On the Semantic Patterns of Passwords and their Security Impact

Rafael Veras, Christopher Collins, and Julie Thorpe
University of Ontario Institute of Technology
Ontario, Canada
{rafael.verasguimaraes, christopher.collins, julie.thorpe}@uoit.ca

***Abstract*—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. Researchers have only touched the surface of patterns in password creation, with the semantics of passwords remaining largely unexplored, leaving a gap in our understanding of their characteristics and, consequently, their security. In this paper, we begin to fill this gap by employing Natural Language Processing techniques to extract and leverage understanding of semantic patterns in passwords. 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 approximately 67% more passwords from the LinkedIn leak and 32% more passwords from the MySpace leak.**

I. Introduction

Passwords are the first line of defence in safeguarding information for many networked systems. Despite many proposals for alternative forms of user authentication, they are not likely to be replaced in the near future, as the alternatives are still immature or economically infeasible. Moreover, passwords offer advantages not always matched by other schemes, including usability and easy recovery from loss [1].

Even after half a century of password use in computing, we still do not have a deep understanding of how people create their passwords. Such an understanding is desirable, as it can inform more realistic estimates of password security, which are essential to inform password policies, proactive password checkers, and password strength meters. It has been increasingly acknowledged that the key to solving the security problems of passwords lies on a better structural understanding of passwords [2], but the community's knowledge is still restricted to superficial patterns. The literature features a wealth of investigations of distribution of characters [3, 4] and types of mangling patterns present in passwords [5, 6]. Metrics of password strength consider mainly length, presence of non-alphabetic characters and character casing [7]; however, deeper patterns, in particular the ones concerning the *meaning* of passwords, remain largely unexplored.

NDSS '14, 23-26 February 2014, San Diego, CA, USA

Historically, the semantics of passwords have been investigated through research instruments of social sciences, such as surveys, with small groups of participants [8, 9]. 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. 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 perform deeper password analyses. These leaked password lists provide the largest samples of real-world passwords to date, offering an enormous opportunity for empirically grounded research.

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 paper aims to address the following questions: Are there preferences in the choice of concepts used in passwords? If so, what are 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? In this paper, we explore semantic patterns in the large list of passwords (over 32 million) stolen and made publicly available in 2009 from the RockYou website.

Our contributions are as follows. (1) We demonstrate how Natural Language Processing (NLP) algorithms can be used to segment, classify, and generalize semantic categories from passwords. 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. (2) We describe the most common semantic patterns that emerge from the RockYou data set. (3) Building upon previous work on Probabilistic Context-Free Grammars, we develop and describe a grammar that captures structural, syntactic, and semantic patterns of a list of passwords. (4) We evaluate the security impact of the semantic patterns found by using our grammar to generate guesses in off-line attack scenarios against other leaked password lists (LinkedIn and MySpace). The results show that our approach can guess 67%

more LinkedIn passwords in the first 3 billion guesses and 32% more MySpace passwords than the state-of-the-art approach, proposed by Weir et al. [5]. The high success rate cracking passwords from sources different than the training data indicate the generality of our approach and of the semantic patterns found. Our results indicate that these semantic patterns update our understanding of password security and we suggest that our grammar could be used to improve proactive password checking and password strength meters.

The paper is structured as follows: in Section II we summarize the literature on password patterns; in Section III, we present an approach and results for segmenting passwords, classifying password segments by POS and semantic category, and abstracting semantic categories; in Section IV, we build a Probabilistic Context-Free Grammar based on semantic and syntactic tags and present experimental off-line password cracking results; finally, in Section VI, we discuss the implications of our findings and future work.

II. Related Work

Research in the field of psychology has employed qualitative research instruments to investigate the semantics of passwords. Brown et al. [9] 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. [8] 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 [10].

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. [5] 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 [11]. 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 [12, 6]. 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 [13]. Bonneau [14] proposes new metrics based on guessing resistance for password strength.

Jakobsson and Dhiman [2] 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. [5]), 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. [15] 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 [14] suggesting that the strength of Yahoo! passwords is similar to the RockYou passwords.

Rao et al. [12] 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 [16] 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 English language, represented in reference corpora, such as the British National Corpus and the Google

TABLE I. REFERENCE CORPORA DETAILED.

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

Web Corpus. However, Ur et al. [15] show that passwords sets are more similar to each other than to a corpus of English, suggesting the existence of a grammar of passwords. In the present paper, we present a model which, independent of passwords following English *grammar*, is capable of capturing their semantic and syntactic essence (the grammar of passwords) and posing a threat against unforeseen targets. In other words, as we assume that passwords are composed of English words, natural language influences password segmentation and tagging, but it does not constrain the grammar learning.

In summary, the aforementioned studies inform extensively how structural patterns are used and their impact on security; in addition, a few studies have shown 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.

III. PARSING AND SEMANTIC CLASSIFICATION

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. In this approach, passwords of all forms and lengths are broken into parts and classified semantically; thus, segmentation is a fundamental step, discussed in Section III-A. We describe our POS tagging methods in III-B, semantic classification and generalization in Section III-C, and resulting semantic categories in Section III-D.

A. 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 [17, 18, 19]. The first application of word breaking in passwords appeared not until recently, by Jakobsson and Dhiman [2]. Their algorithm takes a compilation of general and specialized dictionaries as input and uses a measure of coverage as the primary criterion for selection of candidate segmentations. In addition to coverage, we make use of higher order N-gram frequencies to disambiguate segmentations with equal coverage.

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 I). 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 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.

TABLE II. SOURCE CORPORA DETAILED.

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

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.)[20]. 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; 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 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:

- **Names** Derived from a dataset of the U.S. Social Security Administration (SSA) [21]. 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 [22] 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, 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 [23] 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.

As previously mentioned, word boundaries are not explicit

TABLE III. CANDIDATE SEGMENTATION FOR PASSWORD *Anyonebarks98*

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

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 [2], 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^ $\longrightarrow$ 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 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 the example above there is not much room for ambiguity. In Table III, 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 III 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

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
```

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

```
1: procedure BESTNGRAMSCORE(C)
2:     score ← 0
3:     l ← LENGTH(C)
4:
5:     if l = 1 then
6:         score ← UNIGRAMPROBABILITY(C)
7:     else if l = 2 then
8:         score ← BIGRAMPROBABILITY(C)
9:     else if l = 3 then
10:        score ← TRIGRAMPROBABILITY(C)
11:    end if
12:
13:    if score = 0 then
14:        for i ← 1, 3 do
15:            a ← BESTNGRAMSCORE(C[: i])
16:            b ← BESTNGRAMSCORE(C[i :])
17:            tempScore ← a * b
18:            if tempScore > score then
19:                score ← tempScore
20:            end if
21:        end for
22:    end if
23: end procedure
```

probability of an N-gram $w_1...w_N$ is given by:

$$P(w_1...w_N) = \frac{f(w_1...w_N)}{f(K_N)} \quad (1)$$

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 being 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 scores used in line 9 of Algorithm 1. The score 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.

1) Analysis of Segmentation Results: Now that we have the capability of extracting words from passwords, a question of relevance to password cracking is which words are more common in the RockYou list? A weakness of the the approach of Weir et al. [5] 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

TABLE IV. MOST FREQUENT WORDS FROM THE SOURCE CORPORA FOUND IN THE ROCKYOU LIST.

Rank	Word	% Relative Frequency	Rank	Word	% Relative Frequency
1	a	3.24	51	mi	0.13
2	i	3.01	52	go	0.13
3	love	1.39	53	ka	0.13
4	me	0.63	54	star	0.13
5	in	0.53	55	blue	0.13
6	you	0.49	56	red	0.13
7	baby	0.49	57	big	0.13
8	my	0.44	58	dog	0.13
9	to	0.40	59	al	0.13
10	an	0.38	60	so	0.13
11	is	0.36	61	boo	0.12
12	girl	0.34	62	st	0.12
13	it	0.34	63	us	0.12
14	as	0.28	64	ku	0.11
15	la	0.28	65	le	0.11
16	te	0.27	66	jo	0.11
17	sexy	0.26	67	abc	0.11
18	on	0.26	68	may	0.11
19	am	0.25	69	bear	0.11
20	be	0.24	70	daniel	0.11
21	man	0.24	71	cute	0.11
22	password	0.24	72	cat	0.10
23	the	0.23	73	of	0.10
24	luv	0.23	74	monkey	0.10
25	boy	0.22	75	lu	0.10
26	no	0.22	76	we	0.10
27	amo	0.21	77	da	0.10
28	rock	0.21	78	ever	0.10
29	angel	0.20	79	en	0.10
30	ca	0.20	80	ty	0.10
31	or	0.20	81	jesus	0.10
32	na	0.20	82	chris	0.10
33	el	0.19	83	bitch	0.09
34	and	0.19	84	john	0.09
35	lil	0.18	85	ho	0.09
36	do	0.17	86	one	0.09
37	ha	0.17	87	bo	0.09
38	de	0.16	88	li	0.09
39	princess	0.16	89	by	0.09
40	life	0.16	90	ah	0.09
41	lo	0.15	91	ya	0.09
42	he	0.15	92	tu	0.09
43	ma	0.15	93	gurl	0.09
44	ko	0.14	94	za	0.09
45	at	0.14	95	cool	0.09
46	ta	0.14	96	dr	0.09
47	fuck	0.14	97	just	0.09
48	hot	0.14	98	po	0.09
49	yo	0.14	99	sweet	0.09
50	pink	0.14	100	lover	0.09

highest probability order. Table IV shows the 100 top segments in the RockYou list.

2) Limitations: As with any statistical algorithm, our segmentation algorithm is affected by noise in the training data. This noise is due to our assumption that all passwords contain English words. Therefore, random strings such as "auhopjlif", or foreign language passwords such as "vaipaysandu", will be segmented incorrectly. Such incorrect segmentations will most frequently recognize shorter two-letter words (many of which come from the names dictionary). However, these incorrect segmentations only affect a small percentage of passwords: of the passwords that contain at least two alphabetic characters, only 12% contain two-character words from the names dictionary.

If one intends to use our approach in contexts that require high accuracy, for example, the study of semantics in passwords from a cultural perspective, it would also be desirable to improve our named entity disambiguation, which is somewhat arbitrary. Another limitation of our segmentation is that if new terms begin to be used in passwords (e.g., new company names or slang), they will only be captured once included as part of the source corpora.

B. 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) [24] 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.

TABLE V. 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.

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

TABLE VI. DISTRIBUTION OF THE POS TAGGED SEGMENTS FROM ROCKYOU BY SYNTACTIC CATEGORY.

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

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 [25]. In Table V, 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 (see Section III-C1) 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 [26]. The unigram tagger, as expected, is the one that tags the majority of words (see Table V).

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 of noun and verb respectively. The Table VI shows the resulting distribution of segments by POS.

C. Semantic Classification

After segmenting and POS tagging the passwords, we finally met the requirements to perform a good semantic classification. At this point, we can represent each password

TABLE VII. SEMANTIC CATEGORIES OF GAP SEGMENTS.

Category	Example
number	123
char	LoL
special	*¡—:-)
num+special	0:-3
all_mixed	o/\o5

by an array of 2-tuples $S = [(s_1, t_1), ..., (s_n, t_n)]$, where s_i is a segment and 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. Next, in Section III-C2, we describe how low-level semantic concepts can be abstracted, allowing us to, later on, characterize semantic patterns in a more general way.

1) WordNet-based classification: WordNet 3.0 [27] 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, words are called *lemmas* and concepts are called *synsets*.

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 III-C2). In fact, sentiment analysis would be a suitable way to generalize adjectives, but it is out of scope in this paper. 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 VII, using regular expressions. Next, we test if s is a proper noun. WordNet does not provide comprehensive 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 [28]. 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".

[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 III-C2).

Algorithm 3 Classify segments by semantic category

```
1: procedure ClassifySemantic(S)
2:     K ← []
3:     for all (s, t) ∈ S do
4:         c ← null
5:         if s is a gap segment then
6:             classify by gap category
7:         else if t is a proper noun tag then
8:             c ← source corpus name
9:         else if t is either a verb or a noun tag then
10:            s ← Stem(s)
11:            synsets ← LookupWordnet(s)
12:            if Length(synsets) > 0 then
13:                c ← synsets[0].name
14:            end if
15:        end if
16:        Append(K, (s, t, c))
17:    end for
18:    return K
19: end procedure
```

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 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?

To automatically answer that question, we make use of the *tree cut model* by Li and Abe [29]. 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" [30]. 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). $L_{par}(M)$ represents the size of the cut and is inversely proportional to the abstraction level. $L_{dat}(M)$ measures how far the tree cut model M is from the data, and is proportional to the abstraction level. Technically, the algorithm of Li and Abe [29] minimizes the sum of $L_{par}(M)$ and $L_{dat}(M)$, referred to as model description length $L_{mod}(M)$:

$$L_{mod}(M) = L_{par}(M) + L_{dat}(M) \tag{2}$$

The parameter description length is calculated as in Equation 3, where k is the number of nodes (classes) in the cut and $|S|$ is the sample size:

$$L_{par}(M) = \frac{k}{2} \times log|S| \tag{3}$$

The data description length is given by Equation 4:

$$L_{dat}(M) = -\sum_{s \in S} log\hat{P}(s) \tag{4}$$

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:

$$\hat{P}(s) = \frac{1}{|C|} \times \hat{P}(C) \tag{5}$$

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

$$\hat{P}(C) = \frac{f(C)}{|S|} \tag{6}$$

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

Given the aforementioned probabilities, Li and Abe [29] describe a recursive algorithm to efficiently calculate the tree cut that minimizes the model description length $L_{mod}(M)$.

3) 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 [31]:

1) Duplicate synsets having 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 [31] reports that the algorithm of Li and Abe [29] "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 3 and 4. 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 [31] 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 2:

$$L_{par}(M) + W\left(\frac{log|S|}{|S|}\right) L_{dat}(M) \quad (C > 0) \tag{7}$$

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 1, 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

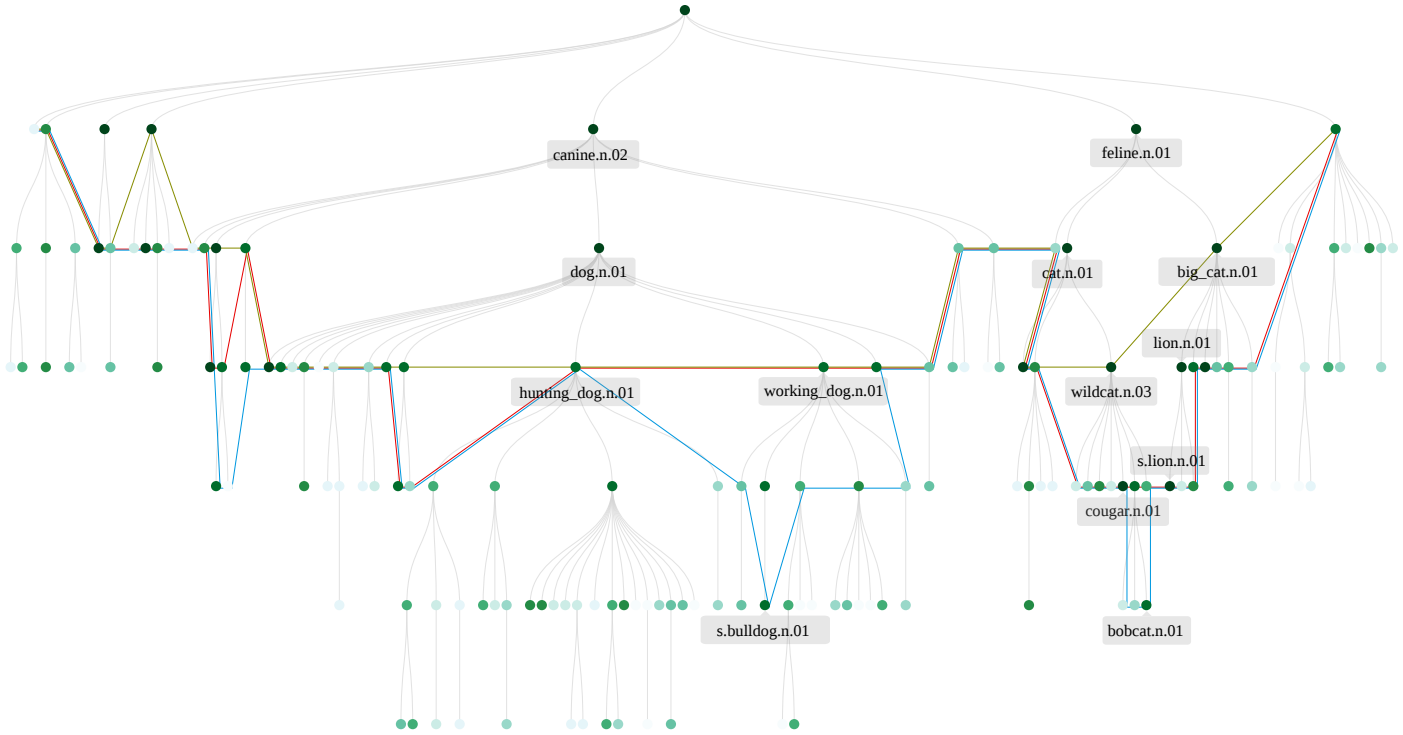

Fig. 1. Subtree of WordNet rooted at the node carnivore.n.01 and the tree cuts resulting of the following weighting values: 1,000 (golden), 5,000 (red) and 10,000 (blue). Higher weight values generate less general cuts.

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.

D. Resulting Semantic Categories

In Table VIII, 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 IX, 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 comprehensive 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

TABLE VIII. SAMPLE OF PASSWORDS WITH SEGMENTS CLASSIFIED BY SEMANTICS. THE SEMANTIC TAG COLUMN SHOWS THE FINAL SEMANTIC CATEGORY OF A SEGMENT, AFTER SYNSET GENERALIZATION.

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 IX. MOST PROBABLE SEMANTIC CATEGORIES IN THE ROCKYOU LIST

	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

two letter words (e.g., *polonium* "Po" and *multiple sclerosis* "MS"). Of particular interest is *tellurium*, 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").

IV. SEMANTIC GUESS GENERATOR

To validate whether the semantic categories we found have any security impact, we create 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. [5], 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.

A. 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 [25]. PCFGs were used in passwords first by Weir et al. [5] 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. [12] 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 syntactic 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. The grammar can be proven to be *regular*, since no rule has more than one nonterminal in its right-hand side, and each of these nonterminals is at the same end of the right-hand side.

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 *love.v.01.VVD* we have the concatenation of a semantic (*love.v.01*) and a POS category (*VVD*). This symbol

TABLE X. SAMPLE GRAMMAR LEARNED FROM THE TRAINING SET *iloveyou, ihatedthem3, football3*

	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

should derive only the verbs semantically categorized as *love* that are inflected in the past tense, e.g., "loved" and "adored". In this way, we increase the descriptive power of the grammar.

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

The base structures 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 X, we show an example PCFG learned from the set of passwords $\{iloveyou2, ihatedthem3, football3\}$.

Consistent with the nomenclature adopted by Weir et al. [5], we call the structures derived from the start variable *base structures*, i.e., right-hand side of all N_1 rules. Table XI lists the most probable base structures learned from the 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 XII shows a comparison between the PCFGs generated by our approach and the approach of Weir et al. [5], both trained with the RockYou list.

B. 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) [32]. 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* $\rightarrow$ *p4ssw0rd*. In this case, a wordlist 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

TABLE XI. 50 MOST PROBABLE BASE STRUCTURES OF THE GRAMMAR TRAINED WITH THE ROCKYOU LIST.

	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 XII. COMPARISON BETWEEN GRAMMARS GENERATED BY THE SEMANTIC AND WEIR APPROACHES TRAINED WITH THE ROCKYOU LIST, AND A COMPARABLE BRUTE FORCE ATTACK. ⋆ SEE SECTION V-C FOR DESCRIPTION OF APPROXIMATION METHODS AND BRUTE FORCE COMPARISON.

					MySpace attack	
Approach	Base structures	Non-terminals	Terminals	Terminal Struct.	Guessed passwords (%)	Approximate # 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 (until same percentage guessed as Semantic)					91.76	3.2×10^{43}

(for the attacker). As previously mentioned, Weir et al. [5] 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 V, 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. [5] 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. [5] (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.

Algorithm 4 Generates guesses in highest probability order

```
1: procedure GENERATEGUESSES(G)
2:         ▷ Initialize priority queue with most probable derivation of each base structure
3:     queue ← initialize priority queue
4:     for all G.base_structures do
5:         guess ← initialize guess
6:         guess.terminals ← most probable terminal values for the base struct.
7:         guess.pivot ← 0
8:         guess.p ← calculate probability of the guess
9:         INSERT(queue, guess)
10:    end for
11:
12:    c ← POP(queue)
13:    while c ≠ NULL do                          ▷ Generate password guesses
14:        OUTPUT(c)                                  ▷ Output current guess
15:        for i ← c.pivot, LEN(c.terminals) do  ▷ Derive lower probability guesses from the same base structure
16:            new ← initialize new guess
17:            new.terminals ← DECREMENT(c.terminals, G, i)   ▷ Replace c.terminals[i] by the next lower probability terminal at i
18:            if new.terminals ≠ NULL then
19:                new.p ← calculate probability
20:                new.pivot ← i
21:                INSERT(queue, new)
22:            end if
23:        end for
24:        c ← POP(queue)
25:    end while
26:
27: end procedure
```

1) Custom Mangling: The semantic guess generator only generates guesses containing lowercase word segments; gap segments, on the other hand, 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 XIII 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.

TABLE XIII. CASE STATISTICS OF WORD SEGMENTS EXTRACTED FROM ROCKYOU PASSWORDS.

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	

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.

C. 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.

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 XII, 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.

2) 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.

3) 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 XII) 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.

V. Experiments

We use the community enhanced version (bleeding jumbo) of John The Ripper 1.8[33]. 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 (semantic_no_rules)
2) Semantic approach with custom mangling rules (semantic_custom_rules)
3) Semantic approach with default JtR's mangling rules (semantic_jtr_rules)
4) Weir approach (weir)
5) Wordlist with JtR's default rules, followed by incremental mode[2] (john_wordlist_inc)

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 IV-B1 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 [34], trained on the same RockYou dataset as the semantic approaches, and also the input dictionary used in their paper (dic-0294). In method 5, we use JtR's wordlist mode with the passwords.txt wordlist (2,151,220 unique values) available at Dazzlepod [35]. According to Dazzlepod, this list has a success rate of 40% using JtR's mangling rules against the Lulzsec collection of hashes (final release).

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

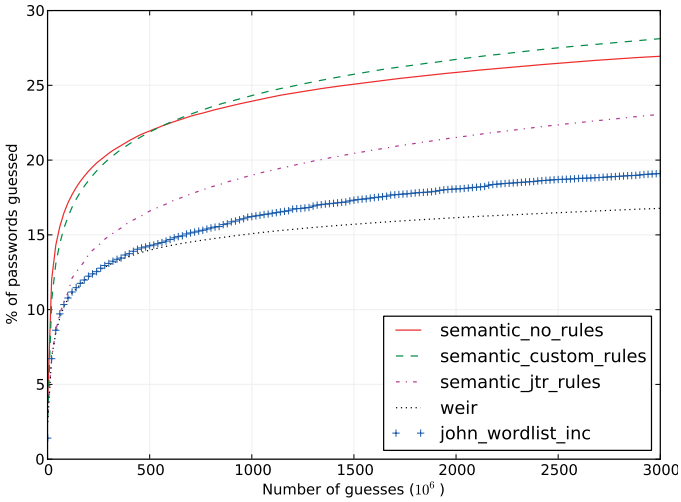

Fig. 2. Results of Experiment 1. The three variations of the semantic approach perform better than the competing approaches.

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 [2].

A. 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 digits 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 2). 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

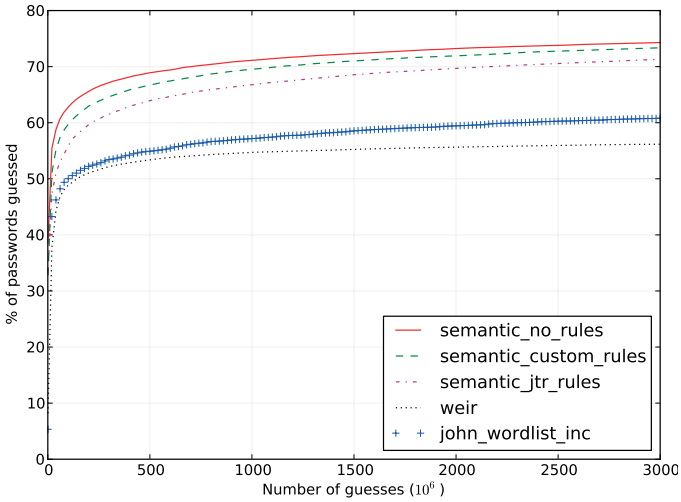

Fig. 3. Results of Experiment 2. The best semantic condition guesses approximately 18% more passwords than the Weir approach.

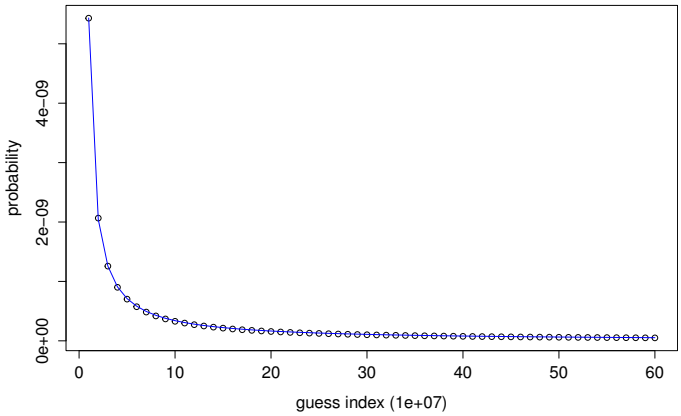

Fig. 4. Nonlinear regression model of the guess probabilities of the semantic approach.

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.

B. 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.

C. 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. It is possible to compute this measure with a simple grammar recognizer, with the constraint that the 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 XII 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; 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. For example, using guess-number calculators, Kelley et al. [11] takes days to compute guess numbers for the Weir approach using 64-node distributed computing clusters. We can obtain an estimate of this measure by fitting a nonlinear regression model to a relatively small sample of the guess probabilities. Figure 4 shows a sample of the first 600 million guesses, reduced to every 10 millionth guess. The blue curve represents an exponential decay model (Equation 8) with the β parameters fitted to the data using the Gauss-Newton method. With the inverse function $g(y)$ we can estimate 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 XII. Likewise, we can determine the approximate number of guesses of a cracking session using the Weir grammar. Table XII 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.

$$f(x, (\beta_1, \beta_2, \beta_3)) = \frac{\exp(-\beta_1)}{\beta_2 + \beta_3 x} \tag{8}$$

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

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, where the hashing time is the bottleneck, turning the cost of hash comparisons much higher.

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 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.

VI. Conclusions

In this paper we have contributed the first framework for the analysis of semantics in passwords. The semantic patterns discovered 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, food, and money. When the term "love" is used, it is most often in the context of "i love X", where X is most often an objective personal pronoun (e.g., you, me, him, her) or a male name. Names, dates, and places were also popular.

We have also presented approaches for guessing passwords more efficiently than existing approaches. We extended the state-of-the-art model of structural password patterns and created a grammar that encapsulates the semantic and syntactic patterns of passwords. With a set of experiments, we demonstrate 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.

We found that our semantic approach can guess over 67% more passwords from the LinkedIn leak than the approach of Weir et al. [5] within 3 billion attempts. We also found that our semantic approach can correctly guess, given an unlimited number of attempts, approximately 50% more passwords from the MySpace leak than the approach of Weir et al. [5], and 32% more within a 3 billion guesses constraint. These experimental results provide evidence that our model captures password creation patterns better than any previous model, and suggest that semantic patterns can reduce the security of passwords in the absence of proactive password checking.

While there are some ethical concerns regarding the use of stolen passwords, we used lists that are publicly available, implying that our use of the passwords does not increase signficantly the risk that the users are exposed to, especially when considering that we do not make use of login or personal information of the users. Moreover, as criminal groups may be using this data for malicious purposes, ignoring their existence can leave the security community unaware of the kinds of attacks that are possible.

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

Semantic Based Guessing. We are currently not exploring the full potential of semantic generalization. By generalizing concepts, we could generate guesses containing words not seen in the training data. For example, if "coffee" in "lovecoffee" is generalized as "drink", the guesser could output "lovejuice" even though "juice" has never been seen in the training sample. 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. While this would make for a more complete approach, we do not expect significant improvement, as the vocabulary learned from the RockYou dataset already includes the most probable words. If we plug in the COCA corpus, approximately 200,000 words would be added to the vocabulary, but these words only represent 4% of the COCA frequency space (meaning they are unlikely to appear in general English).

Proactive Password Checking. We suggest that the semantic grammar we built could be used for proactive password checking [10]. 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 [36]. 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.

Anthropological analysis. Passwords are an interesting

source for cultural studies [37, 38]. 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 [39] 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 hypothesis. Therefore, we envision that the semantic patterns of passwords would make a rich source for anthropological investigation.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and GRAND NCE. We are grateful to Jeffrey Hickson for his assistance and to the Italian Cultural Centre of Durham for supporting the first author with a scholarship.

References

[1] J. Bonneau, C. Herley, P. C. v. Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Proc. of the IEEE Symp. on Security and Privacy*, ser. SP '12. IEEE Computer Society, 2012, pp. 553–567.

[2] M. Jakobsson and M. Dhiman, "The benefits of understanding passwords," in *Mobile Authentication*, ser. SpringerBriefs in Computer Science. Springer New York, 2013, pp. 5–24.

[3] A. Narayanan and V. Shmatikov, "Fast dictionary attacks on passwords using time-space tradeoff," in *Proc. of the 12th ACM Conf. on Computer and Communications Security*, ser. CCS '05. ACM, 2005, pp. 364–372.

[4] C. Castelluccia, D. Perito, and D. Markus, "Adaptive password-strength meters from markov models," in *19th Annual Network & Distributed System Security Symp. (NDSS)*. ISOC, Feb 2012.

[5] M. Weir, S. Aggarwal, B. D. Medeiros, and B. Glodek, "Password cracking using probabilistic context-free grammars," in *Proc. of the IEEE Symp. on Security and Privacy*, 2009, pp. 391–405.

[6] H.-C. Chou, H.-C. Lee, H.-J. Yu, F.-P. Lai, K.-H. Huang, and C.-W. Hsueh, "Password cracking based on learned patterns from disclosed passwords," *Int. Journal of Innovative Computing, Information and Control*, vol. 9, no. 2, Feb. 2013.

[7] R. Shay, S. Komanduri, P. G. Kelley, P. G. Leon, M. L. Mazurek, L. Bauer, N. Christin, and L. F. Cranor, "Encountering stronger password requirements: user attitudes and behaviors," in *Proc. of the Sixth Symp. on Usable Privacy and Security*, ser. SOUPS '10. ACM, 2010, pp. 2:1–2:20.

[8] B. L. Riddle, M. S. Miron, and J. A. Semo, "Passwords in use in a university timesharing environment," *Computers & Security*, vol. 8, no. 7, pp. 569–579, 1989.

[9] A. S. Brown, E. Bracken, S. Zoccoli, and K. Douglas, "Generating and remembering passwords," *Applied Cognitive Psychology*, vol. 18, no. 6, pp. 641–651, 2004.

[10] M. Bishop and D. V. Klein, "Improving system security via proactive password checking," *Computers & Security* , vol. 14, no. 3, pp. 233 – 249, 1995.

[11] P. Kelley, S. Komanduri, M. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. Cranor, and J. Lopez, "Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms," in *Proc. of the IEEE Symp. on Security and Privacy*, 2012, pp. 523–537.

[12] A. Rao, B. Jha, and G. Kini, "Effect of grammar on security of long passwords," in *Proc. of the 3rd ACM Conf. on Data and Application Security and Privacy*, ser. CODASPY '13. New York, NY, USA: ACM, 2013, pp. 317–324.

[13] M. Weir, S. Aggarwal, M. Collins, and H. 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*, ser. CCS '10. ACM, 2010, pp. 162–175.

[14] J. Bonneau, "The science of guessing: Analyzing an anonymized corpus of 70 million passwords," in *Proc. of the IEEE Symp. on Security and Privacy*, 2012, pp. 538–552.

[15] B. Ur, S. Komanduri, R. Shay, S. Matsumoto, L. Bauer, N. Christin, L. F. Cranor, P. G. Kelley, M. L. Mazurek, and T. Vidas, "Poster: The Art of Password Creation," in *Proc. of the IEEE Symp. on Security and Privacy*, May 2013.

[16] J. Bonneau and E. Shutova, "Linguistic properties of multi-word passphrases," in *Proc. of the 16th Int. Conf. on Financial Cryptography and Data Security*, ser. FC'12. Springer-Verlag, 2012, pp. 1–12.

[17] K. Wang, C. Thrasher, and B.-J. P. Hsu, "Web scale NLP: A case study on URL word breaking," in *Proc. of the 20th Int. Conf. on World Wide Web*, ser. WWW '11. ACM, 2011, pp. 357–366.

[18] C.-H. Chi, C. Ding, and A. Lim, "Word segmentation and recognition for web document framework," in *Proc. of the Eighth Int. Conf. on Information and Knowledge Management*, ser. CIKM '99. ACM, 1999, pp. 458–465.

[19] S. Khaitan, A. Das, S. Gain, and A. Sampath, "Data-driven compound splitting method for english compounds in domain names," in *Proc. of the 18th ACM Conf. on Information and Knowledge Management*, ser. CIKM '09. ACM, 2009, pp. 207–214.

[20] M. Davies, "The Corpus of Contemporary American English: 450 million words, 1990-present," 2008- , Accessed June, 2012. [Online]. Available: http://corpus.byu.edu/coca/

[21] SSA, "Popular Baby Names. U.S. Social Security Administration," Accessed March, 2013. [Online]. Available: http://www.ssa.gov/oact/babynames/limits.html

[22] GeoNames, "GeoNames Data," accessed October, 2012. [Online]. Available: http://download.geonames.org/export/dump/

[23] Outpost9, "Wordlists," accessed November, 2011. [Online]. Available: http://www.outpost9.com/

[24] S. Bird, "NLTK: The Natural Language Toolkit," in *Proc. of the COLING/ACL on Interactive presentation sessions*.

Association for Computational Linguistics, 2006, pp. 69–72.
[25] C. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.
[26] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. O'Reilly Media, 2009.
[27] C. Fellbaum, "Wordnet," in *Theory and Applications of Ontology: Computer Applications*, R. Poli, M. Healy, and A. Kameas, Eds. Springer, 2010, pp. 231–243.
[28] ——, *WordNet: An Electronic Lexical Database*, ser. Language, Speech and Communication. MIT Press, 1998.
[29] H. Li and N. Abe, "Generalizing case frames using a thesaurus and the MDL principle," *Computational Linguistics*, vol. 24, no. 2, pp. 217–244, Jun. 1998.
[30] P. D. Grnwald, I. J. Myung, and M. A. Pitt, *Advances in minimum description length: Theory and applications*. MIT press, 2005.
[31] A. Wagner, "Enriching a lexical semantic net with selectional preferences by means of statistical corpus analysis," in *Proc. of the First Workshop on Ontology Learning OL*, ser. CEUR Workshop Proc., vol. 31. CEUR-WS.org, Aug. 2000.
[32] Openwall, "John the Ripper password cracker." [Online]. Available: http://www.openwall.com/john/
[33] Magnumripper, "Community Enhanced Version (Bleeding Jumbo) of John The Ripper 1.8," accessed in May, 2013. [Online]. Available: https://github.com/magnumripper/JohnTheRipper/tree/c63b0187eab690ba92093a7d6182752527ecd26a
[34] M. Weir, "Reusable security," accessed May, 2013. [Online]. Available: https://sites.google.com/site/reusablesec/
[35] Dazzlepod, "Dazzlepod Disclosure Project," accessed May, 2013. [Online]. Available: http://dazzlepod.com/disclosure/
[36] S. Houshmand and S. Aggarwal, "Building Better Passwords Using Probabilistic Techniques," in *Proc. of the 28th Annual Computer Security Applications Conf.*, 2012, pp. 109–118.
[37] J. Bonneau, "What passwords show about ourselves?" *The Gates Scholar*, vol. 7, 2010.
[38] L. Andrews. (2012, Jan.) Passwords reveal your personality. Accessed in July, 2013. [Online]. Available: http://www.psychologytoday.com/articles/200201/passwords-reveal-your-personality
[39] D. Florencio and C. Herley, "A large-scale study of web password habits," in *Proc. of the 16th Int. Conf. on World Wide Web*, ser. WWW '07. ACM, 2007, pp. 657–666.