Passwords Decay, Words Endure: Secure and Re-usable Multiple Password Mnemonics

Umut Topkara   Mikhail J. Atallah [*]   Mercan Topkara
Department of Computer Sciences
Purdue University
West Lafayette, IN, 47906, USA
utopkara,mkarahan,mja@cs.purdue.edu

ABSTRACT

Research on password authentication systems has repeatedly shown that people choose weak passwords because of the difficulty of remembering random passwords. Moreover, users with multiple passwords for unrelated activities tend to choose almost similar passwords for all of them. Many password schemes have been proposed to alleviate this problem, but they either require modification to the password entry and processing infrastructure (e.g., graphical passwords) or they require the user to have some trusted computing power (e.g., smartcard-like portable devices, browser plugins, etc). We propose a scheme that is applicable to any existing system without any modification, as it does not require any form of involvement from the service provider (e.g., bank, brokerage). Nor does it require the user to have any computing device at hand (not even a calculator). Our approach consists of generating a mnemonic sentence that helps the users remember a multiplicity of truly random passwords, which are independently selected. The scheme is such that changes to passwords do not necessitate a change in the mnemonic sentence that the user memorizes. Hence, passwords can be changed without any additional burden on the memory of the user, thereby increasing the system's security. An adversary who breaks one of the passwords encoded in the mnemonic sentence does not gain information about the other passwords. A key idea is to split a password in two parts: One part is written down on a paper (helper card), another part is encoded in the mnemonic sentence. Both of these two parts are required for successfully reproducing the password, and the password reconstruction from these two parts is done using only simple table lookups. Passwords' renewal requires only the re-generation of the helper card. Our scheme resolves the apparent contradictory requirements from most password policies: That the password should be random, and that it should be memorized and never written down. This makes possible passwords that are more secure against an adversary who illicitly gains access to the password file, as a dictionary attack is now unlikely to succeed (the attacker now needs to carry out a more daunting brute force enumerative attack). Even if the adversary somehow obtains the helper card, it gets quantifiably limited information about the passwords of the user (so the helper card may be lost or stolen without disaster immediately striking the user). We quantify the time period required for this adversary to successfully crack the password.

[*]Portions of this work were supported by Grants IIS-0325345, IIS-0219560, IIS-0312357, and IIS-0242421 from the National Science Foundation, and by sponsors of the Center for Education and Research in Information Assurance and Security.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'07 March 11-15, 2007, Seoul, Korea
Copyright 2007 ACM 1-59593-480-4 /07/0003 ...$5.00.

Categories and Subject Descriptors

H [**Information Systems**]: Models and Principles—*Security*

General Terms

Security, Design

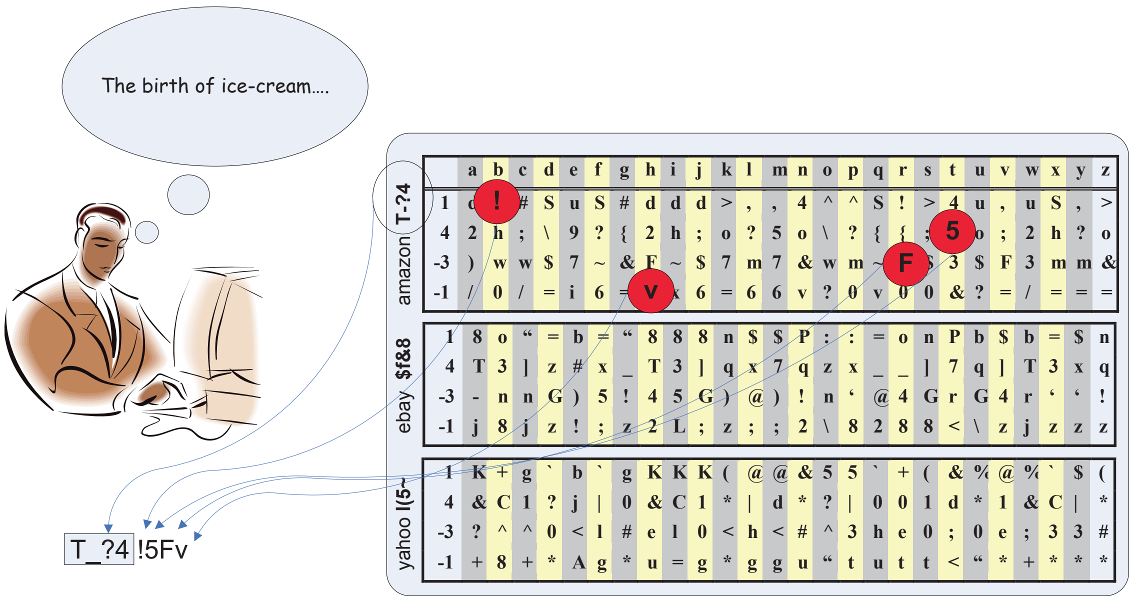
Keywords

Authentication, Passwords, Mnemonic Sentence, Usability, Natural Language Processing

1. INTRODUCTION

Speaking at the opening of the 2005 AusCERT conference, Microsoft's Jesper Johansson said that "companies should not ban employees from writing down their passwords, because such bans force people to use the same weak password on many systems." In [14] Schneier provocatively suggests to users, "write down your passwords", in agreement with Johansson, and adds that people "are much more secure if they choose a password too complicated to remember and then write it down" , since "we are all good at securing a small piece of paper". This is because the probability of the above mentioned sheet of paper compromising the password is much lower than the probability that a weak password will be exploited by an adversary.

In the present paper we show that one can actually write something that is (to the user) a complete description of the password, yet is of little value if the sheet of paper is lost or stolen or otherwise compromised. Our approach is



		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
amazon T-?4	1	d	!	#	S	u	S	#	d	d	d	>	,	,	4	^	^	S	!	>	4	u	,	u	S	,	>
	4	2	h	;	\	9	?	{	2	h	;	o	?	5	o	\	?	{	{	;	5	o	;	2	h	?	o
	-3	)	w	w	$	7	~	&	F	~	$	7	m	7	&	w	m	~	F	$	3	$	F	3	m	m	&
	-1	/	0	/	=	i	6	=	v	x	6	=	6	6	v	?	0	v	0	0	&	?	=	/	=	=	=
ebay $f&8	1	8	o	“	=	b	=	“	8	8	8	n	$	$	P	:	:	=	o	n	P	b	$	b	=	$	n
	4	T	3	]	z	#	x	_	T	3	]	q	x	7	q	z	x	_	_	]	7	q	]	T	3	x	q
	-3	-	n	n	G	)	5	!	4	5	G	)	@	)	!	n	‘	@	4	G	r	G	4	r	‘	‘	!
	-1	j	8	j	z	!	;	z	2	L	;	z	;	;	2	\	8	2	8	8	<	\	z	j	z	z	z
yahoo l(5~	1	K	+	g	`	b	`	g	K	K	K	(	@	@	&	5	5	`	+	(	&	%	@	%	`	$	(
	4	&	C	1	?	j	|	0	&	C	1	*	|	d	*	?	|	0	0	1	d	*	1	&	C	|	*
	-3	?	^	^	0	<	l	#	e	l	0	<	h	<	#	^	3	h	e	0	;	0	e	;	3	3	#
	-1	+	8	+	*	A	g	*	u	=	g	*	g	g	u	“	t	u	t	t	<	“	*	+	*	*	*

Figure 1: Helper card and a password lookup: The user reconstructs the desired password (e.g., for "amazon") in two steps: i) copying part of the password written in clear text (e.g., "T-?4") ii) looking up the rest of the password from the table by using the letters of the mnemonic word (e.g., "birth" in this case) which are indexed by the letter positions given in the first column (e.g., the first, the fourth, the third from last and the last letters of the mnemonic word). Note that the helper card does not have any markings, the highlights in the figure are inserted for the sake of narrative clarity. Every word that is longer than 4 letters are mnemonic words, and are used in the same order to decode passwords (e.g. "birth" encodes the first password, "ice-cream" encodes the second password on the card)

to complement the written sheet of paper (henceforth referred to as the helper card) with an unwritten mnemonic sentence, thereby making what is written on the helper card more cryptic in case it is lost or stolen. In other words, both card and mnemonic sentence are needed for reproducing the passwords; as the mnemonic lies between the user's ears, it provides the security commonly categorized as "something you know", whereas the helper card provides the security categorized as "something you have". We also describe a mnemonic sentence generation mechanism such that an adversary who breaks one of the passwords does not gain additional information about the other passwords even with access to the helper card. Overall our approach does not reduce the security of the existing password authentication infrastructure, it provides a more secure alternative for the users who write down their random passwords.

Before plunging into the details, we briefly mention the challenges we had to overcome, and give a glimpse of our approach to them.

- Using the same mnemonic for multiple passwords, in such a way that one password's compromise does not translate into another password's compromise. The highly structured nature of natural language text, and its known statistical properties, stood in the way of achieving this goal. In a nutshell, we "decoupled" passwords from each other by using only a subset of English: Rich enough to prevent an adversary's enumeration of all possible mnemonics, yet restricted enough that it does not have the above-mentioned drawbacks. For example, the existence of a password that can be encoded with the word "ink" does not imply a more likely existence of another password that can be encoded with the words "paper" or "pen". This requirement turns out to be surprisingly easy to implement, by independently selecting the mnemonic words from each other and later using these words to construct the mnemonic sentence.

- Achieving easy user reconstruction of a password from the helper card and the mnemonic, without any access to a portable computing device (not even a simple calculator). In our system the user makes one simple table lookup per encoded password letter while reconstructing the password. Refer to Figure 1 for an example of the helper card table used in this kind of lookup.

- Generating mnemonic sentences that have both desired properties (of being memorable and having the required encoding capacity). Here is an example: "The birth of ice-cream: Why and how we sneeze at midnight." We achieved this by carrying out judicious word-substitutions on a sentence drawn from the news headlines. We used newspaper headlines since they summarize a story, thus form a connected discourse, which was experimentally shown [8] to be much easier to learn than same amount of nonsense. We will explain how later, for now we merely mention that we do this, guided by the distance between words as reported in WordNet [4].

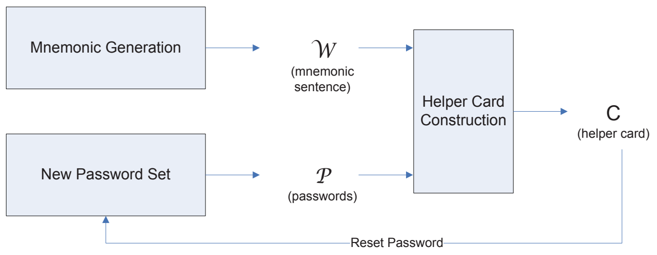


Figure 2: Mnemonic System: The computational parts.

We will first give a usage scenario in Section 2 leaving out the discussion of the details that are transparent to the user. Then in Section 3 we will discuss the details that pertain to the construction of the helper card, the encoding of random strings using a limited vocabulary of words and the generation of natural language sentences. Finally, in Section 4, we include a survey of related work in password authentication and with analysis of their impact on our work as well as their similarities, and differences from our work.

2. MNEMONIC USAGE

In this section we describe a typical scenario of mnemonic password usage.

The life cycle of a mnemonic password (Figure 2) starts with selection of a mnemonic sentence by the user. The system generates a set of mnemonic sentences, from which the user picks one sentence that is easier to remember and memorizes this sentence. A mnemonic sentence $\mathcal{W}$ contains k mnemonic words $W_1, \ldots, W_k$, each of which are of length $\geq m$.

Later, the user is assigned a set of l random passwords, $\mathcal{P} = \{P_1, \ldots, P_l\}$, by third parties (e.g., banks, brokerages, online shops, school or work accounts, etc.) or the user picks some or all of the passwords. Each of these passwords are composed of $\leq n$ password symbols, $P_i = (p_{i,1}, \ldots, p_{i,n})$.

The system then constructs a lookup table by using $\mathcal{P}$ and $\mathcal{W}$, and this table is printed on a card (the size of a credit card for convenience) as in Figure 1. We refer to this lookup table as *helper card*. The user keeps the helper card secure, preferably along with credit cards in the user's wallet.

When the user wants to remember $P_i = (p_{i,1}, \ldots, p_{i,n})$, the user does the following:

1. finds the i^{th} table in the helper card. This table has two parts, a string G_i and a table T_i (e.g., G_1 is `T-?4` in Figure 1).

2. types G_i, which is the first $n - m$ symbols of the password $(p_{i,1}, \ldots, p_{i,n-m})$ printed in clear on the helper (possibly $G_i = \emptyset$, if desired).

3. retrieves W_i from memory (e.g., W_1 is the word "birth" in Figure 1).

4. derives the remaining symbols of the password $p_{i,n-m+j}$ by i) finds $w_{i,j}$ by retrieving the mnemonic letter pointed by the index value stored in the first column of j^{th} row ii)looking up $w_{i,j}$ in the j^{th} row of T_i (e.g., the first encoded symbol of the "amazon" password is decoded using "b", the 1^{st} letter of the word "birth").

Password Changes: The passwords that are assigned to the user can change, and the user need not change the mnemonic; only the table of the helper card that corresponds to the changed password needs to be reconstructed.

Lost Helper: As we noted earlier, the user needs to keep the helper card secure. However, the helper card may be lost or stolen - without disaster immediately striking the user: the user has a limited but known amount of time before a password in the helper card can be compromised. This secure time period is ensured by the encoding of the password on the helper card; the adversary will likely have to perform a known, large number of brute force login trials before successfully compromising the system. As soon as it is known that the helper card is lost or has been copied by an adversary, the user needs to change all the passwords and generate a new helper card to remember them, however the user does not need to change the mnemonic.

Compromised Password: A very important property of password mnemonic sentences are their resilience against an adversary that has compromised one of the passwords. It is possible that one of the passwords is compromised by an adversary without the information of the helper, e.g., by catching user's keystrokes. Since the passwords are independently generated from each other the rest of the passwords cannot be compromised. However, in the case which the adversary also has access to the helper, the compromise of P_i will leak information about W_i. In a normal English sentence, such knowledge would reveal other words, e.g., the word "ink" is usually used in the same sentence with words "paper" or "pen". This would have disastrous consequences since the adversary would learn the passwords encoded by those words for free. In order to rule out such situations, the mnemonic is constructed in a way that W_i will not leak information about any other part of the mnemonic; hence the adversary gains no additional information about the rest of the mnemonic or the rest of the passwords by compromising P_i (more on this in Section 3).

3. BEHIND THE SCENES

This section discusses details of the system that pertain to the parts that do not involve user interaction: The computation of helper cards' content, the generation of candidate mnemonics.

3.1 Passwords and Helpers

We first discuss the relationship between the input (i.e., the password) and the outputs (i.e., the mnemonic and the helper card) in the computational parts of the mnemonic system. As noted earlier, the requirements of our multi-password mnemonic system for helper card and mnemonic sentence are: i) both the mnemonic sentence and the helper card are needed to reconstruct the password, ii) this reconstruction of the password can be performed by the user without computational aids, using table lookups.

We now make a brief digression to discuss why the simple "secret splitting" idea from cryptography is not suitable for our purpose. Recall from [13] that to split a secret password $\mathcal{P}$, into two you (i)pick a random R_s , (ii)create the two parts as R_s (a random seed), and $E(\mathcal{P}, R_s)$. There are several problems with this approach in our application:

1. The R_s (the mnemonic sentence) can not be a random string as it has to be memorable to the user.

2. The decryption of $\mathcal{P}$ is not possible for a human to perform without a computational aid.

Replacing in the above $E(\mathcal{P}, R_s)$ by $\mathcal{P} \oplus R_s$, where $\oplus$ is the bitwise exclusive-or operator, makes it moderately doable by a computer-less human, but has the severe drawback that compromise of one password automatically reveals R_s as well as all the other passwords. Our system uses a different R_s for each password to alleviate this problem.

Note that, $\mathcal{P} \oplus R_s$ would not leak any information about either of R_s or P without the knowledge of the other, hence $\mathcal{P} \oplus R_s$ can be made public. However, R_s, which needs to be kept secret, is still a random string, and is not easier to memorize than the password $\mathcal{P}$ itself. We achieve memorability of R_s by a process called "mnemonic generation", which is the conversion of R_s into an easy to remember human language sentence (e.g., English). The details of mnemonic generation is given in Section 3.3; in summary, our system first finds a set of words that encode each R_s (every P will have a different one), then generates sentences by picking one word from each of these sets with the help of natural language generation techniques.

Mnemonic generation (see Figure 3) is the task of creating a candidate set of easy to remember sentences that can encode a given random seed string. Mnemonic generation concludes with the user's selection of one of the candidate mnemonic sentences for remembering as a mnemonic.

Let R_s be the random seed that generated the candidate mnemonic set and let the mnemonic chosen by the user be $\mathcal{W}$.

The computation of the mnemonic sentence from the R_s uses an encoding of the R_s as words in a vocabulary; the details of this encoding will be discussed in Section 3.3. Let us refer to $d()$ as the corresponding decoding function that can derive R_s from $\mathcal{W}$, such that $d(\mathcal{W}) = R_s$, and refer to $e() = d^{-1}()$ as the encoding function. Note that R_s is different from a password, it is a seed that is used internally by the mnemonic system to produce the candidate set of mnemonic sentences that encode R_s, and is also transparent to the user.

After the mnemonic generation and selection of $\mathcal{W}$ as the mnemonic, a password P is needed to proceed to the helper construction task. Since the mnemonic is selected without the knowledge of password, an adversary who learns the password at this step will not have any information about the mnemonic. Also if an adversary learns the mnemonic, this information will not reveal the password.

Let R_h be the result of $R_s \oplus P$. Since both R_s and P are random strings R_h will also be random. One can compute P using R_s and R_h from $P = R_s \oplus R_h$, or $P = d(\mathcal{W}) \oplus R_h$. In order to be able to perform this operation by hand we choose our decoding function such that $p_i = d(w_i) \oplus r_{h,i}$ as well as $r_{s,i} = d(w_i)$ hold: P can be reconstructed by decoding it symbol by symbol.

Helper construction is the task of building a lookup table that can perform $lookup(i, w_i) = d(w_i) \oplus r_{h,i}$ by looking the i^{th} mnemonic letter, w_i, in the i^{th} row of the table. The password is the concatenation of the successive table lookups for the mnemonic letters in the mnemonic, $P = (lookup(1, w_1), \ldots, lookup(n, w_n))$. An example of helper card lookup was given in Section 2. We will discuss the details of the encoding function in Section 3.2.

3.2 Encoding with Word Sets

Mnemonic generation (see Figure 3) is the task of creating sentences that encode a set of random seed strings. In this subsection we will discuss how to find an encoding function and in the next subsection we will describe the generation of mnemonic sentences from word sets.

Let V be the vocabulary of words that our mnemonic sentences will use, $W_i \in V$, and let $Q = \{0, 1, \ldots, |Q| - 1\}$ be an alphabet, let S be the i.i.d. set of all strings of length m defined on Q, $S = \{x : x \in Q^m\}$, and $R_s \in S$.

In order to efficiently use the memory of users, we should find an *onto* decoding function that maps V to the largest possible random string set (i.e., maximize the size of S, $|S|$) in a given number of lookup operations, m.

Let N be a sequence of letter indices, such that $length(N) = m$. Let $d_N : V \to S$ be a desired decoding function. d_N can be computed from individual letters of its input as indexed by N using m mapping functions $d_{N,i} : \{a, \ldots, z\} \to Q$, such that :

$\forall v \in V, d_N(v) = (d_{N,1}(v_{N[1]}), \ldots, d_{N,m}(v_{N[m]}))$.

In our implementation we performed a heuristic search to obtain a decoding function that maximizes $|S|$ for a given $|Q|$. See Table 1 for statistics of the results of our implementation for different $|Q|$ values when N is a subsequence of $(1, 2, 3, 4, -4, -3, -2, -1)$. Note that negative values in elements of N refer to positions counting from the last letter backwards. The resulting $|S|$ is very small compared to the size of the `dict` file, this is because we have limited our R_s to be i.i.d. in Q^m. It is possible to achieve $|S|$ closer to 26^m, which is the maximum number of strings of length m that can be constructed using English alphabet, if we allowed $Prob(R_s) = 0$ for some $R_s \in Q^m$. We have excluded this case from the scope of this paper, for the sake of simplicity.

We then used $lookup(i, v) = (d_{N,i}(v_{N[i]}) + z_{N,i}) \mod |Q'|$, where $(z_{N,i} + w_i) \mod |Q'| = p_i$ and Q' is the alphabet from which P is derived (e.g., Q' is the set of printable ASCII characters in most UNIX systems). A helper card constructed this way decodes the i^{th} mnemonic letter, w_i (which is the $N[i]^{th}$ letter of the mnemonic word), into the i^{th} password letter, p_i. Note that, the same $d_{N,i}$ is used for every mnemonic word in Figure 1, only $z_{N,i}$ values are different, therefore corresponding rows of different helper tables share the same mapping pattern. For instance the mnemonic letters "h,i,j" always map to the same password letter in the 1^{st} row.

The time required to break a password when the helper card is lost, depends on the size of domain of the encoding function we use, $|S| = |Q^m|$. If t is the time that it will take the adversary to try 1 password for login, on the average it will take $\frac{|Q^m| \times t}{2}$ units of time to break one password using the helper card.

3.3 Mnemonic Generation

Recall that our system starts with generating a set of random seeds which will be used to conceal passwords in helper cards. Let $\mathcal{R}_s = (R_{s,1}, \ldots, R_{s,k})$ be the list of k random seeds that were generated for the user, which can be encoded in a mnemonic sentence that has the capacity to remind up to k passwords. In Section 3.2 we have described an encoding function that maps random strings into sets of vocabulary words, $\mathbf{W}_i = e(R_{s,i})$.

In this section we will describe how our system generates sentences by picking one word from each word set $\mathbf{W}_i$ with

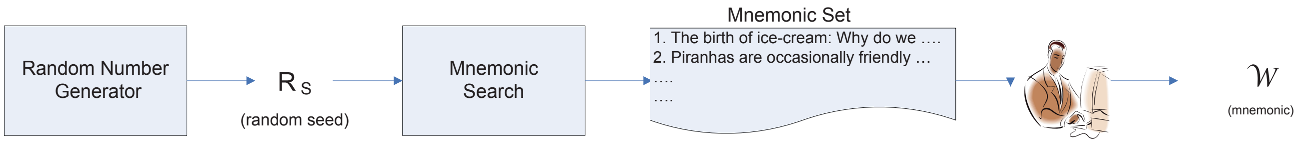


Figure 3: Mnemonic Generation and Selection: Initial selection of the mnemonic sentence involves the users, since memorability of a sentence depends on individual experiences and tastes.

Size of Alphabet $\|Q\|$	Number of Lookups m	Number of Passwords $\|Q^m\|$	Mnemonic Letter Positions N
3	7	2187	1,2,3,4,-4,-3,-1
4	6	4096	1,3,4,-4,-3,-1
5	5	3125	1,3,-4,-3,-1
6	4	1296	2,4,-3,-1
7	4	2401	2,4,-3,-1
8	4	4096	1,4,-3,-1
9	4	6561	1,4,-3,-1
10	3	1000	1,3,-3

Table 1: Decoding function: The results of our heuristic search algorithm to find decoding functions. The input vocabulary was 353056 alphabetic strings from the Linux `dict` file that are longer than 3 letters. The index sequence from which N is computed was $(1, 2, 3, 4, -4, -3, -2, -1)$, and the program was run for alphabet sizes ranging from 3 to 10. The program found an encoding that can map elements from a domain set of 6561 random strings to strings in `dict` file, and has a decoding table that requires 4 lookups.

the help of natural language generation techniques.

In order to achieve memorability of the generated sentences we use a large corpus of newspaper headlines as template sentences: while doing the best effort to maintain the original meaning of them, we modify them through word substitutions to contain a subsequence $(W_1, \ldots, W_k)$ where $W_i \in \mathbf{W}_i$. Sentences that our system generates are usually easy to memorize, since the templates that we use have been curated by newspaper editors to summarize stories thus form a connected discourse, which was experimentally shown [8] to be much easier to learn than same amount of nonsense.

Let L be a template sentence from our corpus, where $L_s = (l_1, \ldots, l_k)$ are the words that our system uses for word substitutions. In order to preserve the attractiveness of the original sentences, we employ a heuristic search for substitutions which minimizes the maximum value of $\delta(l_i, W_i)$, where δ is a function that quantifies the similarity of the input words.

In our implementation we used the `path` function that is part of the `WordNet::Similarity` [10] library as our similarity function δ. `path` returns the length of the shortest path of "is-a" relationships between two concepts in WordNet [4].

Let us use an example to demonstrate the search for a good substitution word, where l_1 is the template word "newspaper" and $\mathbf{W}_1 = \{mirror, letter\}$ are the words that encode our desired random string. The word "letter" would be preferred by our system to substitute the word "newspaper", because $\delta(newspaper, mirror)$ is 8 and $\delta(newspaper, letter)$ is 6 ("newspaper#n#1" denotes the 1st noun sense of the word "newspaper"):

- Shortest path between "newspaper" and "mirror": "newspaper#n#1 press#n#3 print_media#n#1 medium#n#1 instrumentality#n#3 device#n#1 reflector#n#1 mirror#n#1"
- Shortest path between "newspaper" and "letter": "newspaper#n#3 product#n#2 creation#n#2 representation#n#2 document#n#2 letter#n#1"

As we have noted before, if the adversary compromises the helper card as well as the password P_i, the set of mnemonic words from which W_i is picked can be easily revealed. This would have disastrous consequences about the security of the rest of the words in the mnemonic sentence, since natural languages have a regular structure. In our approach, $\mathcal{W}$ is derived from $\mathcal{R}_s$ which is randomly generated. Even if the adversary learns one of the sets $\mathbf{W}_i$ used in the mnemonic sentence, this will not leak information about the rest of the mnemonic words.

Our approach resembles synonym substitution based sentence generation used by several security applications [17, 2, 6, 15]; we include a review of these applications in Section 4. In these applications a given sentence is converted to a new sentence by changing its words with one of their synonyms.

4. RELATED WORK

Morris and Thompson have studied the vulnerabilities of the initial UNIX password scheme in 1979, and found out that 86% of the passwords they have collected were easily cracked in a day during an exhaustive but intelligent search of the password space [9]. They identified several improvements to UNIX password system, including the use of DES encryption and the enforcement of longer password, so that an automated attack would suffer from a longer running time.

In 1989, Feldmeier and Karn revisit the password authentication problem in UNIX [3]. This time they are armed with faster *crypt* function implementation that is used is encrypting passwords in password files, as well as faster and cheaper computers. They suggest facilitating of shadow password file as opposed to public password file, as a first measure to defend the passwords. However they point out that the ultimate password protection is picking passwords that are hard to guess by an automated brute-force attacker. They recognize that the usability is a barrier against forcing users to remember long random passwords. Giving Shannon's results for per-character entropy of English, they suggest that a 5-10 word English sentence will embody an entropy required in an 8 character password. The users of these *passphrase*s will *XOR* each 8 character block of their passphrase and use the result as their random password.

The passphrase system suggested by Feldmeier and Karn could be implemented easily. However the number of key strokes required at every authentication would render this passphrase hard to use for frequent authentication. For this reason they suggest distributed authentication systems such as Kerberos tickets to be used in conjunction with the passphrase system for decreasing the overall number of keystrokes and the additional burden of remembering multiple password phrases as a usability fix. Moreover the authors suggest frequent changes of passwords to increase the difficulty of cracking passwords. They identify the complexity of the users' password as the major security requirement.

In 2004, Yan et al. conducted a controlled experiment to compare the effects of giving three alternative forms of advice about password selection [18]. This trial involved 400 first-year students at Cambridge University. 100 of these students were given the classical instructions on how to pick a password: " Your password should be at least seven characters long and contain at least one non-letter." 100 of them were given a paper that has the letters A-Z and integers 1-9 repeatedly on it, and they were asked to close their eyes and randomly pick symbols from this letter to generate a random password, later they were asked to write it down and carry that paper with them until they memorize the password. The other 100 students were given an instruction sheet that explains how to generate mnemonic passwords. The last 100 were not given any instructions at all. Yan et al. performed several well-known attacks on these passwords, as well as analyzing the statistical properties of these passwords (e.g. length) and the frequency of the users' need for a password reset.

This study challenged several widely accepted beliefs about security and memorability of passwords:

1. It is confirmed that users have difficulty remembering random passwords (Many students continued to carry the written copy of their password for a long time, 4.8 weeks on the average.).

2. The results also confirmed that mnemonic passwords are indeed harder for an adversary to guess than naively selected passwords.

3. Contrary to the popular belief that random passwords are better than mnemonic passwords, this experiment showed that mnemonic passwords are just as strong as the random passwords.

4. This study also showed that it is *not* harder to remember mnemonic passwords, which are just as memorable as naively selected passwords.

5. Another interesting result of this study is that it is *not* possible to gain a significant improvement in security by just educating users to use random or mnemonic passwords; both random passwords and mnemonic passwords suffered from a *non-compliance* rate of about 10% (including both too-short passwords and passwords not chosen according to the instructions).

The lack of compliance of users can also be explained with the *lack of incentive*. User incentive can be created by refusing non-compliant passwords or by providing an easy to use authentication scheme, or by bundling small incentives from several systems into a large incentive. In our system, we provide encouraging incentives to the users for complying with our instructions (e.g., keeping the helper card secure). Some of the user incentives we provide are as follows:

- providing the users with a facility that they can also use secure passwords for their personal (non-work) online accounts, since one mnemonic can encode multiple passwords. This way the users will have a good reason to keep the helper cards secure.

- allowing the users to pick the mnemonic sentences that fit to their taste, hence are memorable to them, from a set of mnemonic sentences generated by the system.

- providing a password reset mechanism that does not require the reseting of the mnemonic.

The reward is high in a system that is usable for multiple passwords. A uniform interface (e.g. helper card) will attract users, because repetition across different systems will bring ease of use, and familiarity.

Kuo et al. performed another user study to check whether mnemonic passwords are vulnerable to dictionary attacks when the dictionary is specifically generated using the popular phrases available on the Internet such as advertising slogans, children's nursery rhymes and songs, movie quotes, song lyrics, etc. [7]. Even though 65% of the mnemonic phrases picked by the users can be found by a Google search, only 4% of the mnemonic passwords, that the users generated, was cracked by a brute force attack using a special dictionary generated by deriving mnemonic passwords from phrases grabbed from popular Internet sites.

A user study by Gaw and Felten showed that users have a tendency to "re-use" their passwords across multiple accounts [5]. They have interviewed with 49 users. The majority of these users had three or fewer passwords and passwords were re-used twice. The top reason mentioned for reusing a password was "easier to remember". Results of this study agrees with our motivation to design a system that encodes multiple passwords with one mnemonic phrase.

In 2005, Jeyaraman and Topkara proposed a system that automatically generates memorable mnemonics for a given random password [6]. This system is based on searching for a mnemonic that encodes the given password in a precomputed database of mnemonics, which is generated by taking sentences from a text-corpus and producing syntactic and semantic variations of these sentences. In order to produce the variants of corpus sentences, they used linguistic transformations (e.g., synonym substitutions). This system was able to generate mnemonics for 80.5% of 6-character passwords and 62.7% of 7-character passwords containing lower-case characters (a-z), even when the text-corpus size is extremely small (1000 sentences). Even though this is a very important first step in developing secure mnemonic password systems, it was limited in providing following items:

- Generating mnemonics that can encode symbols that are not lower-case characters

- Encoding multiple passwords with one mnemonic sentence

- Encoding long passwords

There are several other studies that use synonym substitution as a technique for information hiding into natural language text. Natural language information hiding systems, that are based on modifying a cover document, mainly rewrite the document using linguistic transformations. T-Lex is one of the first implemented systems that embed hidden information by synonym substitution on a cover document [17]. A given message is embedded into the cover text using a pre-generated synonym set database as follows. First, the letters of the message text are Huffman coded according to English letter frequencies. The Huffman coded message is embedded into message carrying words in the cover text by replacing them with their synonyms in the synonym database of T-Lex. The synonym sets in this database are interpreted as a mixed-radix digit encoding according to the set elements' alphabetical order.

In [15], Topkara et al. introduced a natural language watermarking system that embeds the watermark message (e.g, copyright notice) into a given document using robust synonym substitution. Compared to naive synonym substitution, robust synonym substitution introduces ambiguities in order to make it harder for the watermark embedding modifications to be undone. This scheme first selects a subset of words from a given dictionary, and later assigns colors to these words, where the colors are used to represent the role of the word in embedding the watermark, such as "carries 0", "carries 1" or "non-encoding". A secret key, shared between watermark embedder and watermark reader, is used for both the subset selection and the color assignment of the words. When there are many alternatives to carry out a substitution on a word (i.e. more than one synonym carries the required embedding bit), they prioritize these alternatives according to a quantitative resilience criterion and favor more ambiguous alternatives. For example, if the original sentence is "he survived without water and food for 3 days", this watermarking system rewrites it as "he went without water and food for 3 days", since the word "go" is more ambiguous (i.e. has many different meanings). The approximately meaning-preserving changes, that are done in the direction of more ambiguity, are harder for the adversary to resolve with automated disambiguation, however, a human reader can quickly disambiguate the meaning when reading the marked text. This system exploits the well-established fact in the natural language processing community, that humans are much better than computers at disambiguation [12].

The study of Reverse Turing Tests in [11] suggests a method to ensure that it will take a pre-determined time to break a password with an automated attack if the adversary has to use the login system. This is achieved by judicious use of challenges by the system that require computational capabilities of a human (e.g., CAPTCHAs [16]). Our system can be complemented with a similar system such that the adversary is even further limited in the time that it is required to break a password in the helper card. Moreover, if such a system is in use, the number of table lookups can be adjusted to fit the time that the users need to before the adversary can break the password.

In another work, Bergmair et al. proposes a Human Interactive Proof system which exploits the fact that even though machines can not disambiguate senses (i.e. meanings) of words, humans can do disambiguation highly accurately [2]. In this system, the users are shown several "challenges", where each challenge is composed of several sentences generated by synonym substitution of a word in the template sentence. The word that is substituted is replaced with its synonyms from the same sense in some of the challenge sentences and its synonyms from other senses in the remaining sentences. The user is asked to mark the sentences that carry the same meaning. It is expected that they will not pick the sentences that have the substitutions from mis-matching senses. The system keeps the dictionary, that is used to find the synonyms, secret. This dictionary is augmented with new synonyms by asking the user about randomly replaced words: if the users frequently mark the sentence that has the random replacement as an equal meaning sentence, then this random word is added to the synonym set of the word that it replaced in the original sentence. An example of a challenge is as follows, the user is asked to mark the sentences that are meaningful replacement of others in the following set:

- The speech has to move through several more drafts.
- The speech has to run through several more drafts.
- The speech has to go through several more drafts.
- The speech has to impress through several more drafts.
- The speech has to strike through several more drafts.

To the best of authors' knowledge the only password scheme that uses a table lookup technique was employed by a bank in 1992. The customers of this bank were suggested to conceal their PIN by writing it down on a special piece of square cardboard that is designed to be kept along with the ATM card. The cardboard presented a table of 4 rows and 10 columns, each column corresponding to a digit (0-9). The customer was asked to pick a four-letter word and write the letters of the word, in consecutive order, to the columns that correspond to digits of their PIN (i.e. first letter goes to the first row and the column of first digit of the PIN). After this step, the empty table cells should be filled with random letters. Anderson described this scheme in [1], where he also mentioned that this cardboard was increasing vulnerability of the system, since the adversary's job is now reduced to finding the four-letter English words (which are not many) in the consecutive rows of the table and trying the corresponding PIN numbers.

5. CONCLUDING REMARKS

We believe that natural language processing is a good technology to use in password mnemonics, as it offers the multiple advantages that were mentioned over other approaches. This work is an important first step in the direction of a single-mnemonic for multiple passwords with both good security and good usability properties.

The "best use" we recommend for a deployment of our scheme, would include a policy that sets the passwords' expiration period (i.e., requiring renewal) to roughly coincide with the time period required by an adversary, who has misappropriated the helper card, to carry out an attack using that card: This will ensure that even users who do not realize that their helper card was stolen, are not unduly exposed to password compromise. We also note that, in addition to the password renewal not requiring a change of mnemonic, the renewal process does not burden the users with the task of choosing a random new password (users are notoriously poor judges of the quality of randomness): A quality random-password generator can be used instead. The users do have a say, but where it really matters to them: At the initial mnemonic-creation time, for the choice of which candidate mnemonic they will have to memorize (this has to involve the user, as what is memorable for a person is highly subjective).

While it is clear that a password that is compromised has to be changed, and that it requires no change of mnemonic, we stress that it is imperative that the mnemonic be changed if *both* helper card and password(s) are compromised (the latter possibly through shoulder-surfing, spyware, phishing, etc).

As a future work we would like to perform a user study to adjust the parameters of our system to fit the comfort of users. The current mnemonic generation system involves the user only after a candidate set of mnemonics are found. It is also possible to improve the user experience by incorporating the user's preferences and interests to the topic of mnemonic sentences that are generated.

6. ACKNOWLEDGMENT

Authors would like to thank four anonymous referees for their helpful comments and suggestions.

7. REFERENCES

[1] R. Anderson. Why cryptosystems fail. In *CCS '93: Proceedings of the 1st ACM Conference on Computer and communications Security*, pages 215–227, New York, NY, USA, 1993. ACM Press.

[2] R. Bergmair and S. Katzenbeisser. Towards human interactive proofs in the text-domain. In *Proceedings of the 7th Information Security Conference*, volume 3225, pages 257–267. Springer Verlag, September, 2004.

[3] D. C. Feldmeier and P. R. Karp. Unix password security - ten years later. In *CRYPTO*, pages 44–63, 1989.

[4] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[5] S. Gaw and E. W. Felten. Password management strategies for online accounts. In *SOUPS '06: Proceedings of the Second Symposium on Usable Privacy and Security*, pages 44–55, New York, NY, USA, 2006. ACM Press.

[6] S. Jeyaraman and U. Topkara. Have the cake and eat it too – infusing usability into text-password based authentication systems. In *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference*, pages 473–482, Washington, DC, USA, 2005. IEEE Computer Society.

[7] C. Kuo, S. Romanosky, and L. F. Cranor. Human selection of mnemonic phrase-based passwords. In *SOUPS '06: Proceedings of the Second Symposium on Usable Privacy and Security*, pages 67–78, New York, NY, USA, 2006. ACM Press.

[8] G. Miller. Human Memory and the Storage of Information. *Information Theory, IEEE Transactions on*, 2(3):129–137, 1956.

[9] R. Morris and K. Thompson. Password security: A case history. *ACM Communcations*, 22(11):594–597, 1979.

[10] T. Pedersen, S. Patwardhan, and J. Michelizzi. Wordnet::Similarity – Measuring the Relatedness of Concepts. In *Proceedings of Fifth Annual Meeting of the NAACL*, Boston, MA, May 2004.

[11] B. Pinkas and T. Sander. Securing passwords against dictionary attacks. In *Proceedings of the ACM Computer and Security Conference*, pages 161–170, November, 2002.

[12] P. Resnik. Selectional preference and sense disambiguation. In *The ACL SIGLEX Workshop on Tagging Text with Lexical Semantics: Why, What, and How?*, Washington D.C., USA, April 1997.

[13] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1993.

[14] B. Schneier. Write down your password. *http://www.schneier.com/blog/archives/2005/06/write_down_your.html*, June 2005.

[15] U. Topkara, M. Topkara, and M. J. Atallah. The hiding virtues of ambiguity: Quantifiably resilient watermarking of natural language text through synonym substitutions. In *Proceedings of ACM Multimedia and Security Workshop*, Geneva, Switzerland, September 26-27, 2006.

[16] L. von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *Proceedings of Eurocrypt*, pages 294–311, 2003.

[17] K. Winstein. Lexical steganography through adaptive modulation of the word choice hash. In *http://www.imsa.edu/ keithw/tlex/*, 1998.

[18] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: Empirical results. *IEEE Security and Privacy*, 2:25–31, 2004.