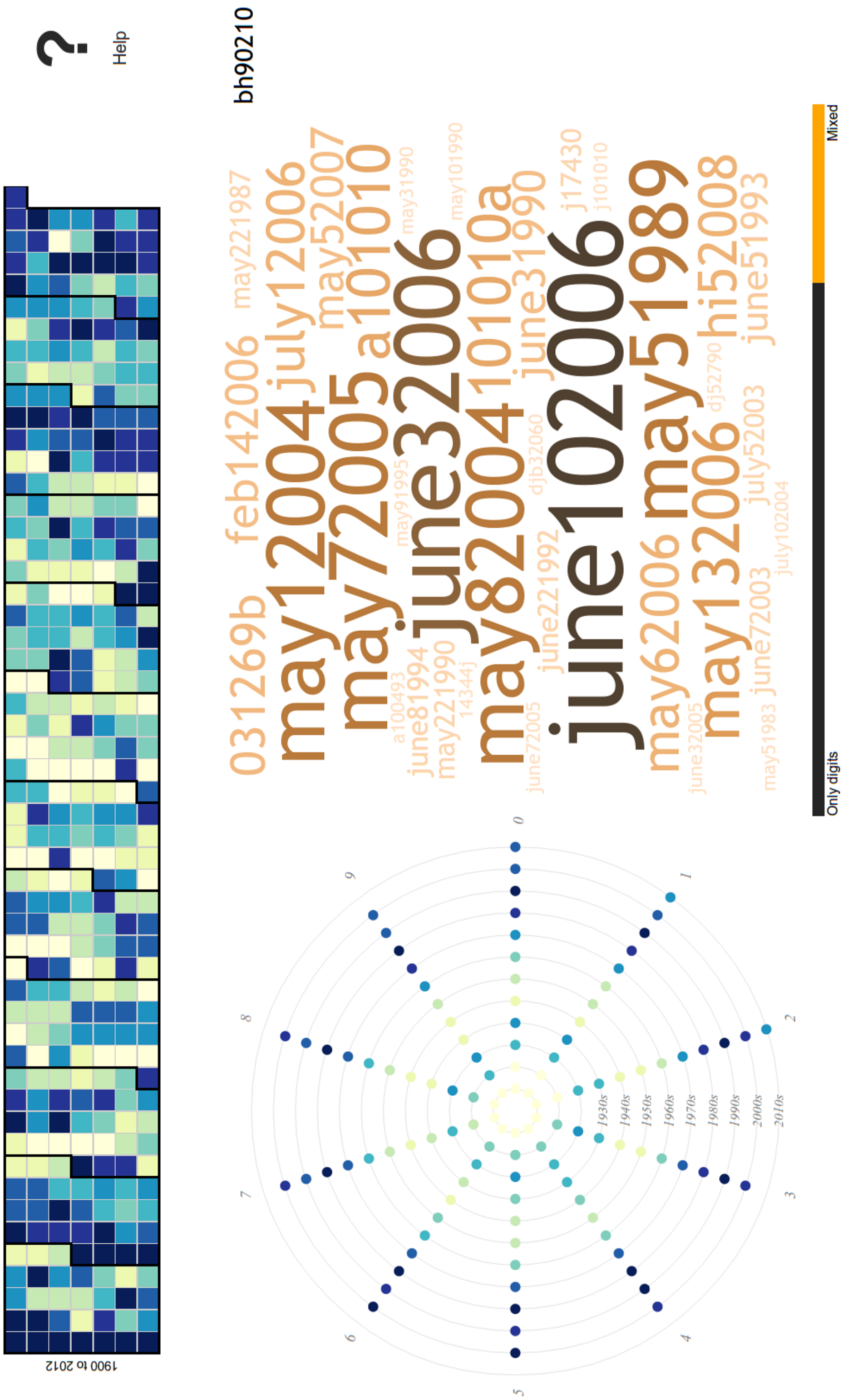

Figure 3.2: Layout overview

3.2 Visualization

To approach the problem of verifying whether dates really do play a significant role in passwords, and if so, discovering whether there are patterns of dates, or specific dates which stand out, we designed an interactive visualization to explore the dataset[1]. We took a multiple coordinated views approach in order to provide several ways to look at the data (see Figure 3.2). The main goals which guided our design are:

Guide the investigation Drawing sound security recommendations from patterns observed in a dataset eventually requires rigorous statistical treatment; however, data manipulation at a low level is cumbersome and does not favour the exploration of data space necessary in the early stages of an investigation. The role of the visualization in this context is to support quick generation and early testing of hypotheses. It should enable insight on possible patterns and provide quantitative information to help deciding whether or not a statistical experiment is worthy. Thus, the formal procedures are left for validation in the final phase of the investigation.

Facilitate exploration of diverse scenarios The tool should enable one to easily delimit scenarios for investigation of localized patterns. This involves the ability to narrow the scope based on time dimension (e.g., decades, years, days...) and password structure (e.g., presence of a numerical pattern or letters).

Easily accessible We took a rapid-prototyping approach, refining the visualization to respond to the questions raised by every new hypothesis drawn, reflecting our increasing understanding of the data. As a consequence we needed a medium that provides easy and fast deployment of new versions and high accessibility to a distributed team.

[1] http://vialab.science.uoit.ca/pwdates/

3.2.1 Representation and Interaction Design

The layout with coordinated views displays the frequency of passwords at multiple aggregation levels (decades, years, months, and days). To provide the analyst with confidence in our parsing algorithm, and to make use of the human ability to see patterns, we also provide a view of the raw passwords. There are three main components of the view. The Radial Plot shows the distribution of dates parsed from passwords along years and decades, the Tile Map depicts the distribution of passwords across days and months, while the raw passwords are shown in a Wordle view. Performing filtering in a high-level view, such as the Radial Plot, narrows the context of the lower level ones, in a top-down fashion; conversely, removing elements from the low level views triggers updates in the high level ones. Despite the huge amount of data, we strive for fluidity to support perception of changes resulting from transition between states. The next subsections describe each component.

3.2.1.1 Radial plot

This view represents years through circles positioned in a radial layout (see Figure 3.2, bottom left). All years of a certain decade are evenly distributed along a ring, in clockwise order. The rings, representing decades, are organized in ascending order from center to periphery. Each spoke represents years ending in a particular digit. The frequency of passwords in a given year is encoded by color, according to a quantile scale that maps the frequency values to the range [0,9], corresponding to the colors of a sequential multi-hue pallette published by Brewer. This scale is meant to reduce the negative visual effect produced by outliers, which occurs with a linear color scale.

The radial view enables observation of cyclical patterns, while also giving us a sense of the linear growth of frequency over the decades; furthermore, it enables rich interaction through selection of rings, circles and labels. The most common cyclical representation is, however, the spiral [Carlis and Konstan, 1998; Tominski, 1999]. We choose instead the ring-based configuration because it allows selection of rings (aggregation by decade), which is an important task in this context.

The default state corresponds to the overview, where the whole dataset is shown in all views, and can be reached by clicking on a blank space in the Radial Plot. Selecting a year by clicking it updates the Tile Map to show the corresponding frequency distribution across days of that year, and the Wordle is filled with the corresponding passwords. In the same way, it is possible to aggregate the years by decade by selecting a ring. Cross-decade aggregation is supported by clicking on an external label at the end of a spoke, e.g., clicking '2' would select the years 1902, 1912, 1922 and so forth.

3.2.1.2 Tile Map

The Tile Map (see Figure 3.2, top) uses a calendar layout to display the frequencies computed for each day in a particular year [Mintz and Wayland, 1997]. The color encoding is consistent with the Radial Plot; that is, frequent regions are evidenced by dark tiles. A click on a tile triggers an update in the Wordle, which will show the raw passwords associated with the selected day. We extend the original use of Tile Maps by plotting aggregated values from multiple years, much like as though several maps were stacked. When used in this way, the calendar nature of the visualization loses its meaning, so we discard the labels informing the days of week (Monday, Tuesday, etc.). Although simultaneous display of multiple Tile Maps in a vertical list eases comparison between years [Wicklin and Allison, 2009], aggregating them in a single unit allows better perception of patterns *accumulated* over a period of time.

3.2.1.3 Word cloud

This visualization builds on the idea of a Wordle diagram, a tightly packed version of a word cloud [Viégas et al., 2009] (Figure 3.2, bottom right). The view is populated with raw passwords which match the selected years (Radial Plot) and day, if any (Tile Map). The passwords are sized according to the number of times they occur in the underlying dataset. An indicator bar is used to show the proportion of matched passwords which are purely numerical compared to those which contain a date-like numeric sequence as well as words and other symbols. This bar is interactive and can be used to restrict the view to the corresponding subset by clicking the corresponding bar.

In order to allow a researcher to remove any passwords which are strong outliers, and to see patterns in the remaining data, we provide the ability to select and remove a password from the Wordle. The filtered word goes to a 'filtered' panel on the right side, then the Wordle is recomputed. When the computation is done, an animation smoothly reorganizes the passwords.

Since it can be difficult to keep track of what has changed when a new layout is calculated (e.g., which passwords got more or less importance after a filter is adjusted), the duration of the transition is proportional to the frequency of the password. So, more frequent (bigger) passwords move slower. While we have not tested this, we feel that this appearance of the larger passwords moving more slowly helps to give stability to the view during the relayout process.

3.2.2 Implementation

The tool is a web-based application that runs entirely in the browser, is written in JavaScript, and built on top of a set of web technologies standardized by W3C; namely, HTML, CSS and SVG. In addition, we use the D3 library [Bostock et al., 2011] to manipulate data and the page's elements, to control animation, map data values to visual attributes and deal with events.

3.3 Semantic Patterns Discovered

When using our date visualization tool, we noticed a number of interesting patterns in user choice (Figure 3.2). To summarize, there appears to be a preference for the following:

- Years after 1969. The popularity of a year is indicated by the darkness of the color in the radial portion of the visualization. See Section 3.3.1 for further details.

- Text words that spell out the name of a month (e.g., "May12009"); see Section 3.3.2.

- Two years immediately after one another (e.g., "20082008" or "19391945").

- The first two days in each month (e.g., “010989”).
- Repeated months/days (e.g., “August 08”).
- Holidays (e.g., Valentine’s day, Christmas day, and New Year’s day); see Section 3.3.3.

We use each of these observations to specify patterns, which we use to compile a dictionary used to analyse security implications (discussed in Section 3.4). We investigate these patterns further in the following subsections.

3.3.1 Recent Years

The radial plot indicates that recent years, in particular after 1969, are the most popular. Years in the 1980’s, followed by 1990’s and then the 2000’s appear to be the most popular. There are still a fair number in the 1970’s and 2010’s, and the popularity noticeably drops after 1969. We investigated this effect further and found that 1,160,801 (86% of purely numeric date passwords) represent dates after 1969. Some possible reasons for this preference are that the dates correspond with: (1) the birthdays of people using these accounts, (2) the dates of significant events for the people using these accounts, and (3) the dates that people created these accounts.

3.3.2 Text Combined with Dates

Using the Wordle portion of the visualization, we examined the most popular text strings that co-occur with dates. We observed that single-characters and initials appear the most frequently, and when full words are used, they are often the months of the year. This motivated us to examine how many passwords match date patterns, where the month is spelled out as opposed to being in a purely numerical format. We generated a set of formats for such dates, for example, MonthDDYY (see all formats in table 3.2). In all cases where the day is a single digit, we assume no leading zero is present. Our results are shown in Table 3.4.

We found these numbers to be quite surprising, given that dates written in this format are rather specific. Table 3.5 combines this result with the pure number

Years considered	# of passwords	% of **all** passwords
1900-2012	124,460	0.38
1969-2012	117,436	0.36

Table 3.4: Passwords in the RockYou dataset that are in a mixed characters and digits representation of a date (e.g., “1May1990”).

results that are dates, showing that, in the RockYou leak, nearly 5% of users *choose a date as their password*, and nearly 4% of users *choose a date on or after 1969 as their password*. As indicated in Table 3.3, the number should be even higher when considering users who choose dates as *part* of their passwords.

Years considered	# of passwords	% of all passwords
1900-2012	1479398	4.54
1969-2012	1278237	3.92

Table 3.5: Passwords in the RockYou dataset that match a date pattern (e.g., “1May1990” or “01051990”). Note that dates which can also be considered a numerical pattern (e.g., “112233”) are not included in this result.

3.3.3 Holidays

Through exploring using our visualization, we discovered that some familiar dates “pop out”, which correlate with holidays such as Valentine’s Day, New Year’s Day, New Year’s Eve, and Christmas Day (see Figure 3.2). While exploring the decades individually, we also noticed a number of other noteworthy dates appearing more frequently than expected, including:

- March 21 (First day of spring; Persian new year)
- December 12, 2012 (date associated with the “2012 cataclysm phenomenon”)
- August 17, 1945 (Indonesian Independence Day)
- April 14 and 15, 1912 (when the Titanic sank)

3.4 Security Implications

Our observations using our visualization tool provide deeper understanding of user choice relating to the semantic category of dates. It provides information regarding how an attacker might perform an offline attack against a system in which he or she has no knowledge of the users, their spoken languages, and the dates they might choose (e.g., does *not* know the user's birthday). Our analysis can also inform password policies and guidelines.

3.4.1 Date-based Guessing Attacks

Here we focus on purely numeric passwords, showing the results of building a dictionary based on each of the patterns discussed in Section 3.3. Our results are provided in Table 3.6. Of particular interest are the bolded values in the last two rows. In the second last row ("combined"), we see that by creating a dictionary which combines all of our visualization-observed patterns, we would be able to guess over 27% of date-based passwords using a dictionary composed of only approximately 15% of the possible dates. The final row shows that we can guess over 22% of date-based passwords using a dictionary composed of only approximately 7% of the possible dates.

Our findings approximate the extent to which these patterns dominate user choices of dates. The breakdown of each individual sub-dictionary, and the combined dictionary (with duplicates removed) is provided in Table 3.6.

Dictionary (1900–2012, unless otherwise specified)	dictionary size	% of full dictionary	# passwords guessed	% of all date passwords	% of all RockYou passwords
(1) All days	**206658**	100.00	1354938	100.00	**4.16**
(2) Valentine's day	752	0.36	6020	0.44	0.02
(3) Christmas day	426	0.21	5675	0.42	0.02
(4) New Year's Eve	426	0.21	4562	0.34	0.01
(5) New Year's Day	539	0.26	9835	0.73	0.03
(6) First days of every month	11193	5.41	105493	7.79	0.32
(7) All days in December	15501	7.50	94957	7.01	0.29
(8) Repeated days/months	5490	2.66	71709	5.29	0.22
(9) Repeated days/months/years	81	0.04	16058	1.19	0.05
(10) Year1Year2	12769	6.21	29976	–	0.09
(11) Repeated years	113	0.05	10490	–	0.03
(2–11) Combined	**31856**	15.49	372640	**27.50**	**1.14**
(2–11) Combined (only 1969–2012)	**14914**	7.26	303334	**22.39**	**0.93**

Table 3.6: Passwords in the RockYou dataset that were guessed by dictionaries representing each of the patterns that we found in our visualization.

Table 3.6 shows that these patterns correctly capture approximately 27% of date passwords, which corresponds to approximately 1% of all RockYou passwords. We emphasize that we have eliminated our identified numerical patterns (e.g., “121212”) from these results, and that by combining raw numerical patterns with this dictionary, even more passwords could be guessed; however our purpose in the present paper is to quantify the effect of popularly-chosen dates. The results of the combined dictionary show that we could guess nearly 1% of all RockYou passwords in approximately 15,000 guesses defined by “popular-looking” dates.

Given that this dictionary uses only purely numerical passwords, it could model an attack under the following threat model — when an attacker only wishes to obtain access to a single account, account-lockouts are not implemented (or the attack is offline), and the attacker knows nothing about the target user group (e.g., language, birth dates, etc.). Of course, numerical patterns appear to be more popular and would pose more of a threat, but on some systems such obvious passwords are blacklisted.

3.4.2 Password Policies and Guidelines

We use the presented visualization to gain further understanding of how people choose dates in passwords. The date subset appears worthy of investigation as it is apparently a common semantic category within user choice; nearly 5% of all user passwords in the RockYou dataset can be considered a pure date. A dictionary that would be able to guess all of these pure dates would consist of approximately $508,492$ entries, which is feasible to guess in a short amount of time in an offline attack. This alone creates patterns that are easy for attackers to guess, implying that it would be prudent to recommend that users do not choose a pure date as their password, even when it adheres to all other password rules (e.g., “May1/2009” would satisfy common password requirements, but likely should be disallowed).

Our findings also strongly suggest the presence of certain patterns in user choice of dates. These patterns tell us something about user preferences, which provide further insight into the password selection process. For example, users seem to prefer dates that fall on the first day of the month, or are a partial repetition. This raises a question of whether users might prefer passwords that can be characterized by multiple patterns? It also raises the question of whether certain numbers are more memorable than others? If either is so, this could have implications for creating better password guidelines to aid users in choosing a more secure yet memorable password.

Chapter 4

Parsing and Classification

After having analysed the date patterns in the subset of numerical sequences, in this chapter we begin describing a more general approach for extraction of semantic patterns in passwords. We apply NLP methods to the segmentation and classification of password samples. With such methods, we decompose passwords into conceptually consistent parts and infer their meaning and syntactic function. The computer-supported semantic classification of passwords is an unprecedented application of NLP. In this approach, passwords of all forms and lengths are broken into parts and classified semantically; thus, segmentation is a fundamental step. Segmentation of passwords is at least as difficult as URL segmentation, because methods cannot rely on presence of space delimiters between words. This means that a good method to resolve ambiguities is critical.

4.1 Segmentation

Extensive research has been done to address the problems of segmentation of texts written in Asian languages, whose writing systems do not feature a white space delimiter and URL word breaking. Passwords are similar to URLs in that both are fairly multilingual and can include numbers and special characters (passwords allow more variation). URLs, however, have much more context information available, i.e., the documents they point to, including body text, title and other metadata. The methods for URL word breaking documented in literature have varying

Corpus	Size
COCA unigrams	497,186
COCA bigrams	1,020,138
COCA trigrams	1,020,009
Total	2,537,333

Table 4.1: Reference corpora detailed.

degrees of similarity with our method. Heuristic-based approaches have proposed the resolution of ambiguities in URL segmentation by looking at document contents [Chi et al., 1999] or scoring classes of word differently (e.g., stop words and known lexicon) [Khaitan et al., 2009]. Other heuristics take into consideration word length. Lexicon based, monolingual approaches using N-grams are also popular, as statistical methods are deemed as more robust given the wealth of data available [Monz and Rijke, 2002]. Other approaches employ Bayesian frameworks [Goldwater et al., 2006].

The first application of word breaking in passwords appeared not until recently, by Jakobsson and Dhiman [2013], who proposes a lexicon-based parser. Like in our method, their algorithm takes a compilation of general and specialized dictionaries as input and uses a measure of *coverage* as primary criterion for selection of candidate segmentations. However, it does not make use of context (high order N-gram frequencies) to disambiguate segmentations with equal coverage.

4.1.1 Dictionaries

Our algorithm takes as input a variety of English corpora. We make a distinction between *source corpora* and *reference corpora*. Source corpora consists of a collection of raw word lists that constitute the algorithm's lexicon; it is the base for building the segmentation candidates. The reference corpora is a collection of part-of-speech tagged N-grams with frequency of use information, which are used for selecting the most probable segmentation (Table 4.1). As we later explain, not all words from the source corpora need to appear in the reference corpora; i.e., not all words need to have an associated frequency. This frees us to compile very comprehensive source corpora. Still, while noise in the source corpora is not a

Word list	Original Size	Trimmed Size
COCA	365,748	359,226
Female names	51,929	51,929
Male names	29,651	29,651
Cities	22,737	21,780
Surnames	28,873	28,412
Months	60	60
Countries	260	260
Total	499,258	491,318

Table 4.2: Source corpora detailed.

threat to the quality of the segmentation—our algorithm will always prefer the most probable candidates—, it impacts on the performance of parsing, since more candidates will be generated and evaluated; therefore, trimming of the word lists is convenient.

The main corpus is the Contemporary Corpus of American English, a large, general-purpose corpus containing part-of-speech tagged unigrams, bigrams and trigrams along with the observed frequencies of occurrence in general language (books, magazines, blogs, speeches, etc.)[Davies, 2008-]. COCA is used as our reference corpus and a trimmed version is used as part of the source corpora. In that version, of the words with three characters, the ones with less than 100 occurrences were removed; of the words with two characters, we selected the top 37, as the less frequent words were mostly acronyms; and the only one-character words kept were *a* and *I*. Those subjective thresholds values are the result of observation of the dataset. The goal is to reduce the number of short, rare words, such as some acronyms, that would slow down the parsing without improving accuracy.

The general nature of COCA is insufficient to support semantic classification of named entities at a later step, especially regarding names and locations. For this purpose, we use a collection of specialized word lists, whose size is presented in Table 4.2:

Names Derived from a dataset of the U.S. Social Security Administration (SSA) [SSA]. All names are from Social Security card applications for births that occurred in the United States after 1879 until February 2012. We further divided this list by gender.

Cities Derived from the Geonames [GeoNames] list of cities which have at least 15,000 inhabitants or are capitals. In order to reduce noise, we removed cities whose name contains four characters and population lower than 240,000 (few notorious cities can be found below this threshold), or fewer than four characters.

Surnames As with many popular word lists on the web, the actual source of the list of surnames is unknown. This list was downloaded from Outpost9 [Outpost9] and had the words with fewer than four characters removed.

Months List of months in english.

Countries List with names of all countries in English.

4.1.2 Algorithm

As previously mentioned, word boundaries are not explicit in passwords. Indeed, due to lack of context, it is impossible to determine the exact words, if any, intended by the password's author. This is worsened by the usual intention to make passwords more cryptic, realized in the form of a variety of *mangling* patterns. Mangling patterns (or rules) are used to generate complex variations of a simple password, e.g., *love*, *l0v3*, *3v0l*, *etc*. According to Jakobsson and Dhiman [2013], the most common rules are concatenation, replacement, spelling mistake and insertion. Because mangling rules are a popular creation strategy, any segmentation algorithm tailored to passwords needs to account for mangling. From a security perspective, it is also important to preserve and later classify such patterns.

Example 1. crazy2duck93^ ⟶ gaps: {2, 93^}; words: {crazy, duck}

Let's assume a password is a sequence of *word* and/or *gap segments*. A word segment is any string that can be found in the source corpora, while a gap segment is any string not present in the source corpora surrounded by word segments or password boundaries at any side. Given the constitution of our source corpora, a

Password		Segments				Coverage
Anyonebarks98	(A)	Anyone	barks	98		0.84
	(B)	Any	one	barks	98	0.84
	(C)	Anyone	bar	ks98		0.69
	(D)	Any	one	bar	ks98	0.69

Table 4.3: Candidate segmentation for password *Anyonebarks98*

word segment is always alphabetic, while a gap can include any character (numbers, symbols or letters). Example 1 illustrates the segmentation of a password containing both types of segments.

In Example 1 there is not much room for ambiguity. In Table 4.3, instead, we have at least four competing candidate segmentations. If we favour *coverage* by word segments, i.e., minimum presence of gaps, we can rule out the candidates C and D. The two remaining candidates have equal coverage; thus another criterion is considered as a secondary disambiguation factor: *frequency of use*. In the English language, the construct (A) is more probable than (B).

The segmentation strategy illustrated in Table 4.3 is described at high level in Algorithm 1. Given a password p, we generate a set W containing all substrings of p; then after a filter, W contains only the strings present in the source corpora (word segments). Next, a list of segmentation candidates is built, each containing a subset of W. The segmentation candidates are only formed by word segments. The list is then filtered to contain only the ones with greatest coverage (sum of length of segments). In the frequent case that more than one candidate remains, we assign an n-gram probability to each candidate and select the best (t). As a last step, the gap segments are re-inserted in t in the appropriate positions.

The selection of the most probable segmentation candidate is based on the reference corpora. As previously stated, it contains high order N-gram frequencies that can help us rank the segmentations by likelihood. Let K_N be an N-gram corpus and $f(K_N)$ the total frequency of N-grams in corpus K. The probability of an N-gram $w_1...w_N$ is given by:

$$P(w_1...w_N) = \frac{f(w_1...w_N)}{f(K_N)} \tag{4.1}$$

Algorithm 1 Segment string into most probable word and gap sequence

```
1: procedure SEGMENT(p)
2:     W ← Generate all possible substrings of p
3:     Remove w ∈ W not present in source corpora
4:     C ← Generate segmentation candidates from W
5:     θ ← Calculate maximum coverage from C
6:     Remove c ∈ C | c < θ
7:
8:     if LENGTH(C) > 1 then
9:         t ← Select most probable c ∈ C
10:    else
11:        t ← C[0]
12:    end if
13:    Insert gaps in t
14:    return t
15: end procedure
```

An annotated trigram corpus can serve as the grounds for very accurate segmentation, but its coverage is usually limited. The higher the N-gram order, the greater the chances of a context not be found in the corpus. There is a clear trade-off between accuracy and coverage and one way to work around it is falling back to less accurate algorithms whenever necessary. We rely on this backoff strategy in the recursive Algorithm 2 to generate probabilities used in line 9 of Algorithm 1. The probability of a segmentation is the product of its N-gram probabilities. Given a segmentation containing three segments, for example, the algorithm computes all combinations of trigram, bigram and unigram probabilities and chooses the one that maximizes the score.

In language modelling, N-gram models are often evaluated in the context of a classic task, where the problem is to predict the next word given the previous. We did not test the accuracy of our model in this traditional framework, as we are only interested in ranking the candidates; in other words, we do not need a precise measure of how much better a candidate is in comparison to another. Table 4.4 shows a sample of the segmentation results produced by the algorithm.

Password	Segment
‘daddiesgurl321	‘
‘daddiesgurl321	daddies
‘daddiesgurl321	gurl
‘daddiesgurl321	321
7eleven	7
7eleven	eleven
buddy5	buddy
buddy5	5
chummy	chummy
dm3455	dm3455
futebol	futebol
ilovesabastion	i
ilovesabastion	love
ilovesabastion	sabastion
jon311	jon
jon311	311
lidia00	lidia
lidia00	00
lredix	l
lredix	red
lredix	i
lredix	x
poohbear14	pooh
poohbear14	bear
poohbear14	14
password	password
princess1	princess
princess1	1
rajidevi13	raji
rajidevi13	devi
rajidevi13	13
teamvampire	team
teamvampire	vampire
yellow	yellow

Table 4.4: Random sample of segmentation results.

Rank	Word	Source corpus	Count	Relative Freq. (%)
1	a	coca	1361806	3.24
2	i	coca	1263919	3.01
3	love	coca	584219	1.39
4	me	coca	263629	0.63
5	in	coca	220521	0.53
6	you	coca	206937	0.49
7	baby	coca	204716	0.49
8	my	coca	186373	0.44
9	to	coca	166795	0.40
10	an	coca	159914	0.38
11	is	coca	151409	0.36
12	girl	coca	142228	0.34
13	it	coca	140943	0.34
14	as	coca	119110	0.28
15	la	fem. name	117583	0.28
16	te	fem. name	112123	0.27
17	sexy	coca	109663	0.26
18	on	coca	107114	0.26
19	am	coca	105293	0.25
20	be	coca	100167	0.24
21	man	coca	99677	0.24
22	password	coca	99296	0.24
23	the	coca	98293	0.23
24	luv	coca	98250	0.23
25	boy	coca	92671	0.22
26	no	coca	92572	0.22
27	amo	fem. name	89068	0.21
28	rock	coca	88746	0.21
29	angel	coca	86063	0.20
30	ca	coca	85751	0.20
31	or	coca	82794	0.20
32	na	fem. name	82108	0.20
33	el	male name	80608	0.19
34	and	coca	78502	0.19
35	lil	coca	74801	0.18
36	do	coca	71859	0.17
37	ha	fem. name	71467	0.17
38	de	male name	69206	0.16
39	princess	coca	69178	0.16
40	life	coca	66054	0.16
41	lo	male name	63621	0.15
42	he	coca	62692	0.15
43	ma	fem. name	61648	0.15
44	ko	male name	60691	0.14
45	at	coca	60528	0.14
46	ta	fem. name	60193	0.14
47	fuck	coca	59928	0.14
48	hot	coca	58486	0.14
49	yo	fem. name	58064	0.14
50	pink	coca	57130	0.14

Table 4.5: 50 most frequent words from the source corpora in the RockYou list.

Algorithm 2 Recursively calculate the N-gram score of a segmentation

```
procedure BestNGramScore(C)
    score ← 0
    l ← Length(C)

    if l = 1 then
        score ← UnigramProbability(C)
    else if l = 2 then
        score ← BigramProbability(C)
    else if l = 3 then
        score ← TrigramProbability(C)
    end if

    if score = 0 then
        for i ← 1, 3 do
            a ← BestNGramScore(C[: i])
            b ← BestNGramScore(C[i :])
            tempScore ← a ∗ b
            if tempScore > score then
                score ← tempScore
            end if
        end for
    end if
end procedure
```

4.1.3 Analysis of Segmentation Results

Now that we have the capability of extracting words from passwords, the simplest analytical question one can make is which words are more common in the RockYou list? This question is of relevance to password cracking, in particular, one of the weaknesses of the the approach of Weir et al. [2009] is the lack of a sound method to assign probabilities to the words their guess generator takes as input. In that case, a ranked dictionary can be used to form guesses in highest probability order. Table 4.5 shows the 50 top segments in the RockYou list.

Another relevant question is how the vocabulary of passwords compares to the language of real world? To answer this question, we use the British National Corpus (BNC) [BNC, 2007] as a reference and the measure of corpus similarity G^2 for ranking the most distinguishing words [Rayson and Garside, 2000]. The G^2

measure is calculated using the following contingency tables and equations:

	Corpus A	Corpus B	Total
$C(word)$	a	b	$a+b$
$C(other\ words)$	$c-a$	$d-b$	$c+d-a-b$
Total	c	d	$c+d$

$$E_1 = c*(a+b)/(c+d) \tag{4.2}$$

$$E_2 = d*(a+b)/(c+d) \tag{4.3}$$

$$G^2 = 2*(a*\ln(a/E_1)+b*\ln(b/E_2)) \tag{4.4}$$

where $C(word)$ in the count of the target word and E_1 and E_2 are the expectation values for the word frequency in corpus A and B, respectively. In summary, G^2 tells us the probability that the frequency of occurrence of a word in one corpus differs significantly from another. Table 4.6 shows the most deviant words between passwords and BNC. A positive G^2 value indicates the word is more common in passwords, while a negative value indicates the contrary.

The results reveal that the probability of connective words, in particular, prepositions (from, in, with, to, etc.) and conjunctions (and, for, but, etc.) is much higher in the BNC corpus than in the RockYou passwords. The most reasonable explanation is the size of the sentences, as BNC frequencies are extracted from a large collection of books, newspapers, magazines, and so forth. Surprisingly, a subset of pronouns (*I*, me and *my*) are much more likely to appear in passwords, contrary to others, for example, *her*, *him*, and *they*. Other words stand out in passwords with no obvious linguistic explanation, such as *love*, *baby*, *sexy* and *princess*. Therefore, we hypothesize that, instead of syntactic patterns, semantics should explain the high occurrence of such words and the disparity of frequencies of words with same syntactic function.

4.1.4 Visual Exploration

We designed a web-based visualization tool to enable visual exploration of the lexical difference between BNC and passwords (Figure 4.1). The visualization features

G^2 Ranking	Word	G^2	BNC Freq. (%)	Pass. Freq. (%)
Negative				
1	the	-3562992.1	6.18	0.23
2	of	-1710019.2	2.94	0.10
3	and	-1357509.7	2.68	0.19
4	to	-945327.8	2.56	0.40
5	that	-726880.0	1.11	0.01
6	was	-573790.3	0.92	0.02
7	’s	-529691.4	0.81	0.01
8	for	-482500.2	0.85	0.04
9	in	-445348.4	1.88	0.53
10	with	-425616.0	0.65	0.01
11	have	-303984.0	0.47	0.01
12	they	-296823.3	0.43	0.00
13	from	-273829.2	0.41	0.00
14	but	-271616.7	0.46	0.01
15	this	-270287.3	0.46	0.02
16	had	-266129.0	0.44	0.01
17	which	-259366.4	0.37	0.00
18	his	-257657.5	0.43	0.01
19	not	-237838.7	0.46	0.03
20	it	-231153.4	1.09	0.34
Positive				
1	love	1241949.6	0.02	1.39
2	i	767866.4	0.90	3.01
3	baby	430337.4	0.01	0.49
4	te	269387.4	0.00	0.27
5	sexy	259820.1	0.00	0.26
6	girl	255209.0	0.02	0.34
7	la	249772.7	0.00	0.28
8	luv	237913.5	0.00	0.23
9	password	237306.2	0.00	0.24
10	me	221084.4	0.14	0.63
11	amo	216783.0	0.00	0.21
12	angel	195553.4	0.00	0.20
13	el	183410.1	0.00	0.19
14	lil	179966.7	0.00	0.18
15	rock	172164.9	0.01	0.21
16	boy	155204.1	0.01	0.22
17	lo	150611.6	0.00	0.15
18	ha	147753.1	0.00	0.17
19	ko	147109.6	0.00	0.14
20	princess	146047.7	0.00	0.16

Table 4.6: G^2 Top ranked positive and negative words.

the 500 most distinguishing words (ranked by $|G^2|$) and allows one to interactively compare measures of word frequency and check the most frequent passwords that contain a certain word. The words are represented by polylines in a parallel coordinates plot [Inselberg, 1985], with polyline color encoding the sign of the G^2 value (blue as positive and brown as negative). Low-level tasks like selection, brushing, search, and axis inversion and axis reordering are all supported. In order to minimize the visual effect of outliers and leverage the screen space, the scale of the axes is not linear, but quantile; this is evidenced by the axis labels distributed unevenly along the axes.

In Figure 4.1, we compare the words *the* and *angel*. Their position in the G^2 axis reveals that *the* is much more likely to appear in English language than in passwords, as opposed to *angel*. While both have a similar frequency in passwords, the G^2 measure distinguishes "angel" as being much more frequent in passwords than expected. On the left pane, one can see the most frequent passwords containing the word *angel*.

The visualization offered important support for validation of the segmentation results, as it provides quick access to the passwords linked to a word. A known weakness of our segmentation algorithm is the production of noise from passwords that do not contain words (e.g., *I* and *a* would be parsed from *a123i321k*), in particular when a comprehensive source corpora is used (e.g., the Asian name *ho* would be parsed from a seemingly random password like *ts63k7ho*). Those issues were easily spotted using the visualization.

4.1.5 Limitations

Our parser is not multilingual. While there are some foreign words in the source corpora, the occurrence of unknown foreign words causes errors in the segmentation. This affects the accuracy of the syntactic and semantic classifications. If one intends to use our approach in contexts that require high accuracy—study of semantics in passwords from the cultural perspective, for example—, it would also be desirable to improve our named entity disambiguation, which is somewhat arbitrary. Another limitation of our parser is that if new terms begin to be used in passwords (e.g., new company names or slang), they will only be captured once

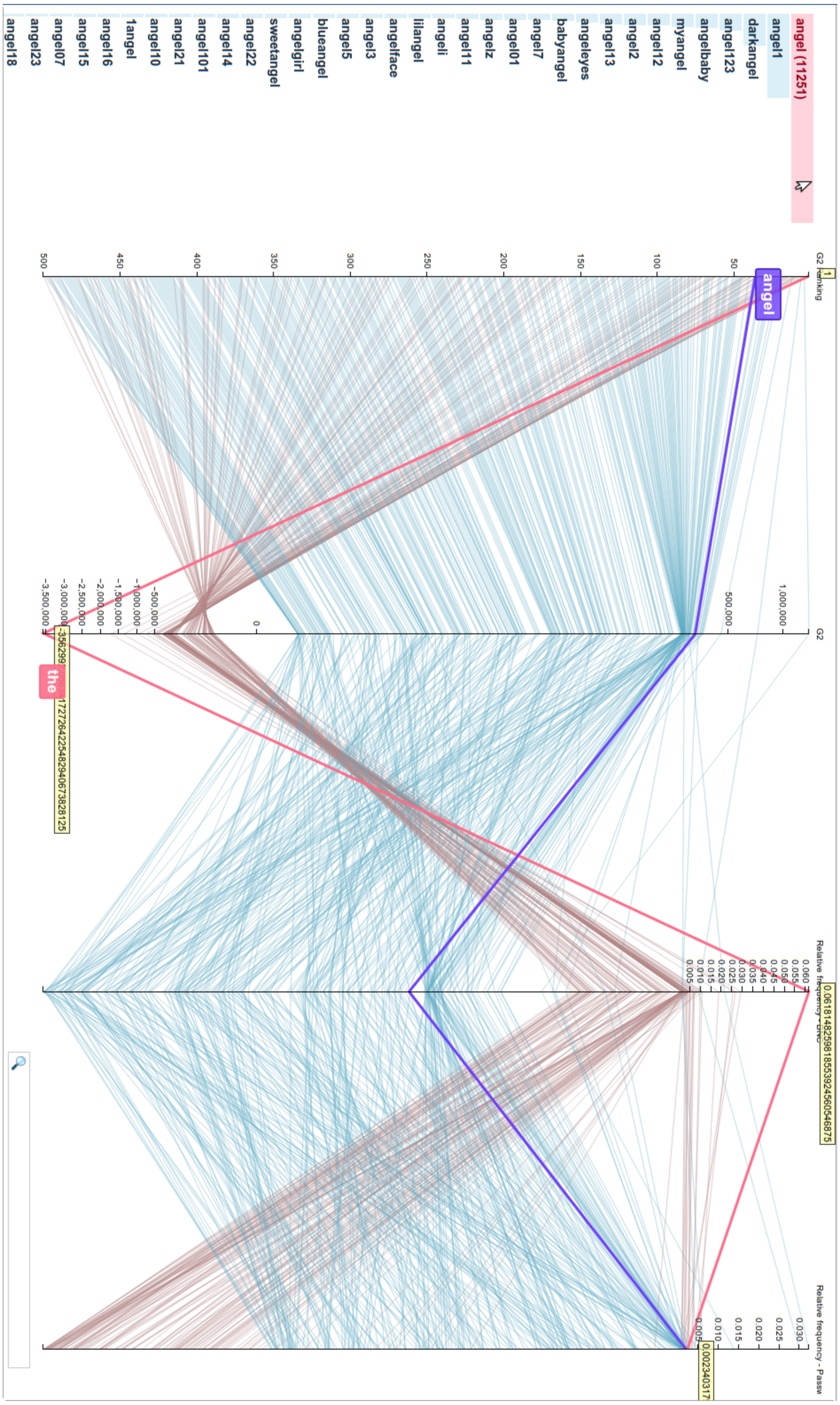

Figure 4.1: Parallel coordinates plot depicting the most frequent word segments extracted from the RockYou list.

Tagger	Coverage (%)
COCA trigram	1.61
COCA bigram	4.48
COCA unigram	89.82
Names	0.25
WordNet	0.4
Default	3.42

Table 4.7: Taggers that compose the backoff model, in order of priority. The coverage column shows the percentage of word segments from the RockYou list tagged by each tagger.

included as part of the source corpora. One could address this problem by using culturally based dynamic dictionaries, such as the ones derived from Wikipedia or Twitter (i.e., hashtag trends).

4.2 Part-of-speech tagging

Part-of-speech tagging is a required step for the semantic classification we perform on nouns and verbs. Beyond that, for security purposes, it is very important to tag words that belong to all other POS classes, because it can potentially lead to further reduction of the search space in cracking attacks. POS tagging benefits from contextual information much like segmentation but, fortunately, there is a wealth of free tools that implement sound POS tagging algorithms which produce reasonable results. In particular, the POS module of the Natural Language Toolkit (NLTK) [Bird, 2006] was used, trained on our data. For each password, the POS function takes as input and array $[s_1, ..., s_n]$, where s_i is a segment, and outputs and array of 2-tuples $[(s_1, t_1), ..., (s_n, t_1)]$, where t_i is a POS tag.

4.2.1 Sequential Backoff Tagger

We rely again on backoff models, since one can be trained easily in NLTK and it has a good balance between simplicity and accuracy [Manning and Schütze, 1999]. In Table 4.7, we show the taggers that compose the backoff model in order of priority. We first try to tag the segments using the COCA trigram tagger (NLTK trigram model trained with the COCA trigrams); if it fails, the COCA bigram tagger is used, and so forth. The tagger is used to tag *only* the word segments of passwords. The names tagger tags anything seen in the names source corpus as *NP* (proper name), while the WordNet tagger searches for a word in the WordNet tree and chooses the POS tag corresponding to the most common sense of the word. Finally, the default tagger is a *custom* tagger which arbitrarily tags any word as *NN* (noun). A default tagger is used to assign the most common tag to words that could not be tagged by any other tagger, so that the backoff tagger has full coverage [Bird et al., 2009]. The unigram tagger, as expected, is the one that tags the majority of words.

4.2.2 Results

The algorithm does a good job in disambiguating the word using the context provided, as in the passwords *gangsterlove* and *ilovestacy* where the word *love* assumes different syntactic functions. The Table 4.8 shows the resulting distribution of segments by POS. The Table 4.9 presents a sample of the POS tagging results.

Category	%	Count
Nouns	73.66	30,935,261
Pronouns	5.70	2,394,372
Adjectives	5.36	2,252,433
Verbs	4.90	2,059,787
Articles	4.06	1,705,886
Others	6.31	2,652,107
Total		41,999,846

Table 4.8: Distribution of the POS tagged segments from RockYou by syntactic category.

4.3 Semantic Classification

After segmenting and POS tagging the passwords, we finally meet the requirements to perform a good semantic classification. At this point, we can represent each password by an array of 2-tuples $S = [(s_1, t_1), ..., (s_n, t_n)]$, where s_i is a segment and

Password	Segment	POS
babygirl87	baby	NN
babygirl87	girl	NN
babygirl87	87	
anthony05	anthony	NP
getyourown	get	VB
getyourown	your	PP$
getyourown	own	JJ
anthony05	05	
gansterlove	ganster	NP
gansterlove	love	NN
gohome01	go	VB
gohome01	home	NR
gohome01	01	
justme7	just	RB
justme7	me	PPO
justme7	7	
L1Lplaya	L1L	
L1Lplaya	play	VB
L1Lplaya	a	AT
ilovestacy	i	PPSS
ilovestacy	love	VB
ilovestacy	stacy	NP
magicmom4	magic	JJ
magicmom4	mom	NN
magicmom4	4	
mowwowdiggydog20	mow	VB
mowwowdiggydog20	wow	UH
mowwowdiggydog20	diggy	NP
mowwowdiggydog20	dog	NN
mowwowdiggydog20	20	
paulradford07	paul	NP
paulradford07	radford	NP
paulradford07	07	
whatever	whatever	WDT
wicked	wicked	JJ

Table 4.9: Sample results of the POS tagging.

t_i is a POS tag (*Null* for gap segments). In this section, we describe an algorithm that takes as input an array of passwords in the format S and outputs for each password an array $K = [(s_1, t_1, c_1), ..., (s_n, t_n, c_n)]$, where c_i is a semantic category. First, we show how WordNet and the source corpora can be used to assign semantic tags to segments (Section 4.3.1). After, in Section 4.3.2, we describe how low-level semantic concepts can be abstracted, allowing us to, later on, characterize semantic patterns in a more general way.

4.3.1 WordNet-based classification

WordNet 3.0 [Fellbaum, 2010] is a large, manually constructed, lexical database of English structured as a network (or graph) of concepts. Each concept is expressed as a synset, a set of synonyms. WordNet covers adjectives, verbs, nouns and adverbs, separately. Concepts are connected through hyperonymy (IS-A) relations[1]; i.e., synsets are arranged into hierarchies, where the top nodes express general concepts and towards the bottom the nodes are increasingly specific. WordNet can be used to group words that share a meaning into a semantic category. For example, the words *car*, *auto*, *automobile* and *motorcar* all refer to the concept *car*, and *car* IS-A *vehicle*. In the WordNet terminology the words are called *lemmas* and the concept is called a *synset*.

In our semantic classification of password segments, verbs and nouns are the *only* classes that receive a semantic tag. Adjectives in WordNet are not connected through hyperonymy relations, but through other relations, such as antonymy, that do not contribute to generalization (see Section 4.3.2). In fact, sentiment analysis would be a suitable way to generalize adjectives, but it is out of scope in this dissertation. All other syntactic classes (e.g., pronouns, adverbs, etc.) are not semantically classified because of their limited semantic content—POS suffices as a categorization criterion.

In Algorithm 3, we detail the steps of semantic classification. If s is a gap segment, it is classified according to the Table 4.10, using regular expressions. Next, we test if s is a proper noun. WordNet does not provide comprehensive

[1]There are several other semantic relations (e.g., antonymy, meronymy, holonymy), some of them featured in WordNet; however, we are only interested in hyperonymy, since it contributes to generalization. See section 4.3.2.

Category	Example
number	123
char	LoL
special	*<|:-)
num+special	0:-3
all_mixed	o/\o5̂

Table 4.10: Semantic categories of gap segments.

support to proper nouns (*NP* tags), which account for 55% of the segments from RockYou that are tagged as nouns; thus, if the word is a proper noun, we rely on the source corpora to tag it as month, female name, male name, surname, country or city, in this order. This is necessary because the corpora is ambiguous, e.g., *Paris* is both in the cities and in the female names word lists, so we disambiguate this step by arbitrarily prioritizing the word list. Next, if the word is either a verb or a noun, we reduce it to its stem (stemming) and find its synsets in WordNet. A word might have different associated synsets (one for each sense), which are ordered according to their frequency count, from most to least frequently used [Fellbaum, 1998]. However, according to the WordNet documentation, frequency information was last updated in 2001 and is no longer maintained; so the sense ordering should not be construed as an accurate indicator of frequency of use. As we do not need very accurate sense disambiguation, we choose the first synset, whose name becomes the semantic tag of the word. The name has the form *word.pos.#*, where # is the sense number; for example, *love.n.01* is the first noun sense of "love". Table 4.11 shows the synsets chosen for nouns and verbs extracted from a sample of the RockYou passwords.

4.3.2 Generalization

We saw in the previous section that our WordNet-based semantic classification groups words with same meaning into synsets; however, it does not consider the hyperonymy relations between synsets. For example, the words *dolphin* and *butterfly* would not be grouped under the *animal* synset, even though they are hyponyms of *animal*. The ability to generalize semantic categories is desirable, given that we

Algorithm 3 Classify segments by semantic category

```
procedure ClassifySemantic(S)
    K ← []
    for all (s, t) ∈ S do
        c ← null
        if s is a gap segment then
            classify by gap category
        else if t is a proper noun tag then
            c ← source corpus name
        else if t is either a verb or a noun tag then
            s ← Stem(s)
            synsets ← LookupWordnet(s)
            if Length(synsets) > 0 then
                c ← synsets[0].name
            end if
        end if
        Append(K, (s, t, c))
    end for
    return K
end procedure
```

could characterize patterns in a more general, concise way; for example, if several kinds of animal appear with consistent frequency in the sample, we could abstract and tag them all as *animal*. Nonetheless, each synset is linked to a chain of hypernyms, and selecting the appropriate hypernym automatically is difficult. Consider the synset *dove.n.01*, whose six first hypernyms are *pigeon.n.01*, *columbiform_bird.n.01*, *gallinaceous_bird.n.01*, *bird.n.01*, *chordate.n.01* and *animal.n.01*. Which synset is more appropriate to represent *dove.n.01* at a higher level? A naive solution would consist in choosing a certain level of abstraction based on observation of the WordNet tree; however, the WordNet tree is highly uneven, to the extent that while a level of abstraction might be appropriate for a certain subtree, it will likely be too general or specific for other. This can be noticed in Figure 4.2, which portrays through a treemap plot the highest level WordNet verb synsets with size proportional to frequency in the RockYou passwords. A gray square represents an internal node, while a beige square represents a leaf node. In the figure, some abstract synsets (e.g., change.v.01 and travel.v.01) have somewhat balanced frequency amongst its children; thus, generalization would be reasonable, as it would provide gain in conciseness with not much loss in accuracy. Other abstract synsets (e.g., connect.v.01 and miss.v.01) have words mostly concentrated in one or few children, making the cost of generalization higher; that is, high loss of accuracy with low gain in conciseness. Ideally, we need to define an *uneven* horizontal cut in the WordNet tree establishing a flexible level of generalization.

To automatically find that cut, we make use of the *tree cut model* by Li and Abe [1998]. Given a sample S, where each data item s is an occurrence of a synset in passwords and a hierarchy (tree) of categories abstracting the synsets, the tree cut model selects the tree cut that represents the best generalization level for the sample. Each internal node of the tree represents a semantic category, and each leaf node represents an instance of the classes above. The frequency of the leaves correspond to the observed frequencies in the samples, and are accumulated by the internal nodes. The tree cut model defines a horizontal cut M across the tree, so that the nodes belonging to the cut abstract all nodes underneath; in other words, a tree cut defines an uneven generalization level for the tree.

The tree cut model is based on the Minimum Description Length Principle, with roots in Information Theory. The principle basically states "that any regularity in a given set of data can be used to compress the data, i.e., to describe it using fewer symbols than needed to describe the data literally" [Grünwald et al., 2005]. Thus, with a good estimation of the probabilities that underlie the occurrence of data items, it is possible to efficiently encode the sample.

Roughly, the tree cut model selects the cut that has the best balance between two metrics: $L_{par}(M)$ (parameter description length) and $L_{dat}(M)$ (data description length), which are involved in a trade-off. $L_{dat}(M)$, which measures how far the tree cut model M is from the data, is proportional to the abstraction level—the greater the abstraction level, the lesser the model fits the data. $L_{par}(M)$, on the other hand, represents the size of the cut and is inversely proportional to the abstraction level. Ideally, we want a small $L_{par}(M)$ (good level of generalization) but with a good fit to the data (small $L_{dat}(M)$). Technically, the algorithm of Li and Abe [1998] minimizes the sum of $L_{par}(M)$ and $L_{dat}(M)$, referred to as model

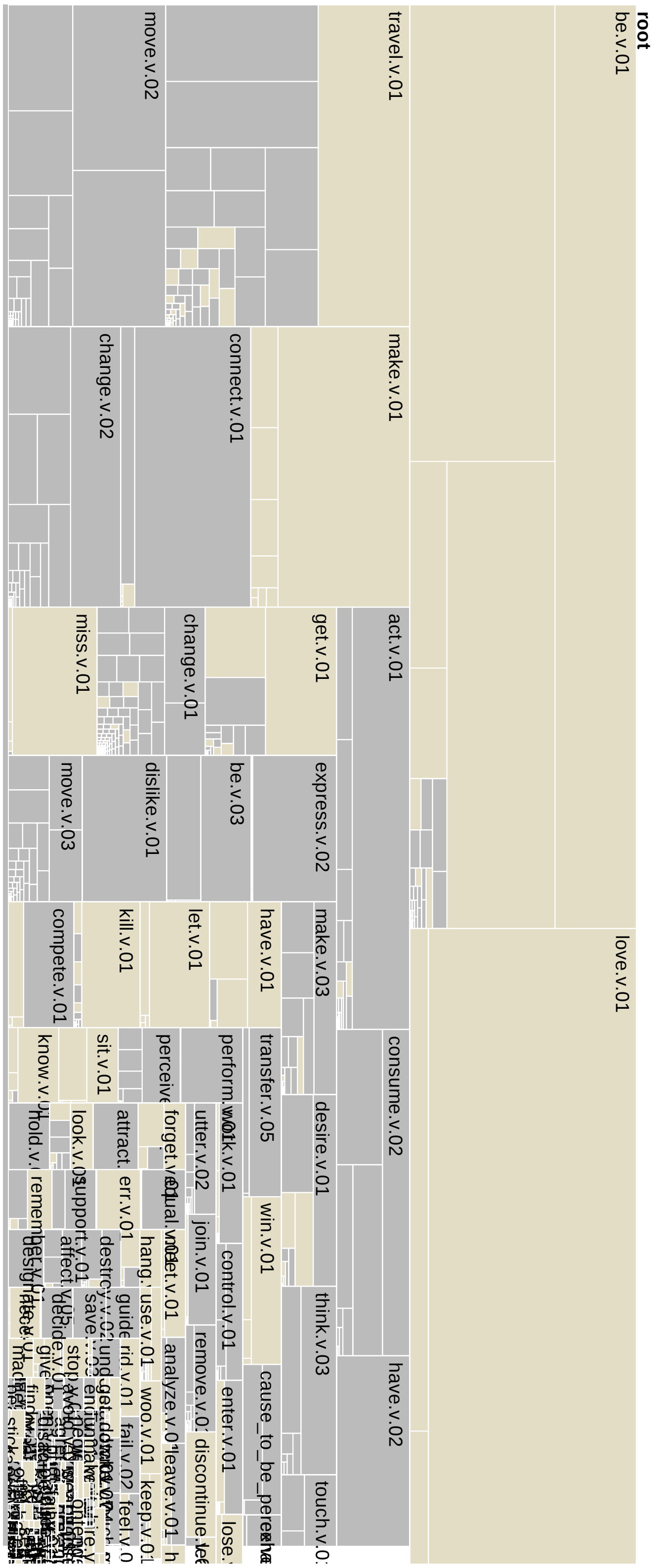

Figure 4.2: Treemap plot showing the distribution of verbs extracted from RockYou among high level WordNet synsets.

description length $L_{mod}(M)$:

$$L_{mod}(M) = L_{par}(M) + L_{dat}(M) \tag{4.5}$$

The parameter description length is calculated as in Equation 4.6, where k is the number of nodes (classes) in the cut and $|S|$ is the sample size:

$$L_{par}(M) = \frac{k}{2} \times log|S| \tag{4.6}$$

The data description length is given by Equation 4.7:

$$L_{dat}(M) = -\sum_{s \in S} log \hat{P}(s) \tag{4.7}$$

where $s \in S$ is the occurrence of a synset in the sample and $\hat{P}(s)$ represents the probability of the category that abstracts the synset in the cut, normalized:

$$\hat{P}(s) = \frac{1}{|C|} \times \hat{P}(C) \tag{4.8}$$

$|C|$ denotes the number of leaves (synsets) under a class, and $\hat{P}(C)$ is given by

$$\hat{P}(C) = \frac{f(C)}{|S|} \tag{4.9}$$

where $f(C)$ is the total frequency of instances of class C in the sample.

4.3.2.1 Adapting the tree cut model to WordNet

The tree cut model was developed for a thesaurus tree; however, WordNet is a directed acyclic graph, so we need to convert it to a tree to get a correct model. Furthermore, the internal nodes in WordNet represent simultaneously semantic categories and word senses, while the tree cut model assumes that internal nodes are categories and leaves are senses. Therefore, the following steps are performed to convert WordNet to a suitable representation [Wagner, 2000]:

1. Duplicate synsets containing multiple parents (hypernyms), for example, *warm_up.v.04*:

   [*use.v.01*, *work.v.12*, *warm_up.v.04*]

[*use.v.01*, *work.v.12*, *exercise.v.03*, *warm_up.v.04*]

2. Divide frequency count between duplicated (ambiguous) synsets.

3. Split internal nodes into word sense and semantic classes by creating a child leaf node that represents the sense. For example:

   [*use.v.01*, *work.v.12*, *warm_up.v.04*]

   becomes [*use.v.01*, *work.v.12*, *warm_up.v.04*, *s.warm_up.v.04*]

In addition, Wagner [2000] reports that the algorithm of Li and Abe [1998] "tends to over-generalize for infrequent verbs and to under-generalize for frequent verbs". Wagner noticed that L_{par} and L_{dat} have different complexities with respect to the sample size $|S|$. L_{par} has the complexity $O(log|S|)$, while L_{dat} has the complexity $O(|S|)$, as seen in Equations 4.7 and 4.6. That means that, in our case, the size of the sample has influence over the level of generalization; so as the sample gets larger the algorithm tends to under-generalize—a fact that has been observed in our experiments. Wagner [2000] then proposes a weighting factor, which is essentially a free parameter that introduces some flexibility in the calculation regarding the level of generalization. This parameter, hereby called W, is introduced in the Equation 4.5 as following:

$$L_{par}(M) + W\left(\frac{log|S|}{|S|}\right) L_{dat}(M) \quad (C > 0) \tag{4.10}$$

The value of the parameter W, however, is chosen arbitrarily; so in order to evaluate the choice of this parameter, we prototyped an interactive visualization that allows the comparison of tree cuts resulting from different W values. In Figure 4.3, the visualization shows a representation of the subtree rooted at the node *carnivore.n.01*, where frequency is cumulative and encoded by color (the higher the value, the darker the node). The golden line represents the tree cut resulting from using $W = 1,000$, while the red line corresponds to $W = 5,000$ and the blue line to $W = 10,000$.

Roughly, the tree cut model only generalizes groups of synsets whose frequencies are, to some extent, uniform, and this extent can be adjusted by the W parameter, as discussed previously. It is evident in the visualization that smaller W values lead to more general cuts. For example, with $W = 1,000$ all types of wild cats are represented by the concept *wildcat.n.01*, which crosses the golden cut; however, the disparity between the frequencies of *wildcat.n.01* and its siblings prevents the generalization to *cat.n.01*. On the other hand, at $W = 5,000$, the algorithm preserves the distinction between all kinds of wild cats, and at $W = 10,000$, the level of specificity is raised, with the cut discriminating types of lynx, such as bobcat.

This behaviour matches closely the human intuition. Entities that occur uniformly tend to be generalized, while deviating entities are treated individually. From an analytical point of view, the generalization helps to shed light upon highly occurring concepts. For example, the fact that none of the cuts crosses *dog.n.01* reveals that in passwords there might be preferences towards certain types of dog, such as bulldog. After examining several parts of the whole tree, we concluded that the value $W = 5,000$ leads to a generalization level that significantly reduces the complexity of the classification (i.e., number of categories), while highlighting highly divergent categories.

4.3.3 Resulting Semantic Categories

In Table 4.11, we show a sample of the results of the semantic classification. The Semantic tag column shows the semantic tags assigned to the password segments after generalization (described in the previous section). The effect of generalization can be observed by comparison of the semantic tags with the corresponding synsets. For example, in the password *671soldier*, the segment *soldier* is classified as *worker.n.01*, a generalization of the synset *soldier.n.01*. Notably, some synsets are not generalized (e.g., *puppy.n.01*).

We also show the top 100 semantic categories in Table 4.12, the contents of which reveal some interesting insights about the semantics of passwords. Categories reported by surveys appear (e.g., names and dates), but also new categories appear such as love (lines 6 and 7), places (lines 3 and 13), sexual terms (lines 29, 34, 54, and 69), royalty (lines 25, 59, 60), profanity (line 40, 70, and 72), animals (lines 33, 36, 37, 92, 96, and 100), food (61, 66, 76, 82, and 93), alcohol (line 39), and money (line 46 and 74). Some categories, noted with +, contain within them some noise caused by the parsing of two-letter words that occur in our com-

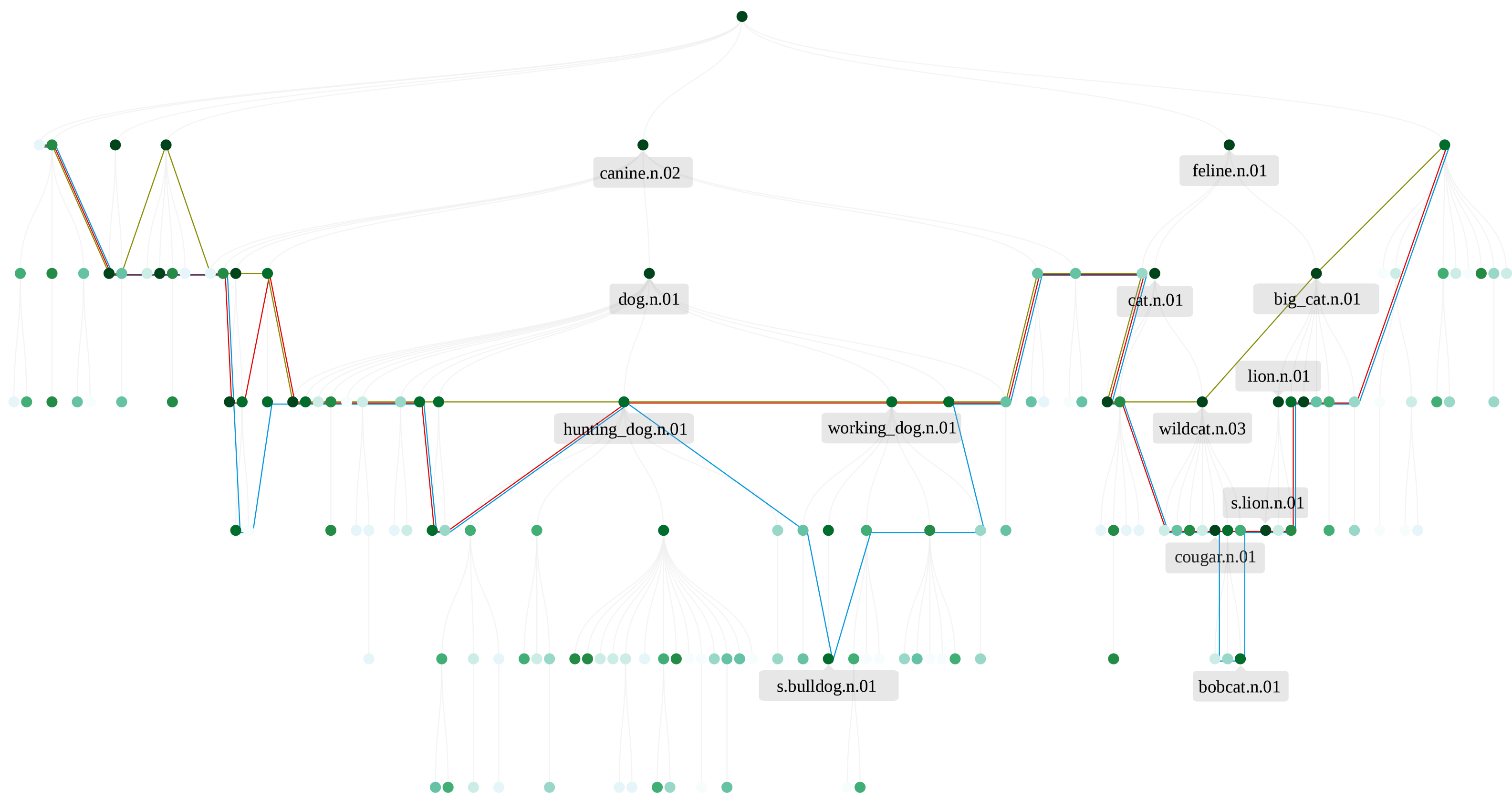

Figure 4.3: Tree visualization showing a subtree rooted at the node carnivore.n.01 and the cuts resulting of the following weighting values: 1,000 (golden), 5,000 (red) and 10,000 (blue).

Password	Segment	POS	Semantic Tag	Synset
hope87	hope	VB	wish.v.01	hope.v.01
hope87	87		number	
serenity	serenity	NN	trait.n.01	repose.n.03
bishop5	bishop	NN	status.n.01	bishop.n.01
bishop5	5		number	
slutsister	slut	NN	vulgarian.n.01	slattern.n.02
slutsister	sister	NN	s.sister.n.01	sister.n.01
fuckyou05	fuck	VB	s.sleep_together.v.01	sleep_together.v.01
fuckyou05	you	PPO		
fuckyou05	05		number	
goblue0507	go	VB	s.travel.v.01	travel.v.01
goblue0507	blue	NN		
goblue0507	507		number	
looted	looted	VBN	take.v.21	loot.v.01
drift21	drift	NN	force.n.02	drift.n.01
drift21	21		number	
candysinger	candy	NN	s.candy.n.01	candy.n.01
candysinger	singer	NN	musician.n.01	singer.n.01
671soldier	671		number	
671soldier	soldier	NN	worker.n.01	soldier.n.01
bravo100	bravo	NN	murderer.n.01	assassin.n.01
bravo100	100		number	
egobrain	ego	NN	pride.n.01	ego.n.01
egobrain	brain	NN	structure.n.04	brain.n.01
pitcher9	pitcher	NN	athlete.n.01	pitcher.n.01
pitcher9	9		number	
puppies	puppies	NNS	puppy.n.01	puppy.n.01
church	church	NN	religion.n.02	church.n.01
‘ale‘8	‘		special	
‘ale‘8	ale	NN	alcohol.n.01	ale.n.01
‘ale‘8	‘8		num+special	
‘18angelnjohany	‘18		num+special	
‘18angelnjohany	angel	NN	s.angel.n.01	angel.n.01
‘18angelnjohany	n		char	
‘18angelnjohany	johany	NP	mname	

Table 4.11: Sample of passwords with segments classified by semantics. The Semantic tag column shows the final semantic category of a segment, after synset generalization.

	Category	% Freq		Category	% Freq
+1	mname	20.609	*51	junior.n.04	0.099
+2	fname	16.697	*52	multiple_sclerosis.n.01	0.098
3	city	3.283	53	collection.n.01	0.098
+4	surname	1.214	54	s.sexual_activity.n.01	0.097
5	s.be.v.01	1.131	55	s.day.n.01	0.097
6	s.love.v.01	0.930	*56	megahertz.n.01	0.093
7	s.love.n.01	0.917	57	s.car.n.01	0.092
8	s.baby.n.01	0.707	*58	s.ad.n.01	0.089
9	month	0.572	59	s.lady.n.01	0.087
10	s.girl.n.01	0.467	60	s.king.n.01	0.086
11	structure.n.01	0.350	61	honey.n.01	0.085
*12	tellurium.n.01	0.345	62	inhabitant.n.01	0.082
13	country	0.341	*63	disk_jockey.n.01	0.082
14	s.rock.n.01	0.336	64	s.hood.n.01	0.080
15	s.male_child.n.01	0.334	65	group_action.n.01	0.079
16	s.angel.n.01	0.311	66	s.pie.n.01	0.074
17	s.man.n.01	0.306	*67	titanium.n.01	0.074
18	password.n.01	0.305	*68	actinium.n.01	0.073
*19	ma.n.01	0.297	69	s.kiss.n.01	0.072
20	woody_plant.n.01	0.290	70	buttocks.n.01	0.071
21	worker.n.01	0.275	71	s.friend.n.01	0.071
22	s.make.v.01	0.221	72	crap.n.01	0.070
23	commodity.n.01	0.214	73	s.miss.v.01	0.070
*24	dad.n.01	0.214	74	s.money.n.01	0.069
25	s.princess.n.01	0.213	75	traveler.n.01	0.068
*26	chemical.n.01	0.206	76	starches.n.01	0.067
27	s.life.n.01	0.203	77	s.ball.n.01	0.064
*28	s.knockout.n.02	0.187	78	s.sunlight.n.01	0.063
29	s.sleep_together.v.01	0.184	*79	gilbert.n.01	0.063
30	s.travel.v.01	0.175	80	fellow.n.06	0.063
*31	ka.n.01	0.171	81	s.get.v.01	0.062
32	s.star.n.01	0.169	82	cocoa.n.01	0.062
33	s.dog.n.01	0.166	83	s.rise.v.01	0.061
34	s.lover.n.01	0.154	84	craft.n.02	0.061
35	herb.n.01	0.145	85	s.monday.n.01	0.060
36	s.cat.n.01	0.135	86	s.family.n.01	0.060
37	s.monkey.n.01	0.133	87	s.hate.v.01	0.060
*38	district_attorney.n.01	0.131	*88	bachelor_of_arts.n.01	0.060
39	alcohol.n.01	0.125	89	s.football.n.01	0.060
40	bitch.n.01	0.122	90	s.manner.n.01	0.060
*41	qi.n.01	0.121	91	s.state.v.01	0.059
*42	polonium.n.01	0.114	92	bug.n.01	0.058
43	wood.n.01	0.114	93	s.candy.n.01	0.058
*44	selenium.n.01	0.110	94	musician.n.01	0.057
45	soccer.n.01	0.110	95	summer.n.01	0.057
46	monetary_unit.n.01	0.109	96	s.dragon.n.01	0.057
47	s.child.n.01	0.106	97	s.kit.n.01	0.057
48	s.bear.v.01	0.106	98	dish.n.02	0.056
*49	en.n.01	0.103	99	s.ice.n.01	0.055
*50	mister.n.01	0.102	100	s.butterfly.n.01	0.055

Table 4.12: Most probable semantic categories in the RockYou list

prehensive names dictionary, such as “li”, and “ho”, where it is likely no name was intended. Other categories, noted with *, likely result from noise artifacts from two letter words (e.g., *polonium* “Po” and *multiple sclerosis* “MS”). Of particular interest is *tellarium*, coming from the word “te", which upon investigation of the data set appears to tend to occur quite often in passwords containing the Spanish phrase “te amo" (which means “i love you").

Chapter 5

Semantic Guess Generator

The hypothesis driving this dissertation is that passwords can be characterized by semantic patterns, which can help us understand more about the effective security provided by passwords in practice. In order to verify this hypothesis, the mere semantic classification of the password segments is not enough. We need a model to capture the structural relationships of semantic classes and encode the probabilities of different constructs. The intuition behind the usefulness of semantic patterns is that some words tend to pair up with specific classes of words. This occurs due to selectional preferences that depend both on part-of-speech and meaning; for example, a verb calls for a noun, and the verb *eat* is most probably followed by the name of a food. From the security point of view, this may represent a significant reduction in the search space in a cracking session, i.e., the guesser will only try or prioritize guesses that are probable both in the semantic and in the syntactic levels. Computational linguists have been representing those patterns through grammars; however, we cannot assume that people follow the grammar of English in passwords, since they have no reason to do so; hence, the algorithm needs to learn the *passwords grammar*. Following Weir et al. [2009], we employ probabilistic context-free grammars to model the syntactic and semantic patterns of passwords. With this model we can learn the semantic patterns from a sample and generate passwords previously unseen. Then a suitable way to evaluate the fitness of our model, i.e., how well passwords can be characterized by semantic patterns, is using it to generate guesses for cracking attacks. The extent by which those attacks are successful is at the same time an indicator of how well the patterns are captured by the model and evidence of their security implications.

5.1 Probabilistic Context-Free Grammars

A probabilistic context free grammar (PCFG) is a context free grammar whose productions have associated probabilities. A PCFG represents a syntax, i.e., it shows how words group together and relate to each other as heads and dependents, and it is used either to parse or generate the sentences of a language [Manning and Schütze, 1999]. PCFGs were used in passwords first by Weir et al. [2009] to learn mangling patterns from the RockYou list and generate guesses in highest probability order. Under the assumption that long passwords are likely to follow English grammar rules, Rao et al. [2013] used a context-free grammar of English to generate guesses targeting long passwords.

A generic PCFG G consists of:

- A set of terminals, $\Sigma = \{w_1, ..., w_m\}$. This is the vocabulary of the grammar, that forms the content of the sentences.
- A set of nonterminals, $V = \{N_1, ..., N_n\}$, also known as variables, are the syntact categories of the grammar.
- A start variable N_1.
- A set of rules $N_i \rightarrow \zeta_j$, where ζ_j is a sequence of terminals and nonterminals and represents the j^{th} rule of N_i.
- A set of probabilities on rules, such that $\forall i \; \sum_j P(N_i \rightarrow \zeta_j) = 1$.

In our PCFG, Σ is a set comprised by the source corpora and the learned gap segments, and V is the set of semantic and syntactic categories. The rules are all of the form $N_i \rightarrow w_k$, i.e., a nonterminal derives exactly one terminal, or $N_1 \rightarrow \xi$, where ξ is a sequence of nonterminals.

Since we have syntactic and semantic categories, and both are relevant to characterize patterns, we combine both types of categories to compose the nonterminal set. For nouns and verbs semantically classified, we overload a nonterminal symbol with both semantic and syntactic information; for example, in the nonterminal

	Rule	Prob.
A	$N_1 \rightarrow$ [PP][love.v.01.VV0][PP][number]	0.33
B	$N_1 \rightarrow$ [PP][hate.v.01.VVD][PP][number]	0.33
C	$N_1 \rightarrow$ [sport.n.01][number]	0.33
D	[PP] $\rightarrow$ i	0.5
E	[PP] $\rightarrow$ you	0.25
F	[PP] $\rightarrow$ them	0.25
G	[love.v.01.VV0] $\rightarrow$ love	1
H	[hate.v.01.VVD] $\rightarrow$ hated	1
I	[sport.n.01] $\rightarrow$ football	1
J	[number] $\rightarrow$ 2	0.5
K	[number] $\rightarrow$ 3	0.5

Table 5.1: Sample grammar learned from the training set *iloveyou, ihatedthem3, football3*

love.v.01.VVD we have the concatenation of a semantic (*love.v.01*) and a POS category (*VVD*). This symbol should derive only the verbs categorized as *love* that are inflected in the past tense. In this way, we increase the descriptive power of the grammar.

Example 2. $N_1 \rightarrow [pronoun][love.v.01.VVD][pronoun][number]$

The rules and the corresponding probabilities can be learned from a password training set by a simple algorithm. Given a segmented password, its semantic/syntactic structure constitutes the right-hand side of the rule. Example 2 shows the rule learned from the password *ilovedyou2*. The segments that carry a semantic tag (nouns and verbs) lead to POS- and semantic-based symbols (*love*), while all others lead to POS-based symbols (*I* and *you*). The probability of such a rule is simply its relative frequency, given by $P(rule) = C_r/C_t$, where C_r is the count of matching passwords and C_t is the total count of passwords. In the same way, the algorithm can learn rules that generate the terminals and their probabilities. In Table 5.1, we show an example PCFG learned from the set of passwords $\{iloveyou2, ihatedthem3, football3\}$.

Consistent with the nomenclature adopted by Weir et al. [2009], we call the structures derived from the start variable *base structures*, i.e., right-hand side of all N_1 rules. Table 5.2 lists the most probable base structures learned from the

	Base structure	Probability
1	[number]	0.1596848
2	[female_name]	0.0400706
3	[male_name][number]	0.0388099
4	[female_name][number]	0.0346827
5	[male_name]	0.0326887
6	[all_mixed]	0.0256102
7	[NN]	0.0200257
8	[NP]	0.0156618
9	[NP][number]	0.0141660
10	[city]	0.0138238
11	[NN][number]	0.0136044
12	[JJ_None][number]	0.0108745
13	[city][number]	0.0094536
14	[JJ_None]	0.0088301
15	[male_name][male_name]	0.0067594
16	[female_name][female_name]	0.0043687
17	[female_name][male_name]	0.0033002
18	[male_name][all_mixed]	0.0032848
19	[month][number]	0.0030383
20	[surname]	0.0026550
21	[male_name][female_name]	0.0025552
22	[number][male_name]	0.0023853
23	[male_name][male_name][number]	0.0021915
24	[male_name][char]	0.0020304
25	[male_name][NP]	0.0020132
26	[surname][number]	0.0019242
27	[NN_password.n.01]	0.0019052
28	[PPSS][VB_s.love.v.01][PPO]	0.0018857
29	[male_name][NN]	0.0017838
30	[number][female_name]	0.0017247
31	[NPS]	0.0017000
32	[female_name][all_mixed]	0.0016357
33	[female_name][NP]	0.0016258
34	[num+special]	0.0015592
35	[JJ]	0.0015569
36	[NP][male_name]	0.0015348
37	[NP][all_mixed]	0.0015201
38	[char][male_name][number]	0.0014844
39	[NN][male_name]	0.0014806
40	[female_name][NN]	0.0014651
41	[NN_s.love.n.01][number]	0.0014467
42	[female_name][char]	0.0014203
43	[female_name][female_name][number]	0.0013897
44	[JJ][number]	0.0013775
45	[country]	0.0013773
46	[NN][all_mixed]	0.0013599
47	[PPSS][VB_s.love.v.01][male_name]	0.0013497
48	[NN][NN]	0.0013090
49	[char]	0.0012494
50	[NP][NP]	0.0012134

Table 5.2: 50 most probable base structures of the grammar trained with the RockYou list.

Approach	Base structures	Non-terminals	Terminals	Terminal Struct.	MySpace attack (%) Guessed passwords	MySpace attack Approx. # of guesses ⋆
Semantic	1,861,821	12,410	4,045,458	1.3×10^{86}	91.76	4.8×10^{11}
Weir	78,126	166	3,554,133	1.8×10^{73}	60.83	8.2×10^{9}
Brute force (bounded by Semantic)					91.76	3.2×10^{43}

Table 5.3: Comparison between grammars generated by the semantic and Weir approaches trained with the RockYou list, and a comparable brute force attack. ⋆ See Section 5.4.4 for description of approximation methods and brute force comparison.

RockYou list. A base structure after the rewriting of all its nonterminal symbols is called a *terminal structure*, and it is effectively a password generated by the grammar. The probability of a terminal structure is the product of the probability of the base structure with the probability of all the rules required for its derivation. For example, $P(youlovethem2) = P(A) \times P(E) \times P(G) \times P(F) \times P(J) = 0.0103125$. Table 5.3 shows a comparison between the PCFGs generated by our approach and the approach of Weir et al. [2009], both trained with the RockYou list.

5.2 Building a guess generator

Password cracking usually involves some software that can read or generate a *guess*, hash it using the same hashing algorithm used by the target and compare it against all the target hashes. The most prominent program is John the Ripper (JtR) [Openwall]. When a comparison results true, we have a *hit*, i.e., a password was successfully guessed. The popular approaches for generating the guesses are either based on word lists or brute force. In the word list approach, the guesses come from a large list of strings, or a compilation of lists. Word lists are manually curated and available from a variety of sources on the web. They usually contain strings that are highly used as passwords, and strings found in previous leaks. The limitation of word lists is obvious: a password not listed there will not be guessed. To overcome this limitation, John The Ripper comes with a mangling option, where it reads a guess from the word list and derives variations based on a configurable set of heuristics, e.g., *password* → *p4ssw0rd*. In this case, a wordlist

Algorithm 4 Generates guesses in highest probability order

```
procedure GenerateGuesses(G)
                ▷ Initialize priority queue with most probable derivation of each base structure
    queue ← initialize priority queue
    for all G.base_structures do
        guess ← initialize guess
        guess.terminals ← most probable terminal values for the base struct.
        guess.pivot ← 0
        guess.p ← calculate probability of the guess
        Insert(queue, guess)
    end for

    c ← Pop(queue)
    while c ≠ NULL do                                   ▷ Generate password guesses
        Output(c)                                       ▷ Output current guess
        for i ← c.pivot, Len(c.terminals) do  ▷ Derive lower probability guesses from the same base structure
            new ← initialize new guess
            new.terminals ← Decrement(c.terminals, G, i)    ▷ Replace c.terminals[i] by the next lower probability terminal at i
            if new.terminals ≠ NULL then
                new.p ← calculate probability
                new.pivot ← i
                Insert(queue, new)
            end if
        end for
        c ← Pop(queue)
    end while

end procedure
```

of a couple of million entries can generate dozens of millions of guesses. In the brute force strategy, an algorithm progressively generates all possible strings up to a maximum length. In addition, JtR features a "smart brute force" mode, where it uses a Markov model to prioritize the generation of guesses containing more frequent letters.

In a realistic cracking session, crackers first exhaust the possibilities of the word list mode and then switch to a brute force attack, which cracks passwords in a much lower hits/guesses ratio. This strategy can potentially crack the most common passwords fast, but will take a long time to guess all the passwords; so the larger the number of passwords cracked before switching to the brute force mode, the better (for the attacker). As previously mentioned, Weir et al. [2009] used PCFGs to learn mangling rules and generate guesses in optimal probability order. Their approach shows good results when the training set is very similar to the target. As we will see in Section 5.4, when the password creation policy of the target is different, affecting the choice of mangling rules, their method degenerates quickly. We hypothesize that using the same PCFG framework, but learning semantic patterns in addition to mangling rules, will be more accurate in generating realistic guesses.

Once we have the grammar trained, building a guess generator is just a matter of outputting the terminal structures in highest probability order. This said, the algorithm for this job is not exactly trivial. Fortunately, Weir et al. [2009] proposed Algorithm 4, which works well for this purpose. Our PCFG is able to generate an enormous number of guesses (1.3×10^{86}) when trained on RockYou. For the sake of comparison, the approach of Weir et al. [2009] (hereafter referred to as the *Weir approach*, for convenience) trained on the same RockYou list and using dic-0294 as the input dictionary can generate around 1.8×10^{73} guesses.

5.2.1 Custom Mangling

The semantic guess generator only generates guesses containing lowercase word segments; gap segments, instead, are learned (and derived) in the form they appear in the passwords. Case mangling of word segments, however, is a desirable feature, since it is a common mangling pattern. Table 5.4 shows the case statistics for the word segments we extracted from the RockYou passwords, where the mangled category corresponds to words that do not fall in any other category, e.g., *hOUse*. Even though lowercase guesses would not be a high limiting factor against RockYou, it would probably severely limit the guessing success of our generator against targets that enforce strong password creation policies. Thus, we developed a version of the guess generator that applies a small set of custom mangling rules to word segments. Gap segments always preserve their original case.

Rule	Count	%
lowercase	39,516,827	94.09
uppercase	1,658,417	3.95
capitalized	718,318	1.71
mangled	106,284	0.25
Total	41,999,846	

Table 5.4: Case statistics of word segments extracted from RockYou passwords.

Capital Capitalizes the first word segment, e.g., *bearDOG123LoL* → *Beardog123LoL*. This rule is only applied to guesses that begin with a word segment, i.e., words derived from all non-terminal symbols, except *mixed_all*, *mixed_num_sc*, *number*, *special* and *char*.

Uppercase Uppercases all characters of word segments, e.g., *bearDOG123LoL* → *BEARDOG123LoL*.

Camel Case Capitalizes all word segments, e.g., *bearDOG123LoL* → *BearDog123LoL*.

It is worth highlighting the sophistication of the camel case rule, which is only possible with password segmentation, a feature not present in the state-of-the-art password crackers.

5.3 Comparison with previous approach

Our approach can be seen as an evolution of the Weir approach. Before presenting the experiments that show to what extent the semantic approach outperfoms the state-of-the-art techniques, in this section we enumerate the points where our technique deviates from the Weir approach.

5.3.0.1 Base Structures

The Weir approach uses only a small set of non-terminal symbols: D_n (digits), S_n (special characters) and L_n (alphabetic strings), where n is the string length. As seen in Table 5.3, our method trained on the RockYou list generates a much finer grained grammar, with 12,410 non-terminal symbols, in comparison with the 166 non-terminals generated by the Weir approach trained on the same list. This leads to more precise probability estimates.

5.3.1 Terminals

As opposed to our method, the Weir approach does not include rules to derive alphabetic strings, i.e., it does not "learn" them. Their method takes a dictionary as input and estimates the probability of a word w of length n as the relative frequency $1/C_n$, where C_n is the count of words of length n. Since the number of distinct short words is reduced (e.g., $argmax(C_1) = 26$), this strategy tends to favour guesses containing short alphabetic strings.

5.3.2 Input

The Weir guessing algorithm takes two parameters as input, a grammar and an input dictionary. In our method, the "input dictionary" (equivalent to Terminals in Table 5.3) is embedded in the grammar. While this provides the already mentioned advantages, it impacts on flexibility. If we want to use a different set of terminals (e.g., COCA unigrams or a foreign language corpus), rules need to be created linking them to the non-terminals (semantic and syntactic categories), which requires re-running the semantic classifier.

5.4 Experiments

5.4.1 Experimental Setup

We use the community enhanced version (bleeding jumbo) of John The Ripper 1.8 [Magnumripper]. This software has a so-called *stdin* mode, where it receives guesses from a third-party program through the standard input. This mode allows us to pipe the guesses from the guess generators to JtR, which performs the hash comparisons. With a small script, JtR's output is parsed and the graphs are generated.

In the experiments, performed in a desktop computer with processor Intel Core i5 CPU 650 @ 3.20GHz $\times$ 4 and 8GB RAM, we limit the number of guesses to 3 billion (due only to memory limitations) and, despite the fact that the target passwords might have known minimum length, we do not filter the guesses, in order to test the methods with no assumptions about the target.

The methods tested are:

1. Semantic approach without mangling rules
2. Semantic approach with custom mangling rules
3. Semantic approach with default JtR's mangling rules
4. Weir approach
5. Wordlist with JtR's default rules, followed by incremental mode[1]

In method 1 the strings are used as generated by our grammar (lowercased), while in method 2 case mangling is applied as described in Section 5.2.1 and in method 3 JtR's default mangling rules are applied. Method 4 uses the strings generated by the software that Weir made available on his personal website [Weir], trained on the same dataset as the semantic approaches (RockYou), and with the input dictionary used in their article (dic-0294). In method 5, we use JtR's wordlist mode with the passwords.txt wordlist (2,151,220 unique values) available at Dazzlepod [Dazzlepod]. According to Dazzlepod, this list has a success rate of 40% using JtR's mangling rules against the Lulzsec collection of hashes (final release).

The primary criterion for the choice of the experimental scenarios is the relevance of the targets, i.e., we focus on large leaks from popular services that gathered major attention and concern of the media. We also consider possible sources of bias, namely, the type of resource being protected, the demographics of users, and the collection method [Jakobsson and Dhiman, 2013].

5.4.2 Experiment 1: Using RockYou Semantics to Guess LinkedIn

In this scenario, the grammars are trained with RockYou, and the target is the LinkedIn list, which was exposed in June 2012. The LinkedIn list contains 5,787,239 unique passwords hashed with unsalted SHA-1, including hashes whose first 5 dig-

[1]Incremental mode in JtR corresponds to the brute force attack enhanced with Markov probabilities.

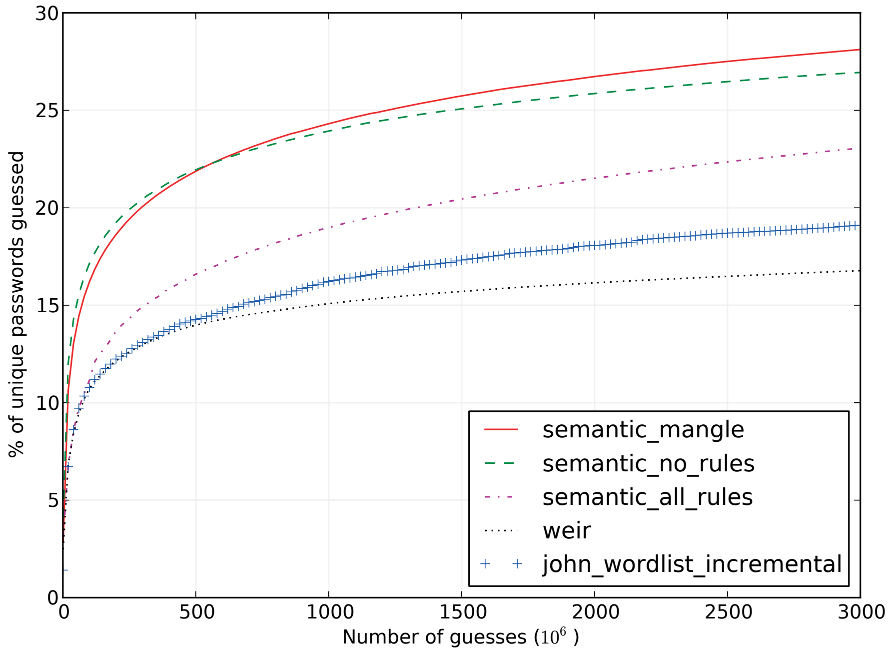

Figure 5.1: Results of Experiment 1. The three variations of the semantic approach perform better than the competing approaches.

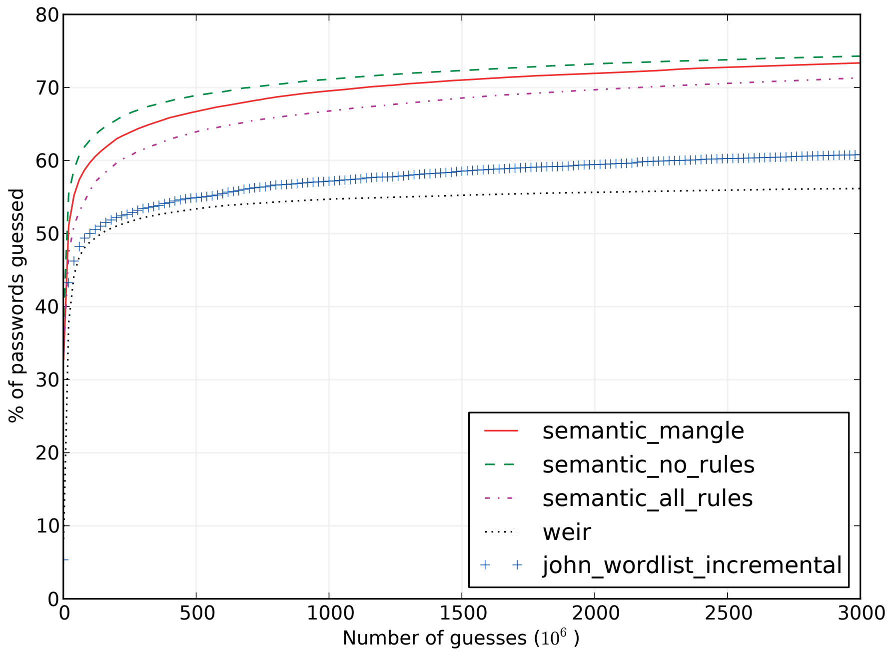

Figure 5.2: Results of Experiment 2. The best semantic condition guesses approximately 32% more passwords than the Weir approach.

its are zeroed; hence, the hash comparison is adapted by passing the parameter --format=raw-sha1-linkedin to JtR. Note that because the LinkedIn list only contains unique hashes, the reported cracking rates do not account for the effect of commonly used passwords. Among the passwords, there are some which are composed of only alphabetic lowercase characters, so we believe the password creation policy was either non-existent or fairly liberal. This leak is relatively free of bias, as the users are predominantly adults with some degree of education and the passwords were somehow stolen (as opposed to phishing). The type of resource being protected (social network profiles), however, is not of the highest risk to personal privacy.

The results show that the semantic approach in all 3 variations outperforms the competing methods (Figure 5.1). The version with custom mangling rules surpasses the version without rules after the 500 millionth guess, probably due to the fact that the target contains passwords with a variety of case configurations. The semantic approach with JtR's default rules is the worst among the three semantic variations. A reasonable explanation for this is that most JtR's mangling rules change the guess structure in some way (e.g., reversing the characters, appending numbers, etc.), violating the highest probability order of the guesses. The Weir approach is in fact the worst, with our approach cracking approximately 67% more passwords than it. This is probably a consequence of it being trained on a list that is not very similar to the target (demographics, type of resource being protected and password creation rules are different). This highlights the robustness of our method: it performs well even when trained with a list that has different characteristics compared to the target. Moreover, our approach cracks 44% more passwords than the Wordlist plus incremental method, which represents the industry standard.

5.4.3 Experiment 2: Using RockYou Semantics to Guess MySpace

In this experiment, we target the MySpace list, one of the first large leaks, exposed in 2006 and collected through phishing. This list is much smaller than the LinkedIn list, containing 49,655 clear text passwords (41,543 unique). In order to keep the consistency with the other experiment, we encoded the passwords with JtR's *dummy* format—hexadecimal prefixed with "$dummy$"—and used the same experimental setup as Experiment 1.

Again, the semantic approach outperforms all the others; in particular, it cracks approximately 32% more passwords than the Weir approach. Because it was obtained through phishing, the MySpace list is arguably composed of weaker passwords. This can be noticed by the fact that the non-mangled version of our algorithm performs better than the version with custom mangling, probably because the proportion of passwords using uppercase characters is not high.

5.4.4 Experiment 3: Final Guessing Success Rate against MySpace

To evaluate the expressiveness of our model, it is necessary to know how many passwords it would eventually guess, i.e., the final guessing success rate; however, it is known that our approach (as well as the Weir approach) can generate a very large number of guesses. Finding the final guessing success rate empirically is, thus, not viable in a reasonable amount of time without powerful computing resources. Yet, it is possible to compute this measure with a simple grammar recognizer, with the constraint that the target passwords should be cleartext. If the grammar recognizes a string, it is guaranteed that the string will be generated, given enough time. To compare the semantic and Weir approaches, we built recognizers for both grammars trained with RockYou and ran them over the MySpace passwords (cleartext). The results, presented in Table 5.3 prove the expressiveness of our model and its superiority in comparison with the Weir approach, which eventually guesses around 30% less MySpace passwords.

Given that our grammar can recognize 91.76% passwords, it is important to know how long it would take to achieve this success rate. We can obtain an estimate of this measure by fitting a nonlinear regression model to a relatively small sample of the guess probabilities. Figure 5.3 shows a sample of the first 600 million guesses, reduced to every 10 millionth guess. The blue curve represents an exponential decay model (Equation 5.1) with the β parameters fitted to the data using the Gauss-Newton method. With the inverse function $g(y)$ we can estimate

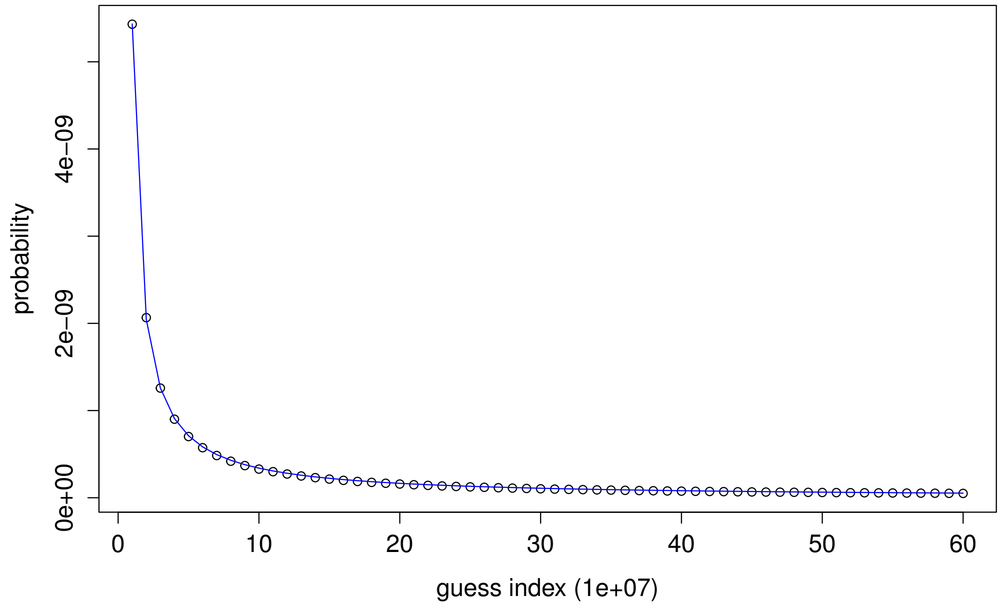

Figure 5.3: Nonlinear regression model of the guess probabilities of the semantic approach.

the guess index x corresponding to a certain probability. As output by our grammar recognizer, the least probable password in the MySpace list is *s6a6t6a6n6i6c*, with $p = 2.8 \times 10^{-27}$. This probability is the ultimate lower bound of the semantic cracking session and can be used as a parameter to $g(y)$. This gives us the approximate number of guesses for the semantic grammar in Table 5.3. Likewise, we can determine the approximate number of guesses of a cracking session using the Weir grammar. Table 5.3 portrays a comparison between the number of guesses of the semantic and Weir approaches, as well as of a brute force attack that would guess 91.76% passwords as a baseline. The brute force attack is simulated by calculating the number of possible guesses needed to cover the search space defined by all strings with length up to 19 characters, which is the longest string guessed by the semantic approach. In this way, we estimate how many guesses are necessary for the brute force attack to achieve the same success rate of the semantic approach.

$$f(x,(\beta_1,\beta_2,\beta_3)) = \frac{\exp(-\beta_1)}{\beta_2 + \beta_3 x} \tag{5.1}$$

5.5 Performance Limitations

In comparison with JtR's modes and the Weir approach, our approach is inferior in terms of time (guesses/second) and memory consumption. The following table shows the average number of guesses per second of each approach, as measured in Experiment 1, against SHA-1 hashes:

Approach	Guesses/s
JtR Wordlist + Incremental	6,172,839
Weir	963,081
Semantic	208,333

Table 5.5: Average guesses/s against SHA-1 hashes.

In fact, we use the same algorithm as the Weir approach to generate guesses, but our grammar contains many more base structures (see Table 5.3).

Further study is needed to detect whether the performance bottleneck is in the complexity of the algorithm or the problem can be solved by optimizing the implementation. Despite that, as presented in the previous section, the semantic approach can be more *efficient* than the other approaches, as measured using an *implementation- and platform-agnostic metric*, namely, success rate (hits/guesses). Notably, the inferior performance can be neglected in cracking sessions against slow hashes, such as *bcrypt*, where the hashing time is the bottleneck, turning the cost of hash comparisons much higher. Table 5.6 shows an empirical assessment of how many hash comparisons (guesses) per second are possible against the hashing functions SHA-1, sha512crypt and bcrypt with a variety of hardware and state-of-the-art software. Whereas about 98×10^6 hash comparisons can be performed with a standard desktop computer against SHA-1, only around 4960 comparisons can be performed against bcrypt in the same conditions. In the latter scenario, the semantic approach would achieve better results than the competing approaches, since it can generate more accurate guesses.

As a workaround to the high memory consumption of our implementation, we introduced a probability threshold to limit the number of guesses added to the priority queue. We use the regression model outlined in the previous section to estimate the lower probability bound of a larger cracking session. This estimate

Algorithm	Iterations	Software	Hardware	Guesses/s
SHA-1	1	JtR 1.7.9-jumbo6	Intel Core i7 990X	98×10^6
SHA-1	1	oclHashcat plus-0.09	4x AMD Radeon HD 6990	155×10^8
sha512crypt	5000	JtR 1.7.9-jumbo6	Intel Core i7 990X	1800
sha512crypt	5000	JtR 1.7.9-jumbo6	ATI Radeon HD 5870	2592
sha512crypt	5000	JtR 1.7.9-jumbo6	Nvidia GTX 580	11405
bcrypt	32	JtR 1.7.9-jumbo6	Intel Core i7 990X	4960
bcrypt	32	JtR 1.7.9-jumbo6	ATI Radeon HD 5870	1745

Table 5.6: Speed of SHA-1 vs. Modern Password Hashing Algorithms [Gosney, 2012]

can be used as the probability threshold value, allowing guesses that would not be output to be discarded, freeing memory. This workaround is used in our experiments, where we predict the probability of the 3-billionth guess and use it as a threshold for the guess generation. In this way, the same amount of memory previously enough to perform only 600 million guesses becomes sufficient to generate around 3 billion. We verified that the probability of this 3-billionth guess was very close to that predicted by the regression model.

Another issue that might be hindering the performance and efficiency of our approach is that our grammar generates duplicates guesses. This occurs because passwords are ambiguous, being possibly generated by different base structures. For example, the password *onego*, can be generated by base structures producing (on, ego) or (one, go). Further study is needed to measure the impact of this issue but, as the experimental results clearly report, it is not compromising significantly the efficiency.

Chapter 6

Conclusions

In this dissertation we have contributed with the first framework for the analysis of semantics in passwords and approaches for guessing passwords more efficiently than the existing ones.

We began by demonstrating how one can analyse date patterns in a passwords sample, in Chapter 3. Then we enumerated the relevant date patterns in the largest real-world password list ever released and indicated the potential vulnerabilities caused by their occurrence through realistic guessing attack simulations. To our knowledge, this is the first systematic exploration of date patterns in passwords. In Chapter 4, we applied Natural Language Processing methods to the segmentation and classification of password samples. With such methods, we decomposed passwords into conceptually consistent parts and inferred their meaning and syntactic function. The computer-supported semantic classification of passwords is an unprecedented application of NLP. Furthermore, we are the first to demonstrate how a computational linguistic model can be used to generalize semantic categories from a password sample based on its semantic profile.

Lastly, and more importantly, in Chapter 5 we extended the state-of-the-art model of password patterns and created a model that encapsulates the semantic and syntactic patterns of passwords. With a set of experiments, we informed the impact of semantic patterns on the security provided by passwords and evaluated the expressiveness of our model against the state-of-the-art approach. The experimental results evidence that our model captures password creation patterns better than any previous model. Besides, they strongly support our hypothesis that semantic patterns represent a serious vulnerability for the password authentication scheme.

6.1 Summary of Results

Our research has achieved quantitative results that have practical value to the security community, being potentially informative to studies on password patterns and creation policies. In the following sections we synthesize the main results.

6.1.1 Date Patterns

Our visualization enabled discovery of a number of semantic patterns in the RockYou list: (a) years after 1969; (b) text words that spell out the name of a month; (c) sequences of two years; (d) the first day in each month; (e) repeated months/days and (f) holidays.

These semantic patterns have security implications—most notably, they enable the creation of language-independent password guessing dictionaries, which require no a-priori knowledge of the users. These dictionaries could be successful in an offline attack or against systems that do not implement account lock-out policies. We created one dictionary of approximately 15,000 popular dates that guessed approximately 1% of passwords from the RockYou dataset. We also found that approximately 4% of RockYou passwords were purely numeric dates, which can be guessed in a dictionary of approximately 200,000 entries. Finally, we found that over 4.5% of RockYou passwords can be characterized as dates (either purely numeric dates or dates that spell out the name of the month).

Our findings suggest it would be prudent to recommend that users do not choose a pure date numeric sequence as their password. Our findings also strongly suggest the presence of certain patterns in user choice of dates. These patterns tell us something about user preferences, which provide further insight into the password selection process.

6.1.2 General Semantic Patterns

We performed a comparison at the lexical and syntactic level between the language used in the RockYou passwords and the natural language, represented by the British National Corpus. The results showed that a large number of short words are much more likely to appear in natural language than in passwords, including prepositions and conjunctions. However, other syntactic classes that also contain predominantly short words, such as pronouns, are significantly more likely to appear in passwords. We also showed that some differences in word frequency are probably result of semantic preferences. Those findings call for a more in-depth investigation from the cultural and linguistic perspectives.

We found that a semantic model trained with a large password list, can be used in a guessing attack to crack up to 67% more passwords than the approach of Weir et al. [2009], and up to 44% more passwords than the *de facto* industry standard (a combination of wordlist and brute force strategies), given the same number of guesses. Those numbers refer to the passwords stolen from LinkedIn, a website currently ranked #14 globally [Alexa]. We also found that our semantic model can ultimately crack, given an unlimited number of guesses, approximately 30% more passwords from the MySpace leak than the approach of Weir et al. [2009], and 32% more within a 3 billion guesses constraint.

In summary, the semantic approach can crack passwords at a higher hits/guesses ratio, giving to an attacker a significant economy of time during cracking sessions against targets hashed with slows algorithms. This represents a serious security vulnerability, as efficiency is critical in a situation where passwords have been stolen and the cracker is trying to guess them before they are reset.

The semantic patterns discovered also provide insight into the passwords that people tend to choose. For example, we have found that many passwords contain concepts relating to love, sexual terms, profanity, animals, alcohol, and money. When the term “love" is used, it is most often in the context of “i love X", where X is either an objective personal pronoun (e.g., you, me, him, her) or a male name. Names, dates, and places were also popular.

6.2 Future work

Our research into the semantic patterns in passwords has raised several opportunities for future research. In this section, we discuss these under three thematic directions: improvements to the semantic approach, proactive password checking, and anthropological analysis.

6.2.1 Semantic based guessing

As described in Chapter 5, the performance of our guess generator currently prevents us from running larger experiments on a standard desktop computer. As future work, we plan to run larger experiments on a High Performance Computing (HPC) platform. This will allow us to detect when the success curve begins to flatten out, which will give us a better understanding of the practical limitations of our approach.

While our experimental scenarios are surely relevant, there are other interesting experiments we would like to perform. In particular, training our grammar with smaller password samples would serve to test if the semantics learned from small samples are capable of compromising the same targets. In this respect, we expect that the generalization of semantic categories compensate, to some extent, the reduced sample size.

We are currently not exploring the full potential of semantic generalization. By generalizing concepts, we can generate guesses containing words not seen in the training data. However, as the vocabulary of our grammar is learned from the training data, we do not generate guesses containing new words. We plan to augment our semantic approach by adding words from the WordNet-derived semantic categories which were unseen in training data. A challenge in this scenario is estimating the probabilities of the unseen words. One approach would be to assign discounted probabilities to unseen words based on their distribution in a natural language corpus such as COCA.

6.2.2 Proactive Password Checking

We suggest that the semantic grammar we built could be used for proactive password checking [Bishop and Klein, 1995]. A proactive password checker could use the PCFG to determine a password's probability, and if highly probable, it could warn the user or reject the password. Our grammar could also be used for password strength meters and suggesting password modifications as has been proposed with structural grammars Houshmand and Aggarwal [2012]. A challenge in incorporating these technologies is determining what the resulting effect is on usability and whether new password patterns emerge. User studies are required to determine the feasibility of such proactive approaches in practice.

6.2.3 Anthropological Analysis

Passwords are an interesting source for cultural studies [Andrews, 2012; Bonneau, 2010]. Their secret nature is a kind of guarantee for people that whatever they write in a password will remain private. The fact that passwords are typed several times a day [Florencio and Herley, 2007] reminds people that any thoughts expressed through them will be brought up often. In systems where changing passwords periodically is mandatory, the passwords are constantly acquiring new contents, which might well be influenced by cultural trends.

When tagged semantically, a list of passwords can be seen as a repository of thoughts with varying sentiments. Given that passwords contain people's names, company names, feelings, actions, etc., answers to questions such as "Is feeling A more frequent than B?" or "Which political view is more predominant?" can potentially feed much discussion and hypotheses. Therefore, we envision that the semantic patterns of passwords would make a rich source for anthropological investigation. In order to support this direction, the incorporation of sentiment analysis is likely required; moreover, a visualization interface would be ideal to support easy visual analysis and tasks such as comparison and filtering.

References

The British National Corpus, version 3 (BNC XML Edition), 2007. URL http://www.natcorp.ox.ac.uk/. Distributed by Oxford University Computing Services on behalf of the BNC Consortium. 32

Alexa. How popular is LinkedIn? URL http://www.alexa.com/siteinfo/linkedin.com. Accessed in July, 2013. 71

Linda Andrews. Passwords reveal your personality, January 2012. URL http://www.psychologytoday.com/articles/200201/passwords-reveal-your-personality. Accessed in July, 2013. 73

M. Bando. *101st Airborne: The Screaming Eagles in World War II*. MBI Publishing Company LLC, 2007. 1

S. Bird, E. Klein, and E. Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009. 38

Steven Bird. NLTK: The Natural Language Toolkit. In *Proc. of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics, 2006. 37

M. Bishop and D. V. Klein. Improving system security via proactive password checking. *Computers & Security* , 14(3):233 – 249, 1995. 5, 73

J. Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *Proc. of the IEEE Symp. on Security and Privacy (SP)*, pages 538–552, 2012. 1, 6, 7, 9

Joseph Bonneau. What passwords show about ourselves? *The Gates Scholar*, 7, 2010. 73

Joseph Bonneau and Ekaterina Shutova. Linguistic properties of multi-word passphrases. In *Proc. of the 16th Int. Conf. on Financial Cryptography and Data Security*, FC'12, pages 1–12. Springer-Verlag, 2012. 7

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 *Proc. of the 2012 IEEE Symp. on Security and Privacy*, SP '12, pages 553–567. IEEE Computer Society, 2012. 1

Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3 data-driven documents. *IEEE Trans. on Visualization and Computer Graphics*, 17(12):2301–2309, 2011. 17

Cynthia A. Brewer. Colorbrewer. URL `http://colorbrewer2.org/`. Accessed July 9, 2012. 15

Alan S. Brown, Elisabeth Bracken, Sandy Zoccoli, and King Douglas. Generating and remembering passwords. *Applied Cognitive Psychology*, 18(6):641–651, 2004. ISSN 1099-0720. 2, 5

John V. Carlis and Joseph a. Konstan. Interactive visualization of serial periodic data. In *Proc. of the ACM Symp. on User Interface Software and Technology - UIST '98*, pages 29–38. ACM Press, 1998. 15

Claude Castelluccia, Daniele Perito, and Durmuth Markus. Adaptive password-strength meters from markov models. In *19th Annual Network & Distributed System Security Symp. (NDSS)*. ISOC, Feb 2012. 2

Chi-Hung Chi, Chen Ding, and Andrew Lim. Word segmentation and recognition for web document framework. In *Proc. of the Eighth Int. Conf. on Information and Knowledge Management*, CIKM '99, pages 458–465. ACM, 1999. 26

Hsien-Cheng Chou, Hung-Chang Lee, Hwan-Jeu Yu, Fei-Pei Lai, Kuo-Hsuan Huang, and Chih-Wen Hsueh. Password cracking based on learned patterns from disclosed passwords. *Int. Journal of Innovative Computing, Information and Control*, 9(2), February 2013. 2, 6

Mark Davies. The Corpus of Contemporary American English: 450 million words, 1990-present, 2008-. URL `http://corpus.byu.edu/coca/`. Accessed June, 2012. 27

Dazzlepod. Dazzlepod Disclosure Project. URL `http://dazzlepod.com/disclosure/`. Accessed May, 2013. 62

C. Fellbaum. *WordNet: An Electronic Lexical Database*. Language, Speech and Communication. MIT Press, 1998. 41

Christiane Fellbaum. Wordnet. In Roberto Poli, Michael Healy, and Achilles Kameas, editors, *Theory and Applications of Ontology: Computer Applications*, pages 231–243. Springer, 2010. 40

Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proc. of the 16th Int. Conf. on World Wide Web*, WWW '07, pages 657–666. ACM, 2007. 73

GeoNames. GeoNames Data. URL `http://download.geonames.org/export/dump/`. Accessed October, 2012. 28

Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. Contextual dependencies in unsupervised word segmentation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, ACL-44, pages 673–680, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. 26

Jeremi Gosney. The final word on the linkedin leak, June 2012. URL `http://securitynirvana.blogspot.ca/2012/06/final-word-on-linkedin-leak.html`. Accessed in July, 2013. 68

Peter D Grünwald, In Jae Myung, and Mark A Pitt. *Advances in minimum description length: Theory and applications*. MIT press, 2005. 43

S. Houshmand and S. Aggarwal. Building Better Passwords Using Probabilistic Techniques. In *Proc. of the 28th Annual Computer Security Applications Conf.*, pages 109–118, 2012. 73

Alfred Inselberg. The plane with parallel coordinates. *The Visual Computer*, 1(2): 69–91, 1985. ISSN 0178-2789. 35

Markus Jakobsson and Mayank Dhiman. The benefits of understanding passwords. In *Mobile Authentication*, SpringerBriefs in Computer Science, pages 5–24. Springer New York, 2013. 2, 6, 26, 28, 62

Sanjeet Khaitan, Arumay Das, Sandeep Gain, and Adithi Sampath. Data-driven compound splitting method for english compounds in domain names. In *Proc. of the 18th ACM Conf. on Information and knowledge management*, CIKM '09, pages 207–214. ACM, 2009. 26

Hang Li and Naoki Abe. Generalizing case frames using a thesaurus and the mdl principle. *Comput. Linguist.*, 24(2):217–244, June 1998. ISSN 0891-2017. 43, 46

Magnumripper. Community Enhanced Version (Bleeding Jumbo) of John The Ripper 1.8. URL `https://github.com/magnumripper/JohnTheRipper/tree/c63b0187eab690ba92093a7d6182752527ecd26a`. Accessed in May, 2013. 61

C.D.A. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999. 37, 54

Fitz-Simons T. Mintz, D. and M. Wayland. Tracking air quality trends with SAS/GRAPH. In *Proc. of the 22nd Annual SAS User Group Int. Conf.*, pages 807–812, 1997. 16

Christof Monz and Maarten Rijke. Shallow morphological analysis in monolingual information retrieval for Dutch, German, and Italian. In Carol Peters, Martin Braschler, Julio Gonzalo, and Michael Kluck, editors, *Evaluation of Cross-Language Information Retrieval Systems*, volume 2406 of *Lecture Notes in Computer Science*, pages 262–277. Springer Berlin Heidelberg, 2002. 26

Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *Proc. of the 12th ACM Conf. on Computer and communications security*, CCS '05, pages 364–372. ACM, 2005. 2

Openwall. John the Ripper password cracker. URL `http://www.openwall.com/john/`. 57

Outpost9. Wordlists. URL `http://www.outpost9.com/`. Accessed November, 2011. 28

W.W.R. Paton, F.W. Walbank, and C. Habicht. *The Histories*, volume 3 of *The Histories*. Harvard University Press, 2012. 1

Ashwini Rao, Birendra Jha, and Gananand Kini. Effect of grammar on security of long passwords. In *Proc. of the 3rd ACM Conf. on Data and Application Security and Privacy*, CODASPY '13, pages 317–324, New York, NY, USA, 2013. ACM. 6, 7, 54

Paul Rayson and Roger Garside. Comparing corpora using frequency profiling. In *Proc. of the workshop on Comparing corpora - Volume 9*, WCC '00, pages 1–6. Association for Computational Linguistics, 2000. 32

Bruce L. Riddle, Murray S. Miron, and Judith A. Semo. Passwords in use in a university timesharing environment. *Computers & Security*, 8(7):569–579, 1989. ISSN 0167-4048. 2, 5

Richard Shay, Saranga Komanduri, Patrick Gage Kelley, Pedro Giovanni Leon, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Encountering stronger password requirements: user attitudes and behaviors. In *Proc. of the Sixth Symp. on Usable Privacy and Security*, SOUPS '10, pages 2:1–2:20. ACM, 2010. 2

SSA. Popular Baby Names. U.S. Social Security Administration. URL `http://www.ssa.gov/oact/babynames/limits.html`. Accessed March, 2013. 27

Christian Tominski. Enhanced interactive spiral display. In *Proc. of the Annual SIGRAD Conf., Special Theme: Interactivity*, pages 53–56, 1999. 15

Blase Ur, Saranga Komanduri, Richard Shay, Stephanos Matsumoto, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Patrick Gage Kelley, Michelle L Mazurek, and Timothy Vidas. Poster: The Art of Password Creation. In *Proc. of the IEEE Symp. on Security and Privacy*, May 2013. 6

Fernanda B. Viégas, Martin Wattenberg, and Jonathan Feinberg. Participatory visualization with Wordle. *IEEE Trans. on Visualization and Computer Graphics*, 15 (6):1137–1144, Nov./Dec. 2009. 16

Andreas Wagner. Enriching a lexical semantic net with selectional preferences by means of statistical corpus analysis. In Steffen Staab, Alexander Maedche, Claire Nedellec, and Peter M. Wiemer-Hastings, editors, *Proc. of the First Workshop on Ontology Learning OL*, volume 31 of *CEUR Workshop Proc.* CEUR-WS.org, August 2000. 45, 46

Matt Weir. Reusable security. URL `https://sites.google.com/site/reusablesec/`. Accessed May, 2013. 62

Matt Weir, Sudhir Aggarwal, Breno De Medeiros, and Bill Glodek. Password cracking using probabilistic context-free grammars. *2009 30th IEEE Symp. on Security and Privacy*, pages 391–405, 2009. 2, 3, 5, 6, 32, 53, 54, 55, 57, 59, 71

Matt Weir, Sudhir Aggarwal, Michael Collins, and Henry Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proc. of the 17th ACM Conf. on Computer and Communications Security*, CCS '10, pages 162–175. ACM, 2010. 6

Rick Wicklin and Robert Allison. Congestion in the sky: Visualising domestic airline traffic with SAS. ASA Statistical Computing and Graphics Data Expo 2009, 2009. 16