# Observing Reusable Password Choices*

Purdue Technical Report CSD–TR 92–049

*Eugene H. Spafford*
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907–1398
spaf@cs.purdue.edu

31 July 1992

**Abstract**

From experience, a significant number of recent computer breakins — perhaps the majority — can be traced back to an instance of a poorly-chosen reusable password. Once a system intruder has gained access to one account by breaking a password, it is often a simple matter to find system flaws and weaknesses that thereafter allow entry to other accounts and increasing amounts of privilege.

The OPUS project being conducted at Purdue is an attempt to screen users' selection of passwords to prevent poor choices. The focus of the project is on using screening methods that are both time and space-efficient and to provide a mechanism that is effective for workstations with little or no disk as well as mainframes.

To test this mechanism, we require a representative sample of real passwords. Thus, we constructed a method of sampling real passwords choices as they were made by users. The challenge of such a sampling mechanism is how to protect it from attack, and how to protect the results from being used against the system. This paper discusses our approach, and some of our initial observations on the words collected.

## 1 Introduction

From experience, a significant number of recent computer breakins — perhaps the majority — can be traced back to an instance of a poorly-chosen reusable password.[1] Once a system intruder has gained access to one account by breaking a password, it is often a simple matter to find system flaws and weaknesses that thereafter allow entry to other accounts and increasing amounts of privilege.

Reusable passwords provide a cost-effective method of authentication. They do not require special hardware, they may be used from any form of dial-in port or terminal connected to the target system, the

* This paper also appeared as [11].

[1] Communications from members of the CERT/CC and from local system administrators support this observation.

mechanism can easily be understood by most users, and they allow users to manage some of their own account security. Reusable passwords have been widely studied[13] and they seem to be well-understood.. For these reasons, they are often used as the primary form of authentication in multi-user systems, and in many single-user products.

Reusable passwords also present problems. Although they are easy to use and low in cost, they rely on the strength of the passwords chosen and their resistance to concerted attacks. When users choose their own passwords, the security is tied to the level of sophistication of each user to make appropriate choices. Uncaring, sloppy, or uneducated users may make password choices that are easily guessed or broken, leading to a system penetration. If the choice of possible characters to use in the password is too small, or if the overall length of the password is too short, the password may be compromisable. Even a rich character set may not be sufficient to create secure passwords if the combination of characters is restricted to an arbitrary set of possibilities. Thus, good password choice should avoid common words and names (cf. [1, 5, 7, 8, 9]).

One weakness with reusable password systems is the choice of the password. As a commonly-used example, consider the UNIX[2] password system.[8] The current password mechanism is based on a cryptographic transformation of a fixed string of zero bits, using the user-supplied password as a key. The transformation is normally an altered version of DES encryption, performed 25 times. The transformation is sufficiently slow so that exhaustive keyspace attacks are currently not practical, although fast implementations exist that can perform many tens of thousands of comparisons per second.

In UNIX, the encrypted version of the password has traditionally been kept in a world-readable file; the safety of the passwords has been protected by the time-complexity of an exhaustive attack. Thus, one of the factors in the safety of UNIX passwords is a large potential keyspace for passwords. If the full character set is used, and six-to-eight-character passwords are chosen, the number of potential passwords to be searched is far too large to be successfully searched, even at high speed. Assuming a usable character set of 120 characters, there are 43,359,498,756,302,520 ($4.34 * 10^{16}$) possible passwords of length one through eight. At 50,000 attempts per second, an exhaustive search of this keyspace would require over 27,480 years to complete.

Unfortunately, in UNIX and other systems, users often select passwords that do not exploit the large keyspace available. Instead, they choose common words and names, or simple transformations of those names. This greatly simplifies an attacker's task if these common words are searched first.

There are four basic methods for a system administrator to enforce better reusable password security on a computer system:

1. Educate and encourage users to make better choices of passwords.
2. Generate strong passwords for users and do not allow them to choose passwords of their own creation. This is often done using some random password generator.
3. Check passwords after-the-fact and force users to change those that can be easily broken with a dictionary attack.
4. Screen users' password choices and prevent weak ones from being installed.

[2] UNIX is a trademark of Unix System Laboratories, Inc.

This first method, that of educating users to choose strong passwords, is not likely to be of use in environments where there is a significant number of novices, or where turnover is high. Users might not understand the importance of choosing strong passwords, and novice users are not the best judges of what is "obvious." For instance, novice users (mistakenly) may believe that reversing a word, or capitalizing the last letter makes a password "strong." Also, no matter how good the education may be, some users may forget, or may believe that it is not significant to follow the guidelines, leading to a transient (or long-term) endangerment.

A further problem is if the education provided to users on how to select a password is itself dangerous. For instance, if the education provided gives users a specific way to create passwords — such as using the first letters of a favorite phrase — then many of the users may use that exact algorithm, thus making an attack easier.

The second method of strengthening passwords is to generate the passwords for the users and not allow them the opportunity to select a weak password. For this mechanism to work well the passwords need to be randomly drawn from the whole keyspace. Unfortunately, this method also has flaws. In particular, the "random" mechanism chosen might not be truly random, and could be analyzed by an attacker. Furthermore, random passwords are often difficult to memorize (especially if they are changed (*aged*) regularly). As a result, users may write the passwords down, thus providing an opportunity to intercept them without the effort of a dictionary search.

The third method of preventing poor password choice is to scan the passwords selected, after they are chosen, to see if any are weak. This is supported by many systems, including *deszip*, *Crack*, and COPS.[4] There are significant problems with this approach:

- The dictionary used in the search may not be comprehensive enough to catch some weak passwords. Outside attackers might scan for these choices, but the system password scanner would not include them in the search.
- The scanning approach takes time, even for a fast implementation. A lucky (or determined) attacker may be able penetrate a system through a weak password before it is discovered by the scanner. This is especially a problem in an environment with a very large number of users and systems.
- The output of a scanner may be intercepted and used against the system.

Additionally, there is not always a correlation between finding a weak password and getting it replaced with a stronger one. In many academic, business and government settings, it is difficult or impossible to force higher-level managers to change their passwords.

The fourth method, that of disallowing the choice of poor passwords in the first place, appears to have none of the drawbacks mentioned above. However, it too has difficulties associated with it. In particular, the storage required to keep a sufficiently large dictionary may prevent this method from being used on workstations and small computer systems. For instance, the standard UNIX dictionary, `/usr/dict/words`, is about 25,000 words and 200,000 bytes of space. A dictionary of 10 to 20 times that size would be necessary for reasonable protection; there are over 170,000 words in Webster's New World Dictionary, and that would occupy well over a million bytes of disk storage. That figure does not include many slang and colloquial words and phrases, nor does it include any user names, local names and phrases, likely words in foreign languages, or other strings shown to be poor password choices. A moderately comprehensive dictionary I have used in password research has over 500,000 entries, and requires over five million bytes of storage.

My full-fledged collection of dictionaries, including words in 11 foreign languages, proper names, an atlas, and a collection of slang terms, occupies over 30 megabytes of storage.

Maintaining a large dictionary is also difficult. To add new words or phrases means that the dictionary must have additional space overhead for indexing or it must be sorted after each addition — otherwise, lookups take time proportional to the length of the dictionary. In small computer environments, neither of these alternatives may be appropriate.

The OPUS project at Purdue is directed towards remedying some of the weaknesses of reusable passwords by screening user choices. OPUS is innovative because it solves the problem of using a huge screening dictionary while occupying a modest amount of disk space and computation. OPUS is designed around a compact representation of a dictionary in the form of a Bloom filter[2]. A Bloom filter is a large hashed structure with boolean values to represent its contents. Each probe value is hashed multiple times, using multiple hashing algorithms. If all corresponding bits are set, the probe represents a value in the table; any misses are definitely not known values. The filter may be tuned for accuracy or size.

In OPUS, when a user attempts to set a new password, it (and simple permutations of it) are hashed into the Bloom filter. If a match is found, the choice is rejected. OPUS can be integrated with other mechanisms, like Kerberos, and may be configured to support password aging as part of its design. Details of the structure of OPUS are not the main focus of this paper; the interested reader may refer to [12] for further design details.

To properly evaluate an implementation of OPUS, it will be necessary to check its behavior against a large body of actual user passwords. These passwords must be typical of those of a real user population that OPUS is intended to support, and they must furthermore not be unduly skewed by the manner of their selection. In particular, attempting to break actual passwords and using only those that are "breakable" does not give an accurate dataset for analysis. Thus, using a set of words produced from a study like that in [6] would be interesting, but not realistic enough for true calibration.

This requirement for real passwords means that we need a collection of password choices by actual users. However, to get these passwords means we must somehow capture "live" passwords from users as they are chosen. This form of sampling may result in a compromise of the security of the system where the passwords are collected. How is it possible to store the passwords begin collected in such a way that they cannot be sampled or broken and used to penetrate the system?

Section 2 describes the design of the password collector that was designed for this task, and some of the considerations involved in designing it. Section 3 describes some preliminary analysis of the password choices collected. Section 4 describes future work.

## 2 Design Considerations

The design of the password collection method was influenced by five significant concerns, of roughly equal importance:

- The data should be safeguarded during collection so that the collected passwords are undecipherable by anyone on the collecting system, authorized or not. That is, the passwords should be protected at least as well as by the existing password mechanism.

- The subsequent analysis of the collected passwords should pose no threat to the security of the systems involved.
- Users whose passwords were being collected should be presented with information about the collection effort and given the opportunity to opt out of the collection effort, if they so desired.
- The collection should be as complete as possible, and not skewed by selection of a particular class of user, nor influenced by existing password screening methods.
- The administrators of the systems involved should feel completely comfortable with the installation of the software and collection of user passwords.

Each of these will be further discussed in the following sections.

## 2.1 Safeguards

Answering the first concern, that of safeguarding the passwords, was perhaps the most crucial aspect of the whole process. If the collected passwords were not safe from a system cracker or accidental disclosure, the system administrators would never wish to install the collection software. If the passwords could be read by inappropriate personnel, users (especially more sophisticated users) would not want to participate in the collection effort.

At first examination, this would seem to be a simple task. The password setting programs (usually, `passwd` and `yppasswd`) run setuid to the super-user. Thus, the collection mechanism could log its output to a protected file, unreadable to any by the super-user. However, further reflection on this approach revealed several flaws:

- UNIX has been shown, time and again, to be susceptible to unanticipated break-ins. Having the passwords in a protected file might still allow them to be read by unauthorized users.
- An accident of protection might render the file readable, thus compromising all of the accounts.
- Some of the users have accounts on other machines in other protection domains. Thus, if the passwords could be read by *anyone* on one machine, it might compromise a machine in another location.
- The file would be saved to tape by the regular backup procedures, and the tape might be read or compromised by someone other than the authorized system administrators.

When all of these points were taken into account, it became clear that the only possible way of safeguarding the collected words was to encrypt them in some strong manner, with a key unknown even to the system administrator. In this manner, the words could be saved to a file and there would be no concern if that file was disclosed or read.

However, no standard private-key encryption mechanism could possibly be used. The reason for this is that the key would have to be present in the collector itself, and this could be reverse-engineered to discover it. The existing UNIX password mechanism, which uses the password itself as the key, could not be used because the results would not be decodable for study after collection.

A public key mechanism does not suffer from these problems if the private key is kept secret. For this reason, the collector was constructed to use the RSA public-key encryption mechanism.[10] Using this method allows a key (the public key) can be be disclosed without compromising the encrypted data (assuming that appropriate keys are chosen).[3] Furthermore, the strength of RSA is known (cf. [3]), and this would help assure everyone involved in the study of the safety of the collected passwords.

Using the method described in [3], two 1000+ binary bit prime numbers were generated, resulting in a modulus of 2015 binary bits. The public key was then generated, being 994 binary bits. These keys are long enough that a brute-force attack with current technology would likely exceed the lifetime of the universe (cf. the discussion in chapter 15 of [5]).

The private key was also calculated but it has not been disclosed to anyone. To prevent the keys or generation algorithms from being discovered, all calculations and storage of values was done in an off-campus location, on removable media, on a machine that had no connections to any other machine during the calculation process. The key is currently stored, in encrypted form, on removable media that will not be mounted on a machine with external connections.

Other, small safeguards were taken in the coding of the collection software to prevent disclosure or subversion of the mechanism. For instance, the code was designed to fork a child process to do the encryption and storage to the database. If a reasonable cpu time limit was exceeded, the child process would terminate. The child process would also ignore signals, and would not leave a core file if it should terminate abnormally.

## 2.2 Analysis

Obviously, protecting the passwords during collection would be an exercise in futility if they were then decrypted and used in a manner that would make them available to others. The purpose of the experiment was to collect passwords for analysis, and to eventually publish that analysis. Thus, it was important to designate the conditions under which the passwords would be decrypted, analyzed, and discussed.

After considerable discussion with the system administrators involved, the following were developed as the ground rules for any use of the data collected. A one year (12 month) limit was placed on all these restrictions following the collection of the last password (1 May 1992). This is based on the assumption that almost all of the users change their passwords once a year, and that most of the accounts monitored were limited-lifespan student accounts that expire after a semester.

The rules I agreed to are:

1. I would be the only person to ever see the private key used in the encryption. I would not disclose that key to anyone, nor put it in a position where it might be disclosed.
2. All key generation and analysis activities would be conducted on an isolated machine not connected to any other machine via phone line or network during such time as sensitive data or software was present on accessible media.
3. No specific words from the collected data would be published in any form unless they also appeared in other, well-known dictionaries and sources, and if I obtained clearance from all the system adminis-

[3] The RSA algorithm is patented in the United States. See the Acknowledgment section.

trators whose machines were monitored. After the time limit, publication of selected or representative examples will be allowed, but the full list will not be published.

4. Decrypted passwords would be analyzed only by myself, or at most by myself and one trusted graduate assistant who also agrees to abide by these restrictions. The system administrators of the machines being monitored have a "veto" over the graduate assistant, should I seek to have one assist in the analysis.

With these restrictions in place, the chief threat to the security of the accounts whose passwords were collected is through my intentional misuse of the data so collected. Although this is indeed a risk, the administrators of the systems involved believe it to be small enough as to be of no real concern.

### 2.3 User choice

As the collection mechanism was planned and tested, we decided it would be appropriate to both tell users about the research, and give them the opportunity to bypass the collection mechanism. This was decided for two major reasons.

First, although there was no compelling administrative requirement to tell users about the collection effort, we[4] believed that it was only correct to provide some notice to the users that their passwords were being collected. The collection effort itself was not intended to be secret, and we felt that giving the users information ahead of time would forestall any questions about how or why we were collecting the choices. We also felt that it was the proper thing to do.

Second, as noted in the previous section, there is some small risk associated with me misusing the collected passwords. Accounts are provided with an intent to provide users some privacy in their account usage, and the collection effort effectively rendered those accounts accessible to me, should I wish to try the list of accumulated passwords. Thus, by providing the users with an option to bypass the collection, they could have somewhat more assurance that their account was protected against any tampering by me. As some of the machines being monitored had accounts for university staff and senior faculty in several departments, this seemed especially appropriate.

Therefore, when the password collection routine was installed the original, unmodified `passwd` and `yppasswd` commands were renamed as `_passwd` and `_yppasswd` for users to execute if they wished. Additionally, the following notice was edited slightly at each site and posted to the local newsgroups:

> Your assistance is requested to aid Professor Spafford (Computer Sciences) in his research on ways of improving computer security. This research is on new ways of protecting passwords to user accounts. Previous research and experience has shown that over 80% of all computer break-ins result from poor choices of passwords, so research in this area is quite important.
>
> You do not need to do anything special to aid in this research. Just set your password as you always do, using the commands you normally use (e.g., "passwd" and "yppasswd"), when you need to use them. Don't do anything differently than you normally would. When you use the commands, your password will be changed as normal. However, something else will also

[4] The system administrators and myself.

> happen: an *encrypted* version of your password will be saved in a special database for further analysis.
>
> You do not need to worry about your password being seen by anyone else, or even traced back to you. The *only* information we will save is what you type as a password — there is no record made of your name, your account name, time of change, or other identifying information. Furthermore, your password is being mixed in with the passwords from thousands of accounts on machines at Purdue. There will be NO way to trace the password back to you.
>
> There is also no way anyone will use this information to break into your account. Professor Spafford has written the software so that it uses a strong RSA public key to encrypt the collected passwords. He is the only person with the private key necessary to read the passwords, and he will only decrypt and work with the passwords on his private machine, unconnected to the network.
>
> The system administrators at CS and PUCC have agreed to participate in this study because of the great value it has for security research. They would not have agreed to have this data collected unless they were sure that it was safe, and that Professor Spafford could be trusted in this experiment.
>
> This collection experiment will run from August 1st until May 1. At that time, the normal password programs will be reinstalled. You should plan on changing your password after May 1 (it is generally a good idea to change your password about every 6 months, anyway). If you do not change your password between August 1 and May 1, no information will be collected about your password at all.
>
> [Description of how to opt out by using _passwd or equivalent stated here.]
>
> If you wish to know more about this study, contact Professor Spafford directly, by email to spaf@cs, or visit him at the CS building. If you have other questions about system operation, contact [put your local contact info here].

Not surprisingly, no users ever contacted me or any of the system administrators about this collection effort. Checking the access times on the renamed, unmodified programs on a few systems revealed that they had not been used or were used infrequently during the collection period.

As noted in the posting, above, the collection process was written so that there was absolutely no information saved about the account when a password was changed. It would have been helpful to get some indication of whether the user involved was a novice or expert, but the general agreement was that there was no easy way to determine that information from the account alone, and that it would further help protect the safety of the users if their passwords were stored unattributed. The software was so designed.

### 2.4 Uniformity

One goal of the collection process was to have as complete and unbiased a sample of passwords as possible. This meant that we wanted to collect passwords from novices as well as experienced users, and without the effects of any existing screening mechanism. Unfortunately, there is no way to be certain that there is a bias present in such a collection. Furthermore, because we notified users that their passwords were being collected, it could be argued that this introduced a significant bias from the beginning.

We attempted to compensate for a few of the confounding effects, however. First, we built the software to be integrated into the existing commands, thus presenting users with an unmodified interface to changing their passwords. Second, the passwords were collected after they were typed but before any local screening mechanism was used (e.g., screening against `/usr/dict/words`). The combination of these two changes, and the apparent lack of use of the original programs (see the previous section), tends to indicate that any bias introduced by the collection methodology is small. There is no statistical evidence to prove this, however.

As a final design feature, the code was written so that passwords set on an account by the super-user, as might happen when an account was created or reinstated, would not be added to the collection. Thus, initial or temporary passwords would not be present and skew the collection.

### 2.5 Administrative concerns

When I first approached some of our system administrators locally about collecting passwords, they were understandably hesitant to agree. To enlist their aid required that I very clearly address their concerns about user account safety and system protection. The actions described in the previous sections helped address many of their concerns, and these items were developed during negotiations with them. However, two other major concerns remained.

Installing the collection software required changes to setuid software on the system. This could clearly present an opportunity to introduce a trojan horse into the system. Furthermore, the collection effort might potentially involve the collection of passwords to administrative accounts. Both of these were sufficient to make a cautious system administrator reluctant to install the collection software.

To answer both of these remaining concerns required very little effort. To address the first, that of installing software into setuid utilities, we distributed the changes to the software as source code and allowed the system administrators to compile, test, and install the software themselves. Thus, the system administrators were able to read the code provided and modify it as they wished for their local system.

The second concern was addressed by including a “stop list” that contained a list of all accounts that would automatically have their passwords excluded from the collection. Into this list could be put any account names that were deemed too sensitive for password collection. In this manner, if an administrative user forgot and used the modified password-changing programs, their passwords would automatically be excluded from the collection. The `root` user was added to this list by default for every machine. Each system administrator could edit that list as they wished to add or delete names.

As a final measure, we designed the software so it would collect to a local file. It was therefore up to the system administrator of each machine to decide when (and if) to provide that file to us. In this manner, if a decision was reached part way into the collection process to disable the collection, it could be done without any passwords ever being delivered to me. As a matter of convenience, when the passwords were actually collected, I did not ask for the files until the end of the collection period.

With all of these conditions and restrictions in place, we found no objection to installing the collection software and running it.

# 3 Preliminary Analysis

The collection software was installed in August of 1991 and was run until May 1, 1992. It was run on 54 machines in the Department of Computer Sciences and the Computing Center, representing approximately 7000 user accounts. Almost 19,100 password change attempts were collected. Of these, 5309 were duplicates. It took five days of computing on a SPARCstation I computer to decrypt the collection.

Because of the restrictions on publication described above, it is not possible to present a complete analysis of the words collected. However, there are some interesting statistics about the words collected, and those can be presented here.

## 3.1 Statistics

Of the 13787 unique password entries examined, the average length was 6.80 characters. The lengths of passwords are given in table 1.

Table 1: Password lengths

| Length | Quantity |
|---|---:|
| 1 | 55 |
| 2 | 87 |
| 3 | 212 |
| 4 | 449 |
| 5 | 1260 |
| 6 | 3035 |
| 7 | 2917 |
| 8 | 5772 |

24 of the passwords used meta-characters (characters where the eighth bit is set; this is normally a parity bit, or used to indicate special characters). Several of these passwords were composed solely of meta-characters. However, the way the password algorithm is designed, these characters are equivalent to the corresponding regular characters, and most of the passwords using these characters were equivalent to simple words in the dictionary.

3988 of the entries consisted of solely lower-case characters, 5259 contained a mixture of upper and lower-case, and 5641 contained at least one upper-case character. Only 188 passwords (1.4%)) contained control characters of any kind.

4372 passwords contained at least one digit. 566 passwords contained a space character (space or tab). 837 passwords used at least one comma, period or semicolon character in the password. 222 passwords contained a dash, equals sign, underscore, or plus sign. 654 passwords used at least one character present on a shifted digit key on a standard Sun keyboard. 229 passwords contained at least one of the remaining non-alphanumeric characters. These statistics are summarized in table 2.

Table 2: Character distributions

| Characters | Count | Percentage |
|---|---|---|
| Lower-case only | 3988 | 28.9% |
| Mixed case | 5259 | 38.1% |
| Some upper-case | 5641 | 40.9% |
| Digits | 4372 | 31.7% |
| Meta-characters | 24 | 0.2% |
| Control characters | 188 | 1.4% |
| Space and/or tab | 566 | 4.1% |
| . , ; | 837 | 6.1% |
| - _ = + | 222 | 1.6% |
| ! # $ % ^ & * ( ) | 654 | 4.7% |
| Other non-alphanumeric | 229 | 1.7% |

### 3.2 Wordlists

As a preliminary indication of how well the collected passwords would fare against a dictionary attack, I did some comparisons of the collected passwords against my collection of wordlists and dictionaries. All comparisons were performed against the lower-case version of all collected passwords (i.e., the upper-case letters were changed to lower-case). Furthermore, words that were all alphanumeric with a leading or trailing digit or punctuation character were also checked with those characters removed; this only accounted for 15 matches.

The first comparison was against words present in the `/etc/passwd` file on the machines being tested. Basically, this information included the user name, phone number, and account names. There were 12839 unique words derived from these files, and 592 passwords (3.9%) were found to match words in that list.

A second comparison was made against the standard dictionary on the system, `/usr/dict/words`. The version used was the one shipped with SunOS 4.1.1, and has 25144 words. 620 of the collected passwords matched words in this dictionary.

A third set of comparisons was then performed against dictionaries in 11 languages. These dictionaries have been collected from various sites on the Internet. They have not been carefully edited in some cases, and many contain typos and attempts to represent diacritical marks in plain ASCII. Also, there is considerable overlap between the various dictionaries involved. However, the majority of words in each dictionary represent standard words and names in the designated language. The results of matching against these is presented in table 3. The words were next compared against a list of names in various languages, both given names and common surnames. 1271 matches were made against this list.

As a last comparison, the collected words were matched against a large "miscellaneous" list of words derived from various collections. These words include movie names, characters from mythology, sports teams, an atlas, and many other word lists. 2498 matches were found in this comparison — the most of any other single list.

In all, 2754 of the collected passwords (20%) were quickly found using simple dictionary and wordlist lookups. From past experience, I would guess that as much as another 10% might be matched by doing more involved transformations on the collected words, such as pairing short words together, substituting the

Table 3: Matching against dictionaries

| Dictionary | Matches |
|---|---|
| Australian/Aboriginal | 133 |
| Danish | 389 |
| Dutch | 313 |
| English | 1798 |
| Finnish | 1068 |
| French | 353 |
| German | 392 |
| Italian | 1087 |
| Japanese | 626 |
| Norwegian | 368 |
| Swedish | 261 |

digit “0” for the letter “O” in words, etc. However, that analysis has yet to be conducted.

## 4 Concluding Remarks

This paper has presented the design of a password collector. The collector was designed to support analysis of a new password screening mechanism, but the collector itself has presented some interesting challenges of design. The collector uses a public-key algorithm to safely store collected passwords for later analysis. The collection process presents no observable danger to the instrumented systems. The approach used may be replicated in other environments, and may be easily extended to collect other information.

Preliminary analysis of the collected passwords reveals them to be somewhat more complex than we had first imagined. Prior to the start of the analysis, we believed we would find a large percentage of the passwords in our dictionaries and common wordlists. In part, this was because of the large number of novice users on the monitored machines, and in part because of the attitude towards passwords security that many of our users seem to exhibit. Surprisingly, only 1 out of 5 passwords were immediately found in the dictionaries. At the same time, the number of passwords containing control characters and punctuation characters is lower than we expected. Overall, however, this experiment illustrated how unconstrained user choices for passwords may compromise security, with at least 20% of all chosen passwords being weak.

More analysis of the collected words remains to be done. Additionally, the collection of password choices may now be used to calibrate a prototype of the OPUS system. Further details of this work will be reported in later papers.

## Acknowledgments

Steve Weeber did most of the initial coding for the password collection software. My thanks to all the users of CS and PUCC computers at Purdue for assisting me in the collection effort, and especially to Dan Trinkle and Kevin Smallwood for managing the collection efforts on the machines under their control. Sam

Wagstaff provided me with assistance on the algorithms necessary to generate the very large prime numbers needed.

Jim Bizdos of RSA Data Security, Inc. was kind enough to grant royalty-free use of the patented RSA public-key mechanism for this research project: my thanks to him and to RSA.

## Availability

The password collection software is available to interested parties, but the private key will not be made available. Permission to use the RSA algorithm in the software must be obtained separately from RSA Data Security, Inc. Use of the software in the USA without permission may be a violation of Federal patent laws.

Pending funding availability, OPUS is targeted as one of the first products of the COAST (**C**omputer **O**perations, **A**udit, and **S**ecurity **T**ools) Project at Purdue. OPUS will first be released to COAST sponsors, and thereafter to the general public. For further information about OPUS or COAST, contact the author.

## References

[1] Ana Maria De Alvaré. How crackers crack passwords, or what passwords to avoid. Technical Report UCID–21515, Lawrence Livermore National Laboratory, 1988.

[2] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[3] Dorothy E. R. Denning. *Cryptography and Data Security*. Addison-Wesley, Reading, MA, 1983.

[4] Daniel Farmer and Eugene H. Spafford. The COPS security checker system. In *Proceedings of the Summer Usenix Conference*. Usenix Association, June 1990.

[5] Simson Garfinkel and Gene Spafford. *Practical Unix Security*. O'Reilly & Associates, Inc., Sebastapol, CA, 1991.

[6] Daniel V. Klein. A survey of, and improvements to, password security. In *UNIX Security Workshop II*, pages 5–14. The Usenix Association, August 1990.

[7] Belden Menkus. Understanding password compromise. *Computers & Security*, 7(5):475–481, December 1988.

[8] Robert Morris and Ken Thompson. Password security: a case history. In *Unix Programmer's Supplementary Documentation*. AT&T, November 1979.

[9] National Computer Security Center. Password management guideline. Technical Report CSC-STD-002-85, US Department of Defense, 1985.

[10] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[11] Eugene H. Spafford. Observations on reusable password choices. In *Proceedings of the 3rd Security Symposium*. Usenix, September 1992.

[12] Eugene H. Spafford. Opus: Preventing weak password choices. *Computers & Security*, 11(3):273–278, 1992.

[13] Eugene H. Spafford and Stephen A. Weeber. User authentication and related topics: An annotated bibliograph. Technical Report 91–086, Purdue University, Department of Computer Sciences, December 1991.