IDENTIFYING PASSWORDS STORED ON DISK

Shiva Houshmand, Sudhir Aggarwal, Umit Karabiyik

Chapter 12

IDENTIFYING PASSWORDS STORED ON DISK

Shiva Houshmand, Sudhir Aggarwal and Umit Karabiyik

Abstract This chapter presents a solution to the problem of identifying passwords on storage media. Because of the proliferation of websites for finance, commerce and entertainment, the typical user today often has to store passwords on a computer hard drive. The identification problem is to find strings on the disk that are likely to be passwords. Automated identification is very useful to digital forensic investigators who need to recover potential passwords when working on cases. The problem is nontrivial because a hard disk typically contains numerous strings. The chapter describes a novel approach that determines a good set of candidate strings in which stored passwords are very likely to be found. This is accomplished by first examining the disk for tokens (potential password strings) and applying filtering algorithms to winnow down the tokens to a more manageable set. Next, a probabilistic context-free grammar is used to assign probabilities to the remaining tokens. The context-free grammar is derived via training with a set of revealed passwords. Three algorithms are used to rank the tokens after filtering. Experiments reveal that one of the algorithms, the one-by-one algorithm, returns a password-rich set of 2,000 tokens culled from more than 49 million tokens on a large-capacity drive. Thus, a forensic investigator would only have to test a small set of tokens that would likely contain many of the stored passwords.

Keywords: Disk examination, stored passwords, password identification

1. Introduction

Passwords continue to be the primary means for authenticating users. Because of the proliferation of websites related to banking, commerce and entertainment, the typical user necessarily maintains multiple accounts and passwords. Meanwhile, for security reasons, many websites have adopted password policies that force users to register passwords

that conform to certain length, symbol and digit requirements. Since it is difficult to remember multiple, complex passwords, users increasingly save their passwords either on paper or on their computers. When storing passwords on disk, users typically use password management software or the password recall option provided by browsers or save passwords directly on their computers or cell phones. A 2012 survey by Kaspersky Lab [8] revealed that 29% of users do not remember passwords, but instead store them on media. The survey also reports that 13% of users create text documents with passwords that they store on their hard drives, 9% save passwords on their cell phones and only 7% use specialized password management software. As passwords become more complex, greater numbers of users will turn to storing their passwords on their computer hard drives. Indeed, an informal survey of 100 students conducted as part of this research revealed that 42% of the students store their passwords and 55% of these students save them on hard drives or cell phones without using encryption or specialized software.

Some researchers have developed techniques for extracting cryptographic keys and passwords stored in browsers [5, 10, 14]. However, a search of the literature does not reveal any research that attempts to distinguish passwords from other strings stored directly on disk by users. This problem, involving the identification of passwords in media, is the focus of this chapter.

An example scenario [4] is a disk containing encrypted files (of say illegal photographs) and there is a likelihood that the user has stored the passwords somewhere on the disk in order to easily access the encrypted files. A forensic investigator could review each saved file and attempt to determine, by context and structure, the strings that might correspond to passwords. This would, of course, be a very tedious task, especially if the disk has high capacity and holds numerous files.

Investigators sometimes use software tools to tokenize all the strings found on disks and incorporate them in dictionaries for offline cracking of encrypted files. However, it is often the case that the list of strings becomes too large to be used in dictionary-based password cracking. The identification problem is to distinguish the tokens that are likely to be passwords and winnow down the list to a manageable size. This problem is non-trivial because distinguishing the strings that may be passwords from a large set of strings has no obvious solution.

The solution described in this chapter first analyzes a disk image and attempts to retrieve all the strings that could possibly be passwords (these are called "tokens"). During the process of retrieval and subsequent processing, which is called "filtering," a potentially very large set of tokens is reduced to a manageable set that contains most of the

passwords. This is accomplished by applying specialized filters to reduce the size of the token set. A previously-trained probabilistic context-free grammar [6, 16] is employed to decide which of the remaining tokens are likely to be passwords by assigning each token a probability. The probability values are used as input to ranking algorithms that produce ordered lists of tokens that represent possible passwords. The token lists can be used in dictionary-based password cracking.

This work is unique in its specification of the problem of identifying passwords on disk, its use of a probabilistic context-free grammar and its development of filters and ranking algorithms. Experiments demonstrate that the approach, on the average, identifies 60% of the passwords in the top 2,000 potential passwords returned to an investigator. Moreover, the approach is directly applicable to identifying passwords stored on cell phones and USB flash drives.

2. Related Work

Little, if any, research has focused on password identification in storage media. Garfinkel et al. [2] have studied the general problem of attempting to capture a storage profile of an individual computer to detect anomalous behavior. They propose the monitoring of the forensic content of the disk (or other media), such as email addresses and credit card numbers. The approach described in this chapter could, in fact, be used by their technique to capture potential password strings.

With respect to identifying passwords, many forensic tools exist for finding desired strings on disk; this means that the specific passwords being searched for are known. Forensic recovery tools such as EnCase and FTK can be used to find passwords on a hard disk, but these tools merely return a list of all the strings on the disk. As will be seen, the real problem is filtering the potentially large number of strings and determining the strings that are most likely to be passwords.

Sensitive Data Manager [7] is designed to search for data such as passwords, credit card numbers and social security numbers in several locations, including files, email, browsers, registry files, databases and websites. As far as searching for data on a hard disk is concerned, Sensitive Data Manager checks only the files that have metadata information available to the filesystem; it does not examine the unallocated space to search for passwords in deleted files. Sensitive Data Manager provides password search customization by enabling keyword and regular expression searches; this requires an investigator to conduct string searches as in the case of EnCase and FTK. Sensitive Data Manager, thus, does not tackle the password identification problem.

A related problem is to find a password that has been used to encrypt a file or to find the login password of a user. Hargreaves and Chivers [5] have attempted to recover encryption keys from memory; they demonstrated that if the memory is preserved when encrypted files are open, it is possible to find the encryption keys. Similarly, Lee et al. [10] have had success in finding login passwords by collecting and examining page files. Their work is interesting, but it is orthogonal to the password identification problem.

Attempts have also been made to find passwords that are stored by browsers. Chrome Password Recovery [14] is an open-source, command-line password recovery tool that retrieves login information (usernames, passwords and website links) from Google Chrome. However, it only focuses on recovering sensitive data such as encryption/decryption keys from encrypted Google Chrome password database files; it is not designed to identify passwords saved on hard disks.

3. Background

This section discusses the principal concepts underlying the proposed solution to the password identification problem.

3.1 Probabilistic Context-Free Grammars

Probabilistic context-free grammars have been used to create better passwords as well as to crack passwords. This research uses a grammar to rank tokens recovered from a hard disk based on their likelihood of being user passwords. Specifically, the probabilistic password cracking approach described in [6, 16] is used to assign probability values to potential password tokens. In this approach, the base structure of a password is defined according to the component type (L for alpha string, D for digit string and S for special symbol string) and the length of the string is incorporated in the base structure. For example, the password string *alice*123#% has the base structure $L_5D_3S_2$. A probabilistic grammar is then determined from a training corpus by calculating the frequencies of all the base structures and the component structures (of each length) found in the corpus. The resulting information is structured as a context-free grammar. For example, *alice*123#% is derived as follows:

$$S \Rightarrow L_5D_3S_2 \Rightarrow aliceD_3S_2 \Rightarrow alice123S_2 \Rightarrow alice123\#\%$$

At each step, a probability is assigned to the rule transition. The product of the transition probabilities is the probability of the resulting string. As shown in [6], this approach can be used to compute the

probability of any token given a context-free grammar derived from a realistic password corpus.

4. Examining a Disk

The goal is to identify passwords in files stored on disk by a user. Note that the assumption is that the user is not actively trying to hide passwords. In fact, the user simply stores the passwords in a file on disk that may or may not contain other text; this is done so that the user can easily access a forgotten password. The file may be in allocated space or unallocated space (i.e., the file has been deleted) or it may be hidden using the operating system.

A more sophisticated user might use specialized software to hide passwords in unallocated space in a partition, in file slack space or in some other non-filesystem space. Although this scenario is not the focus of this research, it should still be possible to find such data on the disk as long as encryption or steganography are not used. For example, if a user explicitly hides data or a file in the slack space of a file or partition, the slack space tool `bmap` or file carver `scalpel` could, respectively, be used to retrieve the data or file from slack space. The recovered data could be written to a text file for the tokenization process described in Section 4.2. The approach could fail under certain circumstances, such as when passwords are hidden in an arbitrary space on disk. Although the string stream could be viewed using a hex editor and the data echoed to a text file, it can be difficult to determine string boundaries when garbage data is intentionally or unintentionally added to the string stream. These circumstances greatly complicate the password identification problem.

4.1 Recovering Files from a Disk

The first step is to retrieve all the different types of files from the filesystem of a given disk image. This can be done using `tsk_recover`, a component of The Sleuth Kit, an open-source, digital forensics tool that provides several command-line tools for analyzing disk images. Note that `tsk_recover` can recover files from allocated space as well as unallocated space. This work specifically focuses on filesystems that are not corrupted. Files for which metadata information is lost or damaged are not considered. Data carving tools could also be used to retrieve files that might reside in other parts of the disk such as slack space, lost partitions and unallocated space in partitions. The tools could be used to create one or more files that could then be analyzed in the same way as the uncorrupted files.

4.2 Retrieving Tokens from Files

After recovering files from the disk, the next step is to extract the strings that are potential passwords. This is accomplished by extracting white-space separated strings from file types such as `.doc`, `.docx`, `.xls`, `.xlsx`, `.rtf`, `.odt`, `.pdf` and `.txt`.

Since some of the file types are not readable by text editors, these files must be converted to the text file format in order to be able to read their contents. The open-source tools, `catdoc`, `docx2txt`, `xls2txt`, `unoconv` and `xls2txt`, `unrtf`, `odt2txt` and `pdftotext` were, respectively, used to convert `.doc`, `.docx`, `.xls`, `.xlsx`, `.rtf`, `.odt` and `.pdf` files to the text file format.

Spaces, tabs and newlines were used as delimiters to tokenize string streams in a file; the associated text file was created by writing each token on a separate line. The resulting text files corresponding to all the files on disk were then searched to find potential passwords.

The accuracy of the tokens retrieved from the files is based entirely on the conversion performance of the tools. Two possible problems exist. First, some strings may be altered during the conversion process. Second, some new strings (i.e., not in the original file) may have been created by the tools. This latter situation only occurred when converting `.xls` and `.xlsx` files containing multiple spreadsheets for which the tool would add each sheet name. In the experiments conducted using the tools, a problem was rarely encountered during the conversion process. All the strings in the files were obtained, including from tables in `.doc` and `.docx` files and everything in the cells of `.xls` and `.xlsx` files. The only thing that was not obtained was text in the images residing in these files. Images sometimes create strings in the conversion process that are completely filtered out later.

4.3 Initial Filtering

Even an average-sized disk typically contains many different file types and files with text content; this results in a massive number of tokens. In order to reduce the number of tokens retrieved, rules were defined to filter some classes of tokens that are very unlikely to be passwords. Several revealed password sets (e.g., results of attacks on various websites such as Rockyou [15] and Yahoo [12]) were examined to obtain insights into the kinds of structures that are rarely seen in passwords. The following initial filters were defined and applied:

- **Non-Printable:** This filter eliminates ASCII characters that are almost always not valid password characters.

- **Length:** Passwords usually have certain lengths based on the password policy that is enforced. This filter applies a conservative bound and only tokens of length l, where $6 < l < 21$, are retained. In the Yahoo set, only 1.93% of the passwords have length less than seven and 0.047% of the passwords have length greater than 20.

- **Floating Point:** The files on disk (especially `.xls files`) often include many floating point numbers. It is a good idea to filter floating point numbers because studies of revealed password sets reveal that there is very little chance that such tokens correspond to passwords. Thus, this filter eliminates strings corresponding to the regular expression `[-+]? [0-9]* .? [0-9]+ ([eE][-+]?[0-9]+)?`.

- **Repeated Tokens:** This filter retains one copy of each repeated token in a file. One might assume that repeated tokens are unlikely to be passwords, but users often use the same password for multiple accounts, so there would be multiple instances of a given password string in a file.

- **Word Punctuation:** This filter eliminates punctuation patterns encountered in sentences. Specifically, tokens containing only alpha strings ending with one of the following characters `; : , . ? ! - )` } were filtered. Also, tokens starting with `(` or { were filtered. An examination revealed that only 0.516% of such tokens were present in a sample of one million passwords in the Rockyou set.

4.4 Specialized Alpha String Filtering

It is obvious that English words constitute a very large part of every text file. Therefore, an extremely prevalent class of tokens found on a hard disk is the set of alpha strings (i.e., strings containing only alphabetic characters). This research considers various approaches for handling such strings. In particular, the following specialized alpha string filters are defined:

- **All-Alphas:** This filter eliminates tokens that only have alphabetic characters. The assumption is that the vast majority of passwords contain other symbols (e.g., digits and special characters). This assumption is validated by most password creation policies.

- **Sentences:** This filter eliminates all alpha strings that are only within sentences. The OpenNLP tool [1] was used to detect sen-

tences after the file type conversion. This tool detects whether or not a punctuation character marks the end of a sentence. However, it cannot identify sentence boundaries based on sentence content. It only returns a set of indices for which each sentence (or non-sentence) is on a separate line. An additional problem encountered after the conversion process to a `.txt` file was that word wrapping was not preserved when line breaks were added; thus, sentences that ran over multiple lines were output as multiple indices by OpenNLP. Heuristics were used to detect indices that corresponded to sentences (and non-sentences). For example, an index that started with a capital letter and ended with a period [13] may be filtered.

- **Capitalization:** This filter eliminates all lower case alpha strings. This is because most password policies require passwords to have characters from one or more other classes (e.g., symbols, digits and capital letters).
- **Dictionary Words:** This filter eliminates alpha strings that appear in an English dictionary. This eliminates strings that are most likely words in sentences while retaining the remaining strings in the token set.
- **Multiwords:** This filter eliminates all alpha strings that are not multiwords. A multiword is a string that consists of two or more words in an English dictionary. Examples are passphrases (without whitespace) that are increasingly used as passwords.

5. Identifying Passwords

This section focuses on the problem of distinguishing and finding a password from among a set of tokens. Specifically, after the hard disk has been examined and all the tokens separated by whitespace are obtained, a mechanism is required to distinguish passwords from other sequences of characters that appear in the text.

5.1 Calculating Token Probabilities

As discussed above, a probabilistic context-free grammar can be created from a large set of real user passwords. The probabilistic context-free grammar models a password distribution and the way users create passwords. Knowledge of the structure of passwords enables passwords to be differentiated from regular text.

Given a probabilistic context-free grammar, the probability of a string in the password distribution can be computed. This research employed

the method described in [6] to calculate the probability. Each string was parsed into its components and the probability associated with each component of the probabilistic context-free grammar was computed.

As an example, consider the string *violin22*, which is represented as the base structure L_6D_2. The product of probabilities of the components (i.e., base structure L_6D_2, alpha string *violin*, all lower case capitalization M_6 and digit component *22*) is the estimated probability value of the string. Using this approach, the probabilities of all the retrieved tokens were computed. Note that in the remainder of the discussion, the retrieved tokens correspond to tokens that remain after the initial filtering.

5.2 Ranking Algorithms

After computing the probability of each token, the tokens must be ranked and a limited set of tokens (say the top N tokens called "potential passwords") must be provided to the investigator to examine as the most likely password candidates from the hard disk. Obviously, it is ideal to have high precision and high recall in the potential password set. Recall is generally more important in offline password cracking while precision is more important in online cracking. One might argue that, in the case of an offline attack, one could consider all the tokens found on the disk. However, it is very important to reduce the size of the potential password set – although password cracking is continually being sped up by GPUs, many hashing algorithms (e.g., the one used in TrueCrypt) can still take a very long time using the resources available to most law enforcement agencies.

In order to obtain the best precision and recall, several approaches were used to identify the top potential passwords from the retrieved tokens. This section describes three algorithms for ranking possible passwords. Note that the strings and the file they are associated with are maintained and this relationship is exploited by the algorithms. The algorithms incorporate a parameter N denoting the number of potential passwords to be returned to the investigator. The algorithms are:

- **Top Overall:** This algorithm selects the N highest probability tokens from among all the retrieved tokens. However, the results presented later in the chapter show that this is not the most effective approach.

- **Top Percent (per File):** This algorithm selects a fixed percentage of the highest probability tokens from each file such that the total number of tokens returned is N (thus, different numbers of

tokens are selected from each file). The resulting tokens are then ordered according to their probabilities.

- **Top 1-by-1 (per File):** The first round of the algorithm chooses the highest probability token from each file and ranks them according to their probabilities. In the second round, the second highest probability token is selected from each file (if available) and they are ranked according to their probabilities. The rounds are continued until N tokens are obtained. Note that tokens in round j are ranked above tokens in round $j + 1$.

6. Experimental Evaluation

This section discusses the experimental results related to the utility of the filtering techniques as well as the effectiveness of the algorithms in identifying passwords. Since disk images containing real passwords were not available, test disk images were created by incorporating files containing a number of real passwords taken from revealed password sets.

6.1 Experimental Setup

The Govdocs1 digital corpus [3] was used as the source of files. This corpus contains about one million freely redistributable files in many formats. Five data disk images of different sizes (50 MB, 100 MB, 250 MB, 500 MB and 1 GB) were created with FAT filesystems. Each test disk image modeled a real disk and corresponded to a subset of a typical complete disk. A fairly large amount of space on a real disk is devoted to the operating system, media files (videos, music, images) and installed programs. The test disks only incorporated files that are likely to be created by users (`.doc`, `.xls`, `.pdf`, etc.). The sizes of the data disk images correspond to the total sizes of these files. For example, the 1 GB test data disk would likely have been derived from a 500 GB hard disk belonging to a typical user. The numbers of files analyzed in the five disks with sizes 50 MB, 100 MB, 250 MB, 500 MB and 1 GB were 108, 143, 426, 571 and 1,194, respectively.

Passwords were randomly selected from a revealed password set and files were randomly selected for storing the passwords. Since no data was available on how users store their passwords (either in one file or many files or at the end of large files, etc.), attempts were made to be as general as possible when adding the passwords. The experiments used passwords from three well-known revealed password sets: one million passwords from Rockyou [15], 300,000 from CSDN [9] and 300,000 from

Table 1. Percentage token reduction per filter.

Disk Filter	50 MB	100 MB	250 MB	500 MB	1 GB
Non-Printable	0	0	0	0.0015	0
Length	59.65	65.57	60.34	40.75	53.08
Floating Point	1.05	0.45	20.71	46.87	28.21
Repeated Token	85.04	82.79	73.78	75.63	70.10
Word Punctuation	68.96	11.90	8.27	6.28	20.42
All-Alpha	77.89	73.11	60.66	31.95	33.71

Yahoo [12]. Interested readers are referred to Ma et al. [11] for statistical information about these password sets.

6.2 Initial Filtering

Experiments were conducted to assess how the filters help reduce the number of tokens. The initial filtering techniques described previously were employed and the numbers of tokens obtained before and after applying each of the initial filters for each disk size were recorded. The most aggressive specialized filtering involving the removal of all alpha strings was also applied.

Table 1 presents the results. Note that all the filters, except non-printable, have a major impact on the results, reducing the large number of tokens obtained from the hard disk to a much smaller and manageable set. The non-printable filter turned out to be important in the next step involving the computation of probabilities, but it was rarely useful for token reduction. The order of application of the filters does not matter (except for the time requirements) because the same results are obtained regardless of the order in which filtering is performed.

Table 2. Token reduction by all filters.

	50 MB	100 MB	250 MB	500 MB	1 GB
# Before Filtering (mil.)	2.45	2.16	6.76	28.84	49.41
# After Filtering (mil.)	0.07	0.050	0.25	1.38	3.21
Total Reduction (%)	97.15	97.68	96.35	95.21	93.50

Table 2 shows the numbers of tokens (in millions) before and after filtering and the percentage reductions after all the filters were applied.

6.3 Ranking Algorithms

Experiments were conducted to evaluate the results of the three ranking algorithms. The Rockyou and CSDN revealed password sets were used to provide passwords that were stored on the test disks. The initial filters and the all-alpha filter were used in the experiments.

Some websites (e.g., Rockyou) did not enforce a password policy at the time they were attacked. Consequently, a good number of passwords in their lists have length less than seven or are alpha strings. Because this series of experiments only sought to evaluate the ranking algorithms, such passwords were not stored on the test disks. Specifically, five passwords were stored on each disk in one set of experiments and fifteen passwords were stored in the second set of experiments. The assumption was that users would typically store between five to fifteen passwords. The passwords were stored either in a file or in a deleted file. The Yahoo password set was used to train the probabilistic context-free grammar that was used to calculate the probabilities of potential passwords.

For each combination of disk size, revealed set and number of passwords stored, the number of passwords that could be found using the three algorithms (top overall, top percent and top 1-by-1) were determined. The results are presented in terms of N potential passwords provided to the investigator, where N is 1,000, 2,000, 4,000, 8,000 and 16,000.

Table 3 shows the results of storing five and fifteen passwords from CSDN. For example, the 1-by-1 algorithm, finds all five passwords in the 50 MB test data disk, three passwords in the 100 MB disk, two passwords in the 250 MB disk, etc., within the top $N = 1,000$ passwords returned by the algorithm.

The results obtained when storing five passwords are discussed first. When comparing the algorithms, given an N value and the number of stored passwords, higher recall implies higher precision and both values can be calculated from the number of passwords found. For example, the average recall value for the 1-by-1 algorithm across different disk sizes for $N = 8,000$ is 92%; for the top percent algorithm, the average recall value is 56%; and for the top overall algorithm, the average recall value is 40%. This shows that the 1-by-1 algorithm has higher precision and higher recall compared with the other algorithms.

The results obtained when storing fifteen passwords are similar. For $N = 8,000$, the average recall value of the 1-by-1 algorithm is 89.3% across the various disk sizes. Table 3 shows that the 1-by-1 algorithm has good performance and is better than the other algorithms.

Table 3. Number of CSDN passwords found.

N	Disk Size	50 MB	100 MB	250 MB	500 MB	1 GB	50 MB	100 MB	250 MB	500 MB	1 GB
		Out of 5 Passwords					Out of 15 Passwords				
1,000	Overall	1	2	0	0	2	1	7	0	2	2
	Percent	2	3	1	1	2	4	10	2	3	3
	1-by-1	**5**	**3**	**2**	**3**	**3**	**11**	**12**	**7**	**8**	**9**
2,000	Overall	1	2	0	0	2	1	9	0	2	2
	Percent	5	3	1	1	2	9	10	2	4	5
	1-by-1	**5**	**4**	**2**	**3**	**4**	**12**	**14**	**9**	**9**	**11**
4,000	Overall	5	2	0	0	2	11	10	0	2	2
	Percent	5	3	2	1	2	10	11	3	5	6
	1-by-1	**5**	**5**	**3**	**4**	**4**	**15**	**15**	**12**	**10**	**12**
8,000	Overall	5	3	0	0	2	13	11	0	2	2
	Percent	5	3	2	1	3	11	11	8	5	8
	1-by-1	**5**	**5**	**4**	**4**	**5**	**15**	**15**	**13**	**10**	**14**
16,000	Overall	5	4	0	0	2	15	14	0	2	2
	Percent	5	4	2	3	3	12	14	9	8	8
	1-by-1	**5**	**5**	**4**	**5**	**5**	**15**	**15**	**13**	**11**	**14**

Table 4 shows the results for stored passwords from the Rockyou password set. When five passwords from Rockyou were stored, the average recall value of the 1-by-1 algorithm for $N = 8,000$ is 84%, for the top percent algorithm the average recall value is 72% and for the top overall algorithm the average recall value is 60%. In the case of a smaller N value ($N = 1,000$), the average recall value for the 1-by-1 algorithm when fifteen passwords were stored is 81.3% and the average precision is 1.2%.

The amount of time taken by each algorithm was also measured. The total time taken for retrieving the tokens, filtering, ranking and returning the top potential passwords from the 1 GB data disk was less than three minutes. The total time for the smallest data disk (50 MB) was just thirteen seconds.

Examination of the results in Tables 3 and 4 reveals that the 1-by-1 algorithm has the most consistent performance.

6.4 Specialized Filtering

The filters used in the experiments described above eliminated all the alpha strings. This is reasonable because the vast majority of password policies would disallow such passwords. Nevertheless, the experiments

Table 4. Number of Rockyou passwords found.

N	Disk Size	50 MB	100 MB	250 MB	500 MB	1 GB	50 MB	100 MB	250 MB	500 MB	1 GB
		Out of 5 Passwords					Out of 15 Passwords				
1,000	Overall	1	2	3	3	2	8	9	8	8	9
	Percent	3	2	1	2	0	8	10	4	3	2
	1-by-1	**4**	**4**	**4**	**4**	**2**	**13**	**13**	**12**	**13**	**10**
2,000	Overall	1	2	3	3	2	8	11	8	8	9
	Percent	4	2	2	3	1	10	10	7	4	5
	1-by-1	**4**	**4**	**4**	**5**	**2**	**14**	**13**	**13**	**15**	**11**
4,000	Overall	4	3	3	3	2	14	12	8	8	9
	Percent	4	4	3	5	2	13	13	9	10	6
	1-by-1	**4**	**5**	**4**	**5**	**2**	**14**	**14**	**13**	**15**	**11**
8,000	Overall	4	3	3	3	2	14	12	8	8	9
	Percent	4	4	3	5	2	13	13	12	13	7
	1-by-1	**4**	**5**	**5**	**5**	**2**	**14**	**15**	**15**	**15**	**11**
16,000	Overall	5	4	3	3	2	15	13	8	8	9
	Percent	4	4	4	5	2	14	13	13	14	7
	1-by-1	**5**	**5**	**5**	**5**	**2**	**15**	**15**	**15**	**15**	**11**

described in this section examine whether less restrictive filtering of alpha strings is useful.

In particular, the experiments used the specialized filters described previously to retain some of the alpha strings. The filters were applied in addition to the initial filters. The experiments used the 1 GB data disk with fifteen stored passwords from the Rockyou set. Alpha string passwords were permitted to be selected from the Rockyou set for storage. Note that these passwords could potentially be filtered before the identification process.

Table 5 shows the results for various specialized filters (N: no filter, C: capitalization, M: multiwords, D: dictionary, S: sentences and A: all-alphas). The numbers in parentheses correspond to how many of the fifteen passwords stored on the disk remained after the filtering process. For example, A (11) means that four of the fifteen passwords stored on the disk were filtered by the all-alphas filter.

The multiwords filter (M) eliminates all single words (whether they are dictionary words or not). The dictionary filter (D), however, only eliminates single words included in the dictionary and retains multiwords. Therefore, when applying the multiwords filter, a more limited set of alpha strings is retained. The dictionary filter used a moderate-sized English dictionary, which was designed for Scrabble-style computer

Table 5. Comparing specialized filters.

		N (15)	**C (11)**	**M (14)**	**D (14)**	**S (15)**	**A (11)**
N=1,000	Overall	0	2	0	0	0	5
	Percent	1	1	3	3	2	1
	1-by-1	**2**	**2**	**4**	**4**	**0**	**8**
N=2,000	Overall	0	2	0	0	0	5
	Percent	1	2	3	3	2	2
	1-by-1	**2**	**2**	**4**	**5**	**0**	**10**
N=4,000	Overall	0	2	0	0	0	5
	Percent	2	3	3	3	3	4
	1-by-1	**2**	**2**	**5**	**5**	**1**	**10**
N=8,000	Overall	0	2	0	0	0	5
	Percent	4	4	5	5	3	7
	1-by-1	**2**	**2**	**7**	**7**	**1**	**10**
N=16,000	Overall	0	2	0	0	0	5
	Percent	4	4	5	5	3	7
	1-by-1	**4**	**5**	**8**	**8**	**7**	**10**

word games; the dictionary was augmented with common names and common words from television and movie scripts. The results without any alpha string filtering (N) are also shown.

Table 5 shows that using less aggressive filters such as the multiwords and dictionary filters reduces password loss due to filtering. However, these filters are not as successful as the more aggressive approach of using the all-alphas filter and subsequently identifying the passwords because they retain too many alpha strings. Because of the large number of alpha strings that appear in the final token list and because of the way in which the probability of each token is calculated (words of the same length have equal probability), a large number of alpha strings with fairly high probabilities are obtained. Hence, when the top N potential passwords are selected by the algorithms, many of the passwords are not found as quickly as when all the alpha strings are filtered.

The sentences filter (S) is designed to reduce documents containing text but, as noted previously, the tool that was used was unable to distinguish between sentences and non-sentences. The results in Table 5 show that the sentences filter is not as useful as the other filters. Better tools for identifying sentences render the sentences filter more useful for password identification.

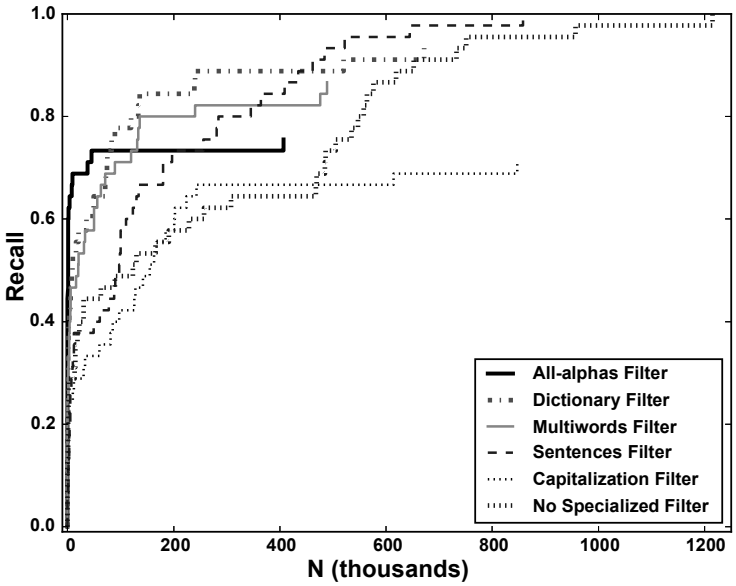

Figure 1. Comparison of specialized filters for various N values.

The results in Table 5 also show that using all the filters is better than using no filters. Also, the all-alphas filter is much more effective than the other filters even though it may filter some passwords before the identification process. Furthermore, as before, the 1-by-1 algorithm has the best overall performance.

To assess how quickly passwords could be identified, the 1-by-1 algorithm was used along with the various specialized alpha string filters (in addition to the initial filters). Figure 1 shows plots of N versus the recall value until all the passwords that can be found by a given filter are obtained (the results are the averages of several runs). Note that some filters cannot achieve a recall value of one because some of the passwords were filtered before the identification process. For example, the aggressive all-alphas filter may not be able to find all of the passwords, but, on the average, it finds nine of the fifteen passwords with a recall of 0.6 and precision of 0.005 at $N = 1,659$. In comparison, when no specialized filters are applied, nine of the fifteen passwords are found only at the very much higher $N = 229,671$.

The results demonstrate that the filtering and identification approach enables an investigator to find most of the passwords at a very small value of N, eliminating the need to check a massive number of strings. Note that if the aggressive filter is not successful, an investigator could use a less aggressive filter. For example, using the dictionary filter, which

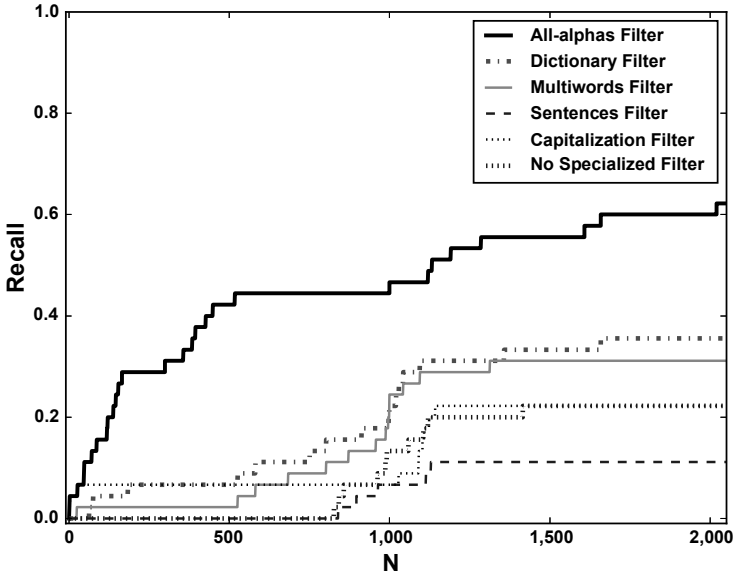

Figure 2. Comparison of specialized filters for small N values.

loses fewer passwords and finds an average of nine of the fifteen passwords at $N = 36,240$, is still much better than not using a specialized filter.

By choosing an appropriate value of N, an investigator can move between the online and offline password cracking modes. In order to better understand the implications of small values of N, Figure 2 presents the results for initial values of N up to 2,000. Note that the all-alphas filter identifies about half of the stored passwords within the first 500 tokens. Even finding just one password on a disk can be very helpful to an investigator because users often reuse a single password for multiple accounts. Note that, for the all-alphas filter in Figure 1, the first password was found on the average at $N = 11$.

Readers may be interested in seeing examples of the potential passwords returned by the 1-by-1 algorithm. Table 6 shows the top twenty potential passwords and their associated probabilities when using the all-alphas filter. Two of the fifteen passwords stored on the disk (i.e., *lyndsay1* and *blueberry1*) are among the top twenty potential passwords returned by the system. It is not clear how the filtering and ranking could be improved because all the other potential passwords appear to be possible passwords as well.

Table 6. Top twenty passwords.

Rank	Potential Passwords	Probability	Rank	Potential Passwords	Probability
1	*charles1*	6.384 E-6	11	*pdprog1*	5.370 E-8
2	*include3*	1.687 E-6	12	*report1*	5.370 E-8
3	*program4*	1.610 E-6	13	*cielo123*	5.080 E-8
4	*carolina23*	6.272 E-7	14	*soldiers1*	4.044 E-8
5	*light20*	1.112 E-7	15	*bluberry1*	4.044 E-8
6	*program97*	7.757 E-8	16	*listeria1*	4.044 E-8
7	*lyndsay1*	7.739 E-8	17	*compendia1*	3.110 E-8
8	*decagon1*	7.739 E-8	18	*framework1*	3.110 E-8
9	*dogbloo1*	7.739 E-8	19	*alpha1s*	2.972 E-8
10	*example1*	7.739 E-8	20	*address2*	2.538 E-8

7. Conclusions

The proposed technique involving a probabilistic context-free grammar, specialized filters and ranking algorithms is very effective at identifying potential passwords stored on hard disks. Experiments demonstrate that the technique can provide a relatively small list of tokens that contain most of the stored passwords. For example, given approximately 49 million tokens from a large hard drive as input, the technique was able to output an ordered potential password set of 2,000 tokens, which contained nine of the fifteen stored passwords. Furthermore, the technique can very quickly find at least a few of the passwords – on the average, one password was found in the top eleven tokens and three passwords in the top fifty tokens.

Future research will apply the technique to other devices and media, including cell phones and USB drives. Also, the research will explore other approaches for identifying and filtering sentences. The filters and the ranking algorithms will also be adapted to leverage other information such as the password policy and the names and dates of birth of family members.

References

[1] Apache Software Foundation, OpenNLP, Forest Hill, Maryland (`opennlp.apache.org`), 2010.

[2] S. Garfinkel, N. Beebe, L. Liu and M. Maasberg, Detecting threatening insiders with lightweight media forensics, *Proceedings of the International Conference on Technologies for Homeland Security*, pp. 86–92, 2013.

[3] S. Garfinkel, P. Farrell, V. Roussev and G. Dinolt, Bringing science to digital forensics with standardized forensic corpora, *Digital Investigation*, vol. 6(S), pp. S2–S11, 2009.

[4] S. Grimm, Personal communication, Webster Groves Police Department, Webster Groves, Missouri, 2014.

[5] C. Hargreaves and H. Chivers, Recovery of encryption keys from memory using a linear scan, *Proceedings of the Third International Conference on Availability, Reliability and Security*, pp. 1369–1376, 2008.

[6] S. Houshmand and S. Aggarwal, Building better passwords using probabilistic techniques, *Proceedings of the Twenty-Eighth Annual Computer Security Applications Conference*, pp. 109–118, 2012.

[7] Identity Finder, Sensitive Data Manager, New York (`www.identityfinder.com/us/Business/IdentityFinder/SensitiveDataManager`).

[8] Kaspersky Lab, Perception and Knowledge of IT Threats: The Consumer's Point of View, Woburn, Massachusetts, 2012.

[9] M. Kumar, China Software Developer Network (CSDN) 6 million user data leaked, *The Hacker News*, December 21, 2011.

[10] S. Lee, A. Savoldi, S. Lee and J. Lim, Password recovery using an evidence collection tool and countermeasures, *Proceedings of the Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 97–102, 2007.

[11] J. Ma, W. Yang, M. Luo and N. Li, A study of probabilistic password models, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 689–704, 2014.

[12] S. Musil, Hackers post 450K credentials pilfered from Yahoo, *CNET*, July 11, 2012.

[13] J. O'Neil, Doing Things with Words, Part Two: Sentence Boundary Detection, Attivio, Newton, Massachusetts, 2008.

[14] D. Riis, Google Chrome Password Recovery Tool (`www.bitbucket.org/Driis/chromepasswordrecovery`), 2012.

[15] A. Vance, If your password is 123456, just make it HackMe, *New York Times*, January 20, 2010.

[16] M. Weir, S. Aggarwal, B. De Medeiros and B. Glodek, Password cracking using probabilistic context free grammars, *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 391–405, 2009.