A New Attack on Random Pronounceable Password Generators

Ravi Ganesan & Chris Davies
Bell Atlantic
Silver Spring, Maryland 20904

ABSTRACT

Given the choice, most users pick poor passwords that are vulnerable to attack. Using random machine generated passwords can ensure that 'good' passwords are chosen, but are user-unfriendly. Machine generated passwords which are 'pronounceable' represent a potential compromise between security considerations and user friendliness. Several such generators have been designed, perhaps the most prominent being the scheme developed by Morrie Gasser [5] in 1977 and which has being recently adopted as a standard by NIST [3].

The security of such generators is typically characterized by the overall size of the password space, which is typically a fairly large number. This is a fairly good security parameter, if the objective of the attacker is to try and compromise a particular *account. On the other hand, if an attacker achieves her objective by compromising* any *account(s) on the system, then the overall size of the password space, in itself, provides an insufficient characterization of the level of security. In fact, as we show in this work, the size of the password space of the pronounceable password generators we examined are fairly huge, yet all suffer from a serious weakness, which allows the attacker to compromise accounts on the system with significantly less effort than the size of the password space would suggest. The attacker cannot choose* which *accounts to compromise, but in many realistic situations, an attacker's objectives can be met by compromising* any *account(s).*

Conceptually, the password space can be thought of as a large bucket, of size K*, from which users pick passwords. It is also true that one can arbitrarily partition this bucket into several smaller buckets, perhaps of different sizes. Consider a small bucket of size* b*. It might be natural to assume that exactly* $\frac{K}{b}$ *of the users would pick passwords from this bucket. Unfortunately, in the pronounceable password generators we examine in this work, it so happens that a disproportionately large number of users pick passwords from reasonably small buckets. For instance, in the NIST standard, one such bucket contains only 0.22% of all passwords but it can be expected that about 5% of all users pick passwords from this bucket. The bottomline is that while the NIST standard claims a password space size of "5.7 billion" for 8 character passwords, an attacker who wishes to compromise any 5 user accounts on a multiuser system with a 100 users, need only search through less than 18 million passwords. The impact of the attack depends on the particular implementation and on factors such as 'salting'. Nevertheless, the generators we examined are so acutely vulnerable to our new attack, that we do not recommend that they be used.*

KEYWORDS: Dictionary Attacks, Passwords, Random Pronounceable Password Generators, Smallest Bucket Attacks.

Introduction

In this section we chronicle the motivation to use random pronounceable password generators and outline the rest of our paper.

Why Random Pronounceable Passwords?

Poorly chosen user passwords remain a major cause of security intrusions. Attackers typically attack such passwords using dictionary attacks. They obtain, either by eavesdropping on the network, by requesting from a security server (this is possible in the Kerberos[7] system) or from a file on a system, several strings each of which represents known plaintext encrypted with user passwords (e.g. in UNIX a string of zeroes is encrypted with the

user password). The attacker then attempts to decrypt these strings by methodically trying passwords from a dictionary of commonly used passwords, achieving success when a string is successfully decrypted to give the original plaintext. A related approach which uses less time (but more space) is to pre-compute the encryption of all the passwords in the dictionary, so once the strings are obtained, a simple look up is needed to obtain the user password.

There are at least three approaches to solving the problem of poorly chosen user passwords, and each has its field of use. First, smart cards or token authenticators can be used to completely replace the password. Second, proactive password checkers (e.g. [1] or [2]) can be used to filter out poor user choices and force the user to pick a good password. Finally, it is possible to have the system generate passwords for the user. The last approach has two variations:

1. Generate completely random passwords which are by definition guaranteed to be 'good'. This approach has the significant disadvantage of making the password hard to remember, and more liable to be written down (which has a security cost) or forgotten (which has an administrative cost).
2. A compromise approach is to have the machine generate a random, yet pronounceable password for the user, with the assumption that a pronounceable password is more easy to memorize, and consequently more user friendly. Such a system typically works by combining random character generation with the rules for pronunciation to generate strings which are (hopefully) pronounceable. There are at least two important aspects to the design of such a generator. First, are the passwords really pronounceable? Since the so called 'rules' of pronunciation are fairly inexact, this is an extremely subjective issue, and we do not address it further. Of more interest to us is the security of such generators. In the rest of this paper we will illustrate a serious vulnerability we have found in all the pronounceable password generators we have examined including the NIST standard. In a companion paper [4] we describe a scheme we have created which does not appear to have the same weakness.

In this paper we focus on two random pronounceable password schemes, one a scheme used in the version for the Kerberos V source distributed by Sandia National Laboratories, and the second the NIST standard [3] based on Morrie Gasser's[5] system. As we shall show, both these systems have the vulnerability we alluded to earlier. We note that we have observed the same weakness in other pronounceable password generators we have studied.

Finally, it is worth observing that while we have not been able to obtain a security analysis of the Sandia scheme, the NIST system is very carefully analyzed in [5]. In fact, the analysis is among the more complete and comprehensive of security analyses we have seen, and remains a good model of how such schemes should be analyzed. Our conclusion that the Gasser/NIST scheme is insecure (much more so now than in 1977 when it was originally released) is a recognition of the fact that all security systems are eventually broken, rather than a reflection of the care and expertise that went into the original analysis; we reemphasize that we found the original analysis rather impressive in its clarity and comprehensiveness. Also, by publishing the draft standard for a period of review and making source code for the system freely available, NIST encouraged and permitted more detailed analyses, such as the one in this paper. This in effect increases the level of security of the eventual standard, and from the perspective of commercial users of Government standards, we are appreciative of this careful and methodical approach to the standards creation process.

Overview of Paper

In Section 2 we give a generalized description of our new attack. In Section 3 we develop criterion for analyzing password systems, which are useful for ensuring that a system is not vulnerable to the attack we describe in Section 2. In Section 4 we analyze the Sandia scheme against this model and illustrate its vulnerability. In Section 5 we analyze the Gasser/NIST scheme and illustrate why it also is insecure. Finally in Section 6 we conclude by suggesting possible solutions to counter the attack.

The New Attack

Our attack is best described in the form of this simple game. Consider a two dimensional space as represented by a rectangle. Let this password space represent the space of passwords generated by a pronounceable password generator. This is the first figure in Figure 0. Each 'dot' represents a possible password generated. Here is the game:

- Player 1 has to 'color' N dots, these correspond to the N passwords picked by users.
- Player 2 (the attacker) divides the rectangle into B (not necessarily equal) sub-spaces, each containing $b_1, b_2, \ldots, b_B$ dots respectively.

If there is a sub-space i that has significantly more than $\frac{N}{B}$ colored dots, then Player 2 wins.

A moment's reflection will show that Player 1 has only one winning strategy, namely, to pick the dots to color at random (i.e. in a uniform distribution).

As we shall see in Sections 4 and 5, the random pronounceable password generators we examine have the unfortunate property that the passwords picked by users tend to be clustered in easily identifiable areas, as a consequence of which, the systems are extremely vulnerable. An attacker does not have to exhaustively search the entire password space, and instead, simply searches a small bucket which contains a disproportionate number of passwords.

How do we protect against such an attack? Firstly, as discussed earlier, an attempt can be made to ensure that users passwords are picked uniformly from all possible choices. One method of achieving this, which may not always be easy to do, is to ensure that all passwords are equally likely to be generated (e.g. see the Gasser variation discussed in the Conclusions). A second method may be to pick a system in which it is difficult for the attacker to discover a 'small bucket'. This is difficult to do, but may well be the only alternative in some cases.

Our depiction of 'dividing the password space' needs clarification. In the Sandia system the division is suggested by the way the system works, namely randomly indexing into one of twenty five buckets from which passwords are picked. In the NIST scheme we divided up the space using the first unit (i.e. passwords beginning with the unit 'a', buckets beginning with the unit 'b', etc.). Both these divisions are arbitrary, and an attacker can choose to divide up the space in any way he chooses (e.g. dividing up the space depending on the unit that appears as the fifth character). As long as a region with a disproportionate number of passwords is discovered, the attacker has succeeded.

Finally, we note that while the disproportionately arises because different passwords have a different probability of being generated, *the key to our attack lies in the aggregate result of being able to locate a sub-space with several such more likely passwords*. Consequently while it may be difficult for an attacker to make a list of 'very likely passwords' for a given system, it may be easier for her to discover a sub-space which contains several such passwords.

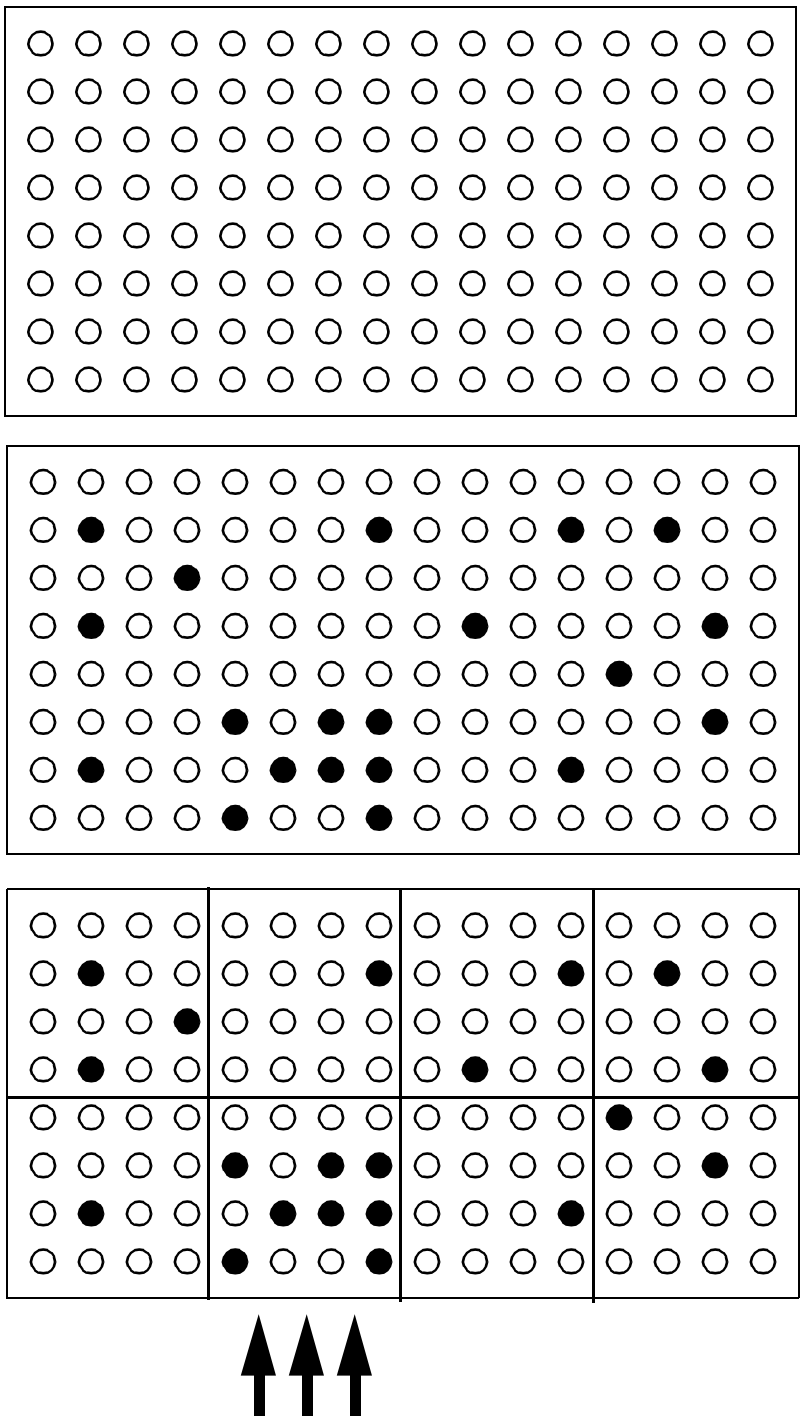


Figure 0: The first rectangle depicts the password space, with each small circle representing a password. The second rectangle shows the selection of passwords (each filled in circle) by users (i.e. the action Player 1 takes). The last rectangle shows one possible way to carve up (i.e. the action Player 2 takes) the password space. In this instance one of the regions (buckets) has a disproportionately large number of passwords, and hence it is profitable for the attacker to exhaustively search this relatively small bucket. This is a simplified example, and in Sections 4 and 5, we will see examples of this attack in which the bucket(s) which the attacker will choose are extremely small relative to the size of the entire password space.

Criterion for Analyzing Password Space Vulnerability

Criterion for the security of password spaces have been fairly well studied, and our intent here is purely to add to, not repeat/replace, well known criterion (for example those that can be found in the Orange book). Our comments apply in general to any password system, though we shall later discuss their applicability specifically to pronounceable password generators. We are mainly interested in the security of passwords in a mutlituser security system (e.g. a minicomputer environment, or a Kerberos server serving several users), where the object of the attacker is to:

Attacker Objective 1: break into ANY account(s) on the system

It is our opinion that this is a more realistic formulation of the vulnerability most systems face, than what appears to have been the traditional formulation of the attacker's motive, namely:

Attacker Objective 2: break a PARTICULAR account on the system

While it can be argued that the motives of an attacker will differ for each situation, we believe that any password system must evaluate security in terms of the difficulty of Attacker Objective 1, for two reasons. First, as stated above we speculate that it is the more common attacker motive, and second, a system secure against this attacker objective is automatically secure against Attacker Objective 2, but the converse is not true. Consequently parameterizing the system on the number of users within the 'security domain' being protected is important.

We now define several parameters and explain them:

- K the absolute size of the password space. This is the space the attacker need search in order to break into a particular user's account.
- N the number of users in the 'security domain'. The definition of 'security domain' is situation specific. Some concrete examples would be: a DEC VMS multiuser minicomputer; a network of SUN workstations and servers which use a common /etc/passwd file managed by the NIS name server, a Kerberos realm, serving an entire organization. The number of users within this domain could range from 50 users on a minicomputer to several thousand users being served by a common Kerberos server.
- T the maximum time in seconds we assume the attacker can spend on the attack. T depends on many factors including t, the time interval between which password aging is enforced. When an attacker captures strings encrypted with passwords she has a limited time to complete the attack, before the passwords change. For instance, after time $\frac{t}{2}$, it is highly likely that half the passwords captured by the attacker have changed, and by time t, all the passwords have changed. Depending on the system other factors come into play, and we shall assume that the attacker has a constant time T which she can spend on the attack.
- E the encryptions/second for the particular password scheme, that the most powerful attacker we wish to protect against can afford (or has access to). Since this "access" may well be illegal this is a hard number to calculate. Unless the attacker in question is a large organization, it may be practical to assume that the attacker

has a high end PC or UNIX server at her disposal. We realize that, for instance, specialized hardware for DES encryption has been proposed/built, but we suspect that most attackers do not (as yet) have access to these machines. The figure E can be calculated in various ways, see for instance Karn and Feldmeir's [6] analysis (naturally the actual numbers in that paper are five years old and are obsolete).

- C an implementation specific constant that captures the amount of effort the attacker must expend per user for a specific system. For instance, in UNIX, an attacker searching through a dictionary of size D, would have to, because of the salting [8], actually search through $D \times 4096$ dictionary words. In this case the constant C would be 4096. However, if the attacker is using pre-encrypted dictionaries and has enough space to store all the 'salted' variations, then, at run time, the attacker need expend no further energy (assuming that time to search a list, etc., are small compared to time for encryption) and now C is different. As is clear from this example, this constant needs to be chosen with care, after understanding both the implementation and the possible methods the attacker will use. In the rest of this paper, we will not always explicitly discuss implementation specific details, and it should be understood that individual implementations will have different values of C, and that a password space that we claim is 'small' in general may well be acceptable in implementations where C is large.

Based on these numbers we now define the criterion for protection against attack. The first criterion, which at times tends to be the only one considered by the designers of some systems is:

Criterion 1: $K \gg E \times T \times C$

i.e. they choose a password space that is large enough that it cannot be easily broken by an attacker in a 'reasonable' time. Gasser's analysis adds two closely related, very useful criterion, namely:

Criterion 2: The probability of occurrence of the most probable passwords in the password space should be low

So for instance, although the UNIX password space is very huge, the fact that users pick common natural language words with a very high probability implies that the system by itself becomes vulnerable to dictionary attacks and does not meet this criterion. Gasser discusses this criterion in the context of pronounceable password generators, where as he points out, it is no use if the overall key space K is very huge if a few passwords have a very high probability of being generated, and are generated very frequently by the system. A closely related criterion, which appears to be implicit in Gasser's discussion of the password probability distribution is:

Criterion 3: It is highly desirable that all passwords in the password space be roughly equally probable

This is really a generalization of Criterion 2, and ensures that there does not exist a subset of the password space which can be attacked in lieu of the entire space. Meeting Criterion 3 appears to be difficult (for instance, it is impossible to alter the NIST standard to meet this criterion), and attempting to meet Criterion 4 and 5 might be more realistic.

Criterion 4: In a N user system with a password space of size K, the attacker should have to search through, on average, a password space of $\frac{K}{N}$ in order to break into any one account. Consequently $\frac{K}{N}$ needs to be sufficiently large. This can be expressed[1] as: $\frac{K}{N} \gg C \times E \times T$.

We recommend Criterion 4 be used in place of Criterion 1 since any system meeting Criterion 4, will by definition meet Criterion 1, whereas the converse is not true.

Finally, for pronounceable password generators to avoid the general problem described in Section 2, we describe another criterion.

Criterion 5: It should not be possible to divide the password space into B buckets $b_1, b_2, \ldots, b_B$, with the probability of users picking passwords from the respective buckets being $p_1, p_2, \ldots, p_B$, such that there exists one or more buckets where $p_i \gg \frac{|b_i|}{K}$.

Meeting Criterion 5 is a necessary (but not sufficient) condition for meeting Criterion 4. Further, observe that in cases where Criterion 3 is not met (most realistic cases) meeting Criterion 4 and 5 appear to be fairly good substitutes.

We now turn our attention to the Sandia scheme and the Gasser/NIST systems and illustrate why they do not meet Criterion 5, and consequently Criterion 4, and hence suffer from serious vulnerabilities.

Vulnerabilities in the Sandia System

The system we refer to as the “Sandia System” is a pronounceable password generator distributed by Sandia Labs along with their version of the Kerberos V source code. As of the time of writing this paper we have not been able to obtain further information on the antecedents of the scheme (it appears to have been originally developed by IBM) and we assume that it is used widely within Sandia. We used the files 7c1Cpwd.c and 7c1dpwd.c in Sandia’s Kerberos V distribution as our source.

The scheme is fairly simple and works as follows:

- 25 templates have been created to represent typical rules of pronunciation in English. For instance "cvcvcvc" is a template representing words of the form a vowel followed by a consonant followed by a vowel.....
- The templates are formed from sets representing, vowels, consonants, double vowels, ending vowels, etc.
- To generate a password, the system randomly indexes into one of the 25 templates (all 25 templates are equally likely to be picked). We refer to the templates as template-buckets.
- It then picks, at random, a password from that particular template-bucket. This is a 7 character password.

1. Since the attacker need only, on average, search through half any given space to expect to find a password, the more precise figure is K/2N.

- In order to inflate the password space, the following is done. Either 1 of 10 digits, or 1 of 26 alphabets, is randomly added to the password, to bring the total password size to 8 characters. If the eight character is a digit from 0..9, then since there are 10 choices of digits and since the digit can be added in one of eight positions, the password space is expanded by a factor of 80. Similarly, if one of the characters from A..Z is randomly stuffed in, then the effective password space is increased 208 fold.
- Users are presented with several such passwords and asked to pick one.

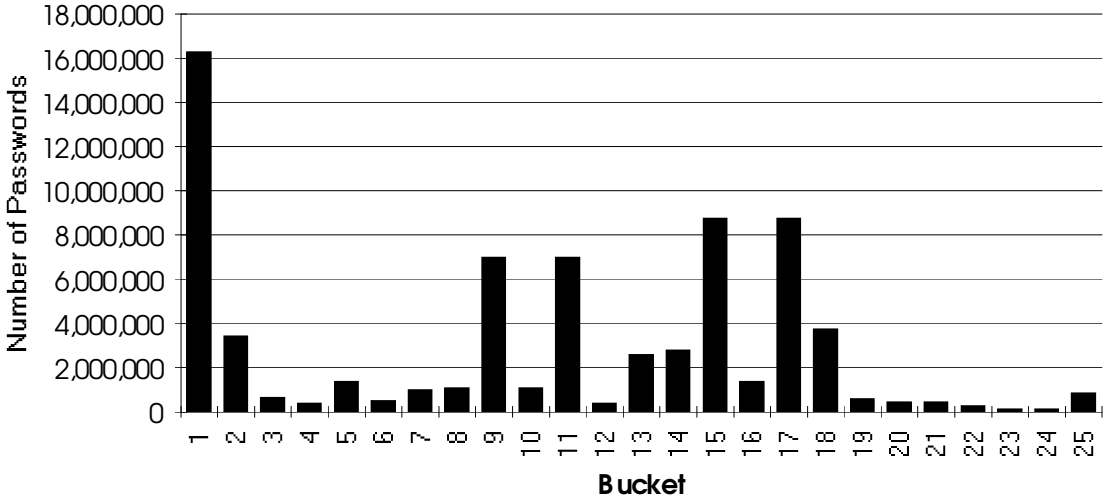
Before proceeding to our main security analysis, we wish to note two points:

1. In our purely subjective opinion, the pronouncability of the passwords generated in the first 7 characters is as good (or as bad, depending on your perspective) as any other generator. However, the addition of the eigth character, in our opinion, makes the result fairly difficult to pronounce. This is especially because the 8th character/ digit may well appear in the middle of a pronounceable syllable, thus making the entire word difficult or impossible to pronounce.
2. Presenting users with several choices and letting them pick one, introduces another filter from which passwords get selected. It is conceivable that the passwords picked by users are actually from a much smaller space than would be suggested by the system parameters. We have no way of evaluating if this is indeed the case, and assume that passwords are picked randomly from those presented to the user.

Since the 25 template-buckets are indexed into with uniform probability, it is likely that 1/25 th or 4% of all users in a N user system pick passwords from a particular template-bucket. Given the number of characters in the sets for vowels, consonants, etc., it is trivial to calculate the size of each template bucket. The size of each of the template-buckets (without the addition of the random eight character is shown below in Figure-1.

As can be seen, the distribution is highly non-uniform, with most of the passwords in a few large buckets. This dramatically changes the expected security of the scheme. The total space of 7 character passwords is 71,213,792, and after stuffing the eigth character the total space expands to an impressive 14.5 billion. However, in a 100 user system, 4 users picked passwords from the smallest bucket, which has a mere 135,800 7 character passwords, and after stuffing with the 8th character increases to a password space of 27 million. While an attacker today may balk at searching through 14.5 billion passwords, she can search a space of 27 million without too much effort in order to break into 4 user accounts on a hundred user system (from another perspective: the attacker would on average have to search through less than 3.5 million passwords to expect to break into 1 account on a 100 user system).

Figure 1: Distribution of passwords in template-buckets in Sandia scheme



In our opinion, the net result is that the system may well be more secure with user chosen passwords. For instance, in many systems (though of course not in all) systems administrators have heard enough about dictionary attacks to pick complex passwords. Whereas in using this system they may well pick passwords from small buckets, and in effect the pronounceable password generator has weakened overall security instead of strengthening it.

Finally, wish to reiterate, that the non-uniform distribution of the passwords into buckets, is the main point we wish to note, and the absolute numbers are of less interest since the number of passwords an attacker can actually try will depend on a number of other factors (e.g. salting [8] in UNIX systems).

We now turn our attention to the Gasser/NIST scheme.

Vulnerabilities in the NIST System

The NIST system is far more complex, and harder to analyze, but in effect has exactly the same vulnerability to smallest bucket attacks as the Sandia system. We provide a high level description of the functionality and refer the interested reader to [5] or [3] for more details. The scheme works as follows:

- There are 34 units, the characters A..Z (except Q), CH, GH, PH, RH, SH, TH. WH, QU and CK. Each unit has an associated probability of being picked, which corresponds roughly to the probability of its occurrence in English.

- There are a series of rules for the appearance and positioning of units, and these rules are encoded in two tables - the unit and digram tables. The former describes special rules for where the units may appear, and describes whether they are vowels or consonants, etc. The latter describes the rules according to which two units can be juxtaposed.
- To generate a password the system picks the first unit, from one of the 34 units, based on the probabilities associated with each of the units.
- The system then forms syllables by picking successive units from the list of 34, based on the rules in the unit and digram tables. These syllables are then concatenated together to form the passwords.
- Lastly, we note, that it will often happen that a particular unit that is picked will not be appropriate in a particular place. At this point that unit is rejected, and another unit is picked. If the next unit is also rejected, another unit is picked. This process is repeated 100 times, after which the entire syllable is rejected. As noted in [5] the limit of 100 is rarely reached.

The system is analyzed in [5] and [3], and the following results are obtained:

- The password space is of size 18 million for 6 character passwords, 5.7 billion for 8 character passwords and 1.6 trillion passwords for 10 character passwords.
- The most probable passwords have a low probability of occurrence (see Criterion 2).
- The probability of occurrence of most passwords are (very roughly) equal.

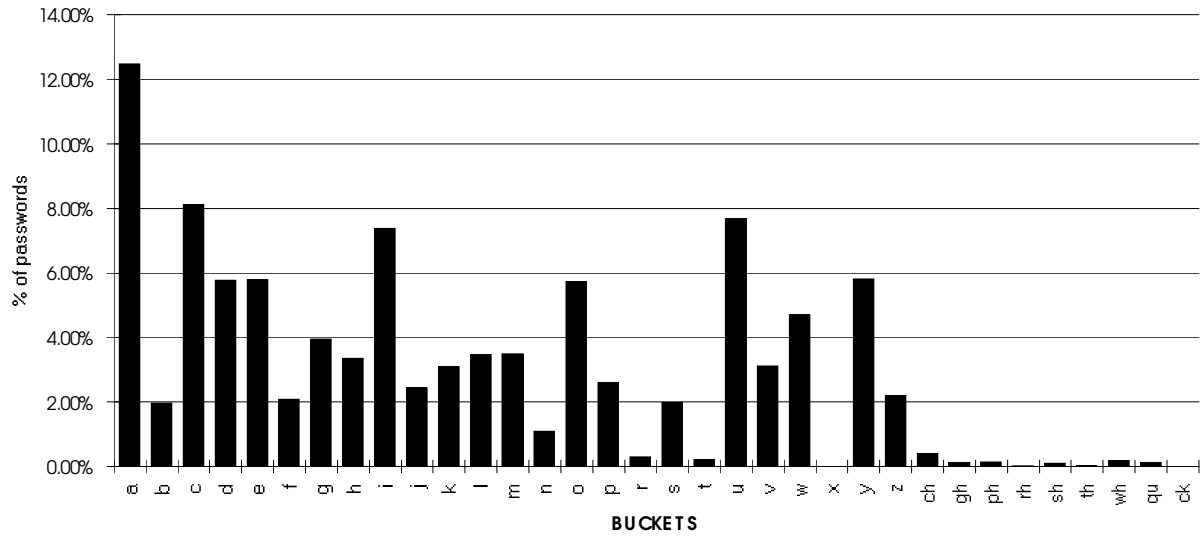
In the Sandia scheme the choice of buckets needed to successfully attack the system was readily obvious. In the Gasser/NIST scheme, the choice is not so obvious, and, as discussed in Section 2, several choices are possible. As it happens, our first attempt at discovering buckets, by differentiating using the initial unit, turned out to result in a successful attack, and consequently we did not look further. It is possible that other choices of buckets might result in a distribution even more favorable to the attacker. So in our attack on the Gasser/NIST scheme each unit represents a bucket of passwords and there are 34 such buckets. However, unlike the Sandia scheme which randomly indexes into the buckets, in the NIST scheme the probability that a user picks a password from a particular bucket is determined by the probabilities associated with the individual unit (since it is very unlikely that a password will not be completed once the initial unit is picked).

Unlike the Sandia scheme it is not easy to directly calculate the distribution of passwords into buckets in the NIST scheme. To calculate this distribution, which is shown below in Figure-2 we resort to the technique Gasser frequently uses in his analysis. In this technique passwords are generated completely at random, and are then passed through the system acting in filter mode, where it determines if the password could have been generated by the generator. As the passwords were generated randomly, sorting the sample into buckets will reflect the actual distribution of passwords into buckets.

Clearly, like in the Sandia scheme, the distribution of passwords is highly non-uniform. However, unlike the Sandia scheme, all the buckets themselves are not equi-probable. Rather, the probability that a bucket is chosen by the system is tied to the probabilities assigned to the individual units. Figure-3 juxtaposes the distribution of the passwords into

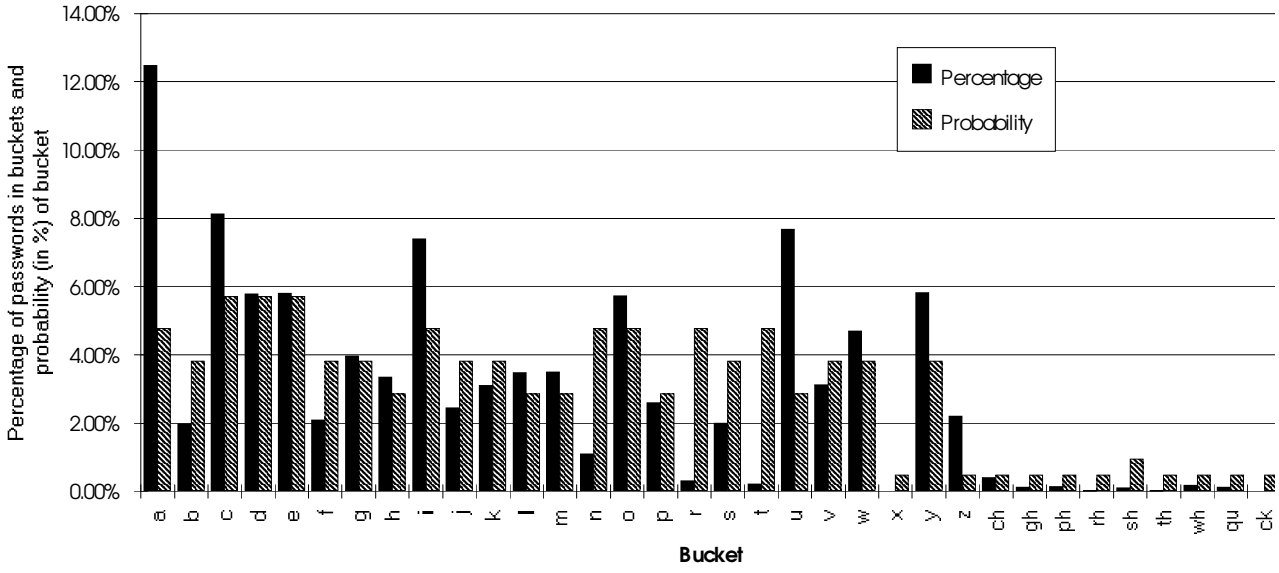
buckets with the probability of a particular bucket (which we can assume is equal to the probability that a user will pick a password from a particular bucket).

Figure-2: Distribution of passwords into buckets in NIST standard.



As can be seen from Figure 2 there are several small buckets (the buckets for R, T, X, GH, SH, TH, QU and CK) but Figure 3 suggests that rather than the smallest bucket itself, it is more beneficial for the attacker to attack the small buckets with a relatively high probability of being chosen (e.g. the buckets for R and T). It is likely that slightly less than 5% of users will pick passwords from the R bucket and another 5% from the T bucket. Yet the size of the R bucket is a mere 0.31% of the overall password space, and that of the T bucket, a mere 0.22%. Consequently, though the NIST standard would suggest a security of a password space of 5.7 billion, for 8 character passwords, an attacker can expect to break into 4 accounts of a 100 account system after searching through a mere 12.5 million passwords, and can expect to break into one account, on average, after searching 1.6 million passwords! Once again, as in the Sandia scheme, we stress that the absolute numbers are less relevant than the non-uniformity in the distribution. The actual numbers will change depending on factors like salting, but the existence of the very small buckets creates a serious weakness, which causes us to suggest that the scheme not be used without some ‘fix’ to the problem.

Figure 3: Juxtaposition of percentage of passwords in each bucket with probability of the bucket being chosen.



Conclusions: Protecting Against Smallest Bucket Attacks

Here are a number of options to protecting against smallest bucket attacks, listed in no particular order:

1. Use a generator which distributes passwords into buckets uniformly. One such design is described in a companion paper[4].
2. Use the Gasser variant: In this system, passwords are generated at random, and passed through the generator acting as a filter. The distribution of passwords into buckets is still non-uniform, but the probability of a user picking a password from a bucket is exactly equal to the ratio of the size of the bucket to the overall key size, thus meeting Criterion 5. Gasser proposed this system but pointed out that it is somewhat inefficient. Since 1977, when Gasser proposed this method, however, the speed of computers has increased to the point where the inefficiency is a non issue. However, we are extremely reluctant to recommend this as a fix for three reasons. Firstly, we describe the complexity of the Gasser scheme, from a security analysis perspective. We believe that to be 'certifiably' secure, a system must be as simple as possible, so that the design and implementation can be readily analyzed and tested. The Gasser scheme does not meet this test. Secondly, password generators should

be 'portable' across multiple human languages, without requiring significant redesign and recertification. The Gasser scheme is largely English specific. Lastly, in our purely subjective opinion, passwords generated by the Gasser scheme are not sufficiently pronounceable (relative to some other generators we have seen). Since the designs described/referenced in [4] (one our own design, and one a design available from a major manufacturer) are automatically portable to all languages and, in our subjective opinion, generate more pronounceable passwords, we recommend these be considered as serious alternatives to the Gasser variant. We wish to observe however, that having invented one of the suggested alternatives, we may be prone to some amount of bias!

3. Use a password length which makes even the size of the smallest bucket an attacker can find large enough. We have not experimented with 10 character passwords for which the key size is stated to be of size 1.6 trillion. However, assuming the distribution of passwords into buckets is similar to that of 8 character passwords, the attacker would have to, for a 100 user system, on average have to search through 440 million passwords. This may be acceptable. We do not however, recommend this option since subjecting users to 10 character passwords seems a high price to pay for an effective key space size of a mere 440 million.
4. The scheme itself can be fixed to ensure uniformity of bucket sizes by changing the rules in the unit and digram tables. Given the complexity of the Gasser scheme, we would expect that this would be extremely difficult to achieve.
5. It is probably worth investigating the effect of the following experiment: Instead of picking units according to their probability of occurrence in English, pick units with a uniform distribution (i.e. each of the 34 units has a $\frac{1}{34}$ probability of being picked. Since the rules of pronunciation are determined by the digram tables, this will not affect the pronouncability of the resulting passwords. We suspect that this will not solve the problem, but it is an interesting experiment since, in our opinion, the probabilities tied to the units serve no purpose anyway, and a simpler system is easier to analyze.
6. As a quick 'fix' we recommend completely removing all the very small buckets. Observe that this does not solve the problem, and only succeeds in improving the odds against the attacker slightly. The attacker can then switch his attention to one very large bucket, and then focus on the small buckets within that large bucket, e.g. passwords beginning with AR or AT. It is this complication that leads us to recommend that an alternate, easier to analyze, system be used.

Our bottomline recommendation is that NIST adopt a simpler more readily analyzable system which is designed to meet the five Criterion we identified in Section 2.

Acknowledgments

We are extremely grateful to Andy Goldstein of DEC for providing us with the 10 million passwords that we used to perform our analysis of the NIST scheme. We thank Chuck Dinkel of NIST for providing us with useful information on the NIST standard. We are deeply grateful to Raymond Pyle of Bell Atlantic for his insightful review and encouragement. Finally, our gratitude to the anonymous referees for forcing us to carefully rethink our work, and hence make it more coherent.

References

[1] Bishop, M., “Proactive Password Checking”, *4th Workshop on Computer Security Incident Handling*, August 1992.

[2] Davies, C. and R. Ganesan, “BApasswd: A New Proactive Password Checker”, *16th National Computer Security Conference*, September 1993.

[3] FIPS PUB 181, “Automated Password Generator”, Federal Information Processing Standards Publication. 1993.

[4] Ganesan, R., “BApronounce: A New Random Pronounceable Password Generator”, Submitted for publication.

[5] Gasser, M. “A Random Word Generator for Pronounceable Passwords”, *National Technical Information Service (NTIS)* AD A 017676.

[6] Karn, P.R. and D.C. Feldmeier, “UNIX password security - Ten years later”, *Advance in Cryptology - CRYPTO 89. G. Brassard (Ed.) Lecture Notes in Computer Science*, Springer-Verlag. 1990.

[7] Kohl, J., C. Neuman and J. Steiner, “The Kerberos Authentication Service”, *MIT Project Athena (October 8, 1990)* Version 5.3.

[8] Morris, R. and K. Thompson. “Password Security: A Case History”, *Communications of the ACM*, 22(11). November 1979.