Password Exhaustion: Predicting the End of Password Usefulness

Luke St.Clair, Lisa Johansen, William Enck, Matthew Pirretti, Patrick Traynor, Patrick McDaniel, Trent Jaeger
`lstclair, johansen, enck, pirretti, traynor, mcdaniel, jaeger@cse.psu.edu`

Abstract

Passwords are currently the dominant authentication mechanism in computing systems. However, users are unwilling or unable to retain passwords with a large amount of entropy. This reality is exacerbated by the increasing ability of systems to mount offline attacks. In this paper, we evaluate the degree to which the previous statements are true and attempt to ascertain the point at which passwords are no longer sufficient to securely mediate authentication. In order to demonstrate this, we develop an analytical model for computation to understand the time required to recover random passwords. Further, an empirical study suggests the situation is much worse. In fact, we found that past systems vulnerable to offline attacks will be obsolete in 5-15 years, and our study suggests that a large number of these systems are already obsolete. We conclude that we must discard or fundamentally change these systems, and to that effect, we suggest a number of ways to prevent offline attacks.

1 Introduction

Password-based authentication mechanisms are the primary means by which users gain legitimate access to computing systems. Because of their central role in the protection of these systems, the vulnerabilities inherent to these methods have long been known throughout the security community. The best known of these vulnerabilities is password choice. A variety of studies [17, 20, 26] cite the lack of *entropy*, or unpredictability, included in each password as the root of the problem. Because of the chronic under-use of the available key space, as many as 30% of user passwords are recoverable within a period of hours [19].

The common wisdom is that if users can be educated to select "perfect" passwords, offline brute-force attacks to recover such information will remain beyond the computational ability of modern machines. In actuality, the current entropy in a perfectly-random 8 character password, the most common password length, is actually less than that of a DES key. Since DES was effectively broken by brute-force attacks in 1999 [2], this assumption is questionable. However, now we are seeing a variety of password policies request 15 character passwords. In this case, the entropy is comparable to 3DES or AES. Also, we are seeing a prevalence of password policies for guiding users to select effective passwords. Our interpretation is that the community is staking the future viability of password systems on increases in password length and policies to ensure effective use of the password space.

In addition to the future increases in computing power, the viability of password systems is limited by the entropy that people can really use in practice. Given than human beings are only capable of remembering approximately seven random items [8], an increase in password length does not necessarily mean a commensurate increase in real entropy. As passwords lengths increase, users may develop techniques to use predictable chunks of passwords randomly arranged. Also, users will be tremendously challenged to memorize multiple passwords of such length.

In this paper, we investigate two fundamental claims: (1) near-term increases in available computing power will soon enable brute-force cracking of perfectly-random 8 character passwords on a variety of easily available platforms and (2) that the maximum entropy that we can expect from a password is limited and is no more than the commonly used 8 characters we have already, thus rendering password systems that permit offline attacks obsolete. First, we use current forecasting of hardware performance to estimate the end of the computational infeasibility for offline password attacks given the entropy of an 8 character password. We find that computing power that should be easily available to a typical user will be sufficient to break a perfectly random 8 character password in May 2016. For more motivated attackers, the time to crack is very short. Second, we examine the entropy of real passwords and the impact of password policies upon that entropy. We find that in real passwords of the CSE department at Penn State, the entropy is only slightly better than a rather dire analysis done by NIST of the entropy available in real passwords. Further, we find that password policies significantly limit password entropy and do not appreciably improve the protection of passwords. No solution is known to exist that can save password systems that are susceptible to offline attacks from obsolescence in the near future.

The remainder of this paper is organized as follows: in Section 2, we discuss predictions of future computer performance and their bearing on password vulnerabilities; Section 3 examines the ways in which entropy is actually removed from systems and revisits the above predictions; Section 4 considers solutions to this problem; Section 6 offers concluding remarks.

2 Future of Password Recovery Power

This section considers how hardware improvements and processor availability impact the security provided by password authentication systems. We begin by introducing a model of computing used to assess the vulnerability of password systems to offline attacks. Using this model, we postulate the present and future security of popular password systems.

2.1 Forecasting Model for Password Recovery

This section introduces a model assessing the viability of current and future password systems. We investigate the impact of increasing processor speeds and parallelism on brute-force attacks. These factors, modeled as functions $s(t)$ and $p(t)$, are based on future computing predictions made by experts within each field. We conclude by analyzing the number of systems available to attackers.

2.1.1 Model Definition

Our model is composed of the following components: password space, processor performance, parallelism, and system size.

Password Space(c): The password space is the set of all possible passwords that a system can represent. In terms of password recovery, the password space indicates the average amount of work required to recover a password. Given the limitations of human memory, we shall assume a typical user password is composed of 8 characters, where each character can be any of the 95 characters readily represented with a keyboard. In the best case scenario (from the point of view of system's security), user passwords will be uniformly distributed across the password space. Thus, an adversary on average must search half of the password space to recover a password. We represent the average number of tries required to recover a password as:

$$c = 95^8/2 \approx 3.3 \times 10^{15}, \tag{1}$$

where each attempt to break a password is termed as a *try*.

Processor Performance($s(t)$): Processor performance models the amount of work that can be accomplished by a single processing element. To avoid ambiguity, this factor specifically measures the amount of work a single processor core can perform. To map this factor to password recovery, we shall define processor performance as the number of seconds required to perform a single try. We denote processor performance as a time varying function $s(t)$. In Section 2.1.2 we consider several models that predict how processor performance will change with time.

Parallelism($p(t)$): Parallelism models the increasing prevalence of processor replication in contemporary computing systems. This factor measures the number of processor cores present within a single computer. The password space can be subdivided into disjoint components and independently processed by different processor cores. The level of parallelism present in a given machine greatly increases the rate at which the password space is examined. For instance, a machine with 4 processing cores can simultaneously perform 4 tries. We denote the level of parallelism present in a given computer as a time varying function $p(t)$. In Section 2.1.3 we consider different models that have been used to forecast the number of independent processing cores present within a single computer.

System Size(z): System size models the increasing prevalence of computational devices. For instance, the number of computers in homes is steadily increasing [12]. Further, the number of computers present in computing clusters is quickly growing. Finally, the overwhelming size of botnets is increasing. To capture this trend we represent the number of independent computers present in a system as z.

Password Cracking Forecasting Model($T(t)$): Given our definitions of $z, p(t), s(t)$, and c we can now introduce our model of forecasting how the computing trends of increasing processor performance and increasing parallelism will effect the viability of brute-force password cracking attacks. Our model represents the amount of time required to recover a random 8 character password on a computing system that is characterized with by $s(t), p(t)$ and z:

$$T(t) = \frac{(3.3 \times 10^{15}) \cdot s(t)}{p(t) \cdot z}. \tag{2}$$

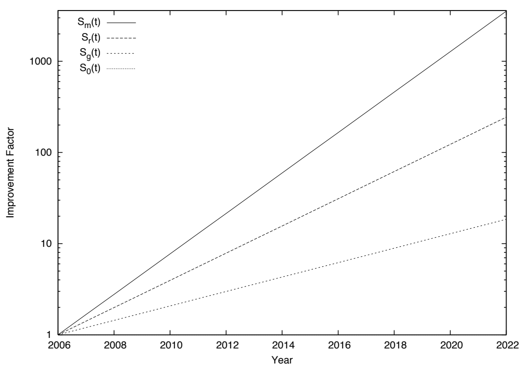
2.1.2 Predicting Processor Performance $s(t)$

This section examines predictions on the growth of future processor performance as made by experts in the field. It then defines the function for this growth, $s(t)$, over time which is used in our model.

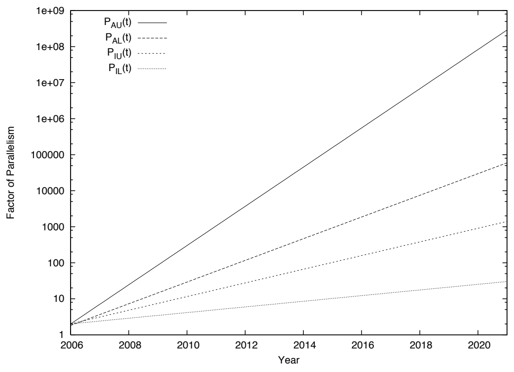
Determining future processor performance has been a widely studied problem for more than 50 years. Moore stated in 1965 that chip density will double every 12 to 18 months. This prediction, called Moore's Law, is the standard for the measurement of future computing power. Unfortunately, chip density is reaching its limits due to incredible heat and power consumption [10]. Because of these challenges, the industry is looking towards other methods to increase overall computing power instead of focusing on clock speed. Nanotechnologies, compiler optimization and other innovations are being consider as approaches for increasing chip performance [10, 16]. However, the industry are still looking to Moore's law, not as an estimate of future chip density but, as a prediction of future computing power [16].

Shown in figure 1(a), Moore's Law is our function, $s_M(t)$. However, studies have shown that the rate of performance growth proposed in Moore's law is unrealistic [14]. This study states that, over the past 7.5 years, the actual rate of computer performance growth has been closer to 41% per year. This more conservative predictor serves as a second function for performance growth, $s_R(t)$.

However, because we seeing the limits of physics in applying traditional methods for performance improvement, $s_M(t)$ and $s_R(t)$ may be unrealistic. Because chip density reaching its limit, there has been much discussion that Moore's law is no longer valid [13]. Experts are beginning to doubt that processor power is going to continue to grow at rates that we have seen in the past. Taking this into consideration, we define two more functions, $s_{SG}(t)$ and $s_0(t)$, to more conservatively project the growth of processor performance. The first function, $s_{SG}(t)$, is an estimate for slowing growth. We choose 20% for the value of growth per year for this function because it is half of the researched rate of performance growth, $s_R(t)$. The second function, $s_0(t)$, assumes no growth over the next 15 years. All four of our processor performance growth functions are plotted in figure 1(a).



(a) Processor Performance



(b) Parallelism Impact

Figure 1: Future Computing Performance Estimates

2.1.3 Predicting Parallelism Factor $p(t)$

Parallel computing is popularly seen as a counterbalance to slowing growth of processor performance. Multi-core technologies, multiple processors and hyper-threading have been proposed to increase performance. Intel and AMD have released estimates for the amount of parallelism that they expect will exist in a single computer in the near future. Intel believes that a processor will have anywhere from tens to hundreds of cores within ten years [10]. AMD projects that processors will contain more than 2 cores by 2007 and more than 8 by 2008 [1].

Given these estimates, we predict where parallelism will be in the near future. We extrapolate two functions for AMD's estimates and two functions for Intel's estimates. The first function of parallelism growth, $p_{AU}(t)$, is based on AMD's upper estimate of the number of processors available in a single computer while $p_{AL}(t)$ is their lower estimate. Similarly, Intel's upper estimate defines the function $p_{IU}(t)$ and their lower estimate defines $p_{IL}(t)$. From the graph of these four functions shown in figure 1(b) we see that AMD's estimates, $p_{AU}(t)$ and $p_{AL}(t)$, are the upper approximations for parallelism growth while Intel's estimates, $p_{IU}(t)$ and $p_{IL}(t)$, establish the lower bound.

2.1.4 System Size z

Another way to gain parallelism in the password recovery computations is to use multiple computers. The more computers that are used, the less time it takes to recover a password. A system of computers can consist of personal computers, clustered nodes, or large networks like botnets. Any personal computer user can own any number of computers. In the case of a computing cluster, the size of these systems can be much larger. Sizes of 20 to 500 nodes are typical for a computer cluster. Large networks of computers provide the largest factor of parallelism when executing one task. A botnet, a large number of compromised machines that can be remotely controlled by an attacker, is an example of such a network. Botnets can range in size from thousands to hundreds of thousands of computers. This growing availability of computers directly affects the amount of parallelism available to any user.

2.2 Future Password Recovery

This section shows the extent to which computing performance improvements threaten the security of password authentication systems as determined by our model. We begin by briefly describing the password systems that we attempt to break and the time to password recovery for each system on today's commodity hardware. The processor performance, $s(t)$, parallelism factor, $p(t)$, and system size, z, of the future computing systems used in the experiment are defined. The ability to recover passwords of these future systems is then evaluated in figure 2 and discussed.

In our experiments, we analyze three password systems: Unix crypt, MD5 crypt and Kerberos. Unix crypt and MD5 crypt are used to authenticate users in Unix/Linux systems. Kerberos is a single-sign-on network authentication system. For more information about these password systems see the appendix. We ran password recovery software on commodity hardware to determine the speed of tries for each system. The results from these experiments, illustrated in table 1, were used as input to our model.

System	Tries/Sec
Unix Crypt	1557890
MD5 Crypt	17968
Kerberos	55462

Table 1: Password Recovery Speeds

For the following analysis, we posit one possible future computer type. Given the fact that chip technologies are reaching the limits of power and heat, the slow growth processor performance function, $s_{SG}(t)$, is used. In order to determine a median parallelism factor, we take the average of the two middle values; AMD's lower estimate and Intel's higher estimate. This results in a function, $p_{AVG}(t)$, that characterizes the average of $p_{AL}(t)$ and $p_{IU}(t)$. We analyze the impact of this computing power as it exists within these systems of the future.

The number of computers within a system can range anywhere from 1 to hundreds of thousands. We examine a personal computer system, clusters, and botnets. A personal computer system consisting of 2 computers, equation 3, clusters of 10 and 100 nodes, equations 4, and botnets of 1000, 10000, and 100000 compromised hosts, equations 5, are analyzed. Thus we use the following models of future computing in our experiments:

$$T(t) = \frac{(3.3 \times 10^{15}) \cdot s_{SG}(t)}{p_{AVG}(t) \cdot 2}. \tag{3}$$

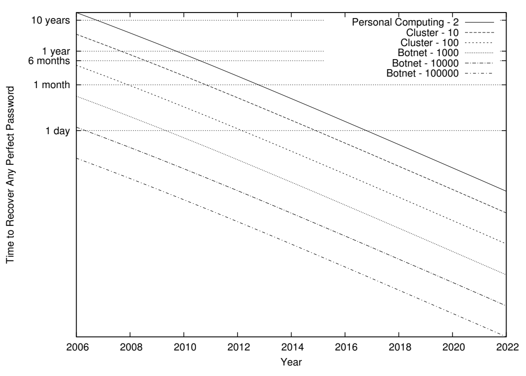
$$T(t) = \frac{(3.3 \times 10^{15}) \cdot s_{SG}(t)}{p_{AVG}(t) \cdot 10}, T(t) = \frac{(3.3 \times 10^{15}) \cdot s_{SG}(t)}{p_{AVG}(t) \cdot 100}. \tag{4}$$

$$T(t) = \frac{(3.3 \times 10^{15}) \cdot s_{SG}(t)}{p_{AVG}(t) \cdot 1000}, T(t) = \frac{(3.3 \times 10^{15}) \cdot s_{SG}(t)}{p_{AVG}(t) \cdot 10000}, T(t) = \frac{(3.3 \times 10^{15}) \cdot s_{SG}(t)}{p_{AVG}(t) \cdot 100000}. \tag{5}$$

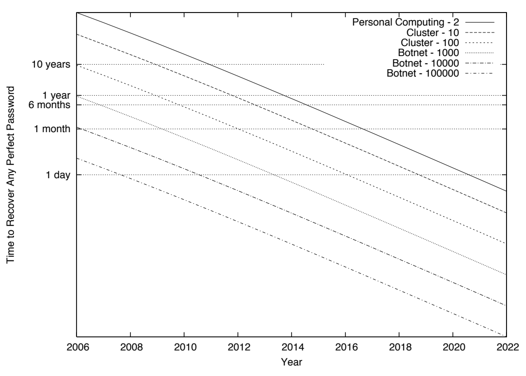
Beginning with the initial values presented in table 1, we were able to determine, for each password system, what each computer system was capable of recovering over the span of 15 years. The results from these experiments are presented in figure 2.

Examining our graphs, we draw conclusions about when a system's passwords will be recoverable by a single computing system. The most apparent result is that a botnet with 10,000 or more compromised computers is able to recover any password from any system in under 6 months today. In less than five years, any botnet with at least 1,000 compromised computers can recover any password in under a month.

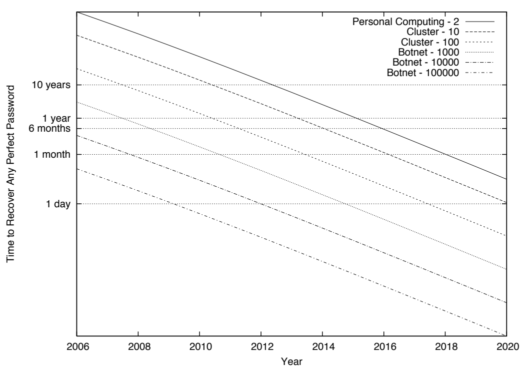
Given a smaller system, like a cluster, we see that password recovery, naturally, takes longer. An average sized cluster is able to recover any Unix crypt password in under 6 moths today. However, within only 8



(a) Unix crypt



(b) Kerberos



(c) MD5 crypt

Figure 2: Future Password Recovery Analysis

years, a cluster of minimal size will be able to recover any password from our three presented password systems!

Examining our extreme case, personal computing systems, we recognize the most startling results. Within three years, any Unix crypt password will be recoverable in under 6 months . Unix crypt is obviously broken. The more devastating result is that any password from any of the evaluated systems is recoverable in under 6 months by a single personal computing system in 10 years. This means that our trusted authentication systems will be vulnerable to raw computing power from the comfort of your own home before you say ”Happy New Year” to ring in 2017.

3 Passwords in Practice

In this section, we show that the current state of password security is actually much worse than the theoretical model presented in Section 2.1.4 suggests. The preceding model takes only future hardware improvements into consideration, while in reality, an attacker almost always does not have to try every one of 95^8 possible passwords. For example, password policies serve to reduce the amount of work an attacker is required to do to recover perfect passwords. However, these perfect passwords are often not even used; in practice, users choose passwords that contain much less than their maximum allowable entropy. We demonstrate the degree to which this is true by showing that passwords are recovered at a much greater rate than theoretically random passwords would be. Finally, we use previous studies to estimate the amount of entropy in the average password. While there are discrepancies between how much entropy is purported to exist in passwords and how much is measured in our experiment, the likely cause is the need for improvements in software-based password recovery techniques.

3.1 Password Policy Restrictions

The results in the preceding section all assume that perfectly random passwords of memorable length were chosen that utilize the entire 95-characters commonly found on an English keyboard. However, based on the recommendations of the security community[7, 25], many sites have begun to implement password policies that restrict the types of passwords that may be chosen. For instance, some sites don't allow users to choose characters outside the alphanumeric set. Others require passwords to be between a minimum and maximum length. Still others make restrictions on the types of characters that must be present in a password. While these rules help to prevent users from choosing dictionary-based passwords, we show both that this is not effective in preventing brute-force attacks (as shown in the next section) and that it decreases the total password space from which users willing to pick perfectly random passwords are able to choose. We examine a number of policies, and graph how long it takes to crack perfectly random passwords chosen within these sets of policies against the results from section 2.1.4. In this way, we show that the protection password hashes provide against offline attacks is actually much worse than the model we have presented so far.

First, we relate the number of passwords to policy restrictions as follows. Let R_i be the set of passwords that do not satisfy a certain policy i. For instance, if policy i required users to choose a lower-case letter, $|R_i| = (95 - 26)^8$. We also define $R_i \cap R_j$ to be the intersection of passwords that do not satisfy both policies i and j (i.e, both policies are not satisfied). Now, we apply a variant of the inclusion-exclusion principle to the total password space to get the following formula, which computes how many passwords satisfy all of the policies specified:

$$\begin{aligned} |\bigcap_{1\le i\le k} \neg R_i| = (95^8)/2 \quad &+ \quad (-1)^1 (\sum_{1\le i\le k} |R_i|) + (-1)^2 (\sum_{1\le i_1\le i_2\le k} |R_{i_1} \cap R_{i_2}|) + \\ \ldots \quad &+ \quad (-1)^{k-1} (|R_1 \cap R_2 \cap \ldots \cap R_{k-1} \cap R_k|) \end{aligned}$$

The results in this section, which are summarized in figure 3, are examined under the same models of increasing computational power discussed in section 2. Since we have already demonstrated that older Unix crypt hashes are too weak to provide practical security, we instead plot the time it takes to crack Kerberos passwords these following policies. In this way, we demonstrate the degree to which attacks can be sped up in a system that would otherwise remain somewhat secure for the next few years.

The first policy we examine under this formula is from a recommendation made by the SANS Institute password policy page[7]. SANS (SysAdmin, Audit, Network, Security) is a large collaborative group of security professionals that provide information security training and certification[6]. This policy is intended to be used by businesses when they establish password policies for their enterprise networks. SANS recommends that users pick at least one upper and lower case character, 1 digit and 1 special character. Applying the formula above, this fairly typical policy reduces the number of valid passwords more than a factor of 2. This reduction in the number of potential passwords that can be chosen is shown in figure 3(a), which uses the same predictive model as figure 2(b), but with the reduced password space dictated by this policy. As a point of reference, under the full password space, a single user will be able to crack a Kerberos password in under a year in October, 2013. Under the restricted password space that SANS defines, this will happen around October, 2012. Unfortunately, SANS advocates that employees not conforming to these policies be fired, which effectively takes away the choice of users to pick a password outside the space an attacker would know to search.

The Computer Science and Engineering department at The Pennsylvania State University recently enacted a password policy applying to the password choices of all students and faculty in the department. Following the common wisdom, they used the Sun password policy mechanisms to define a policy requiring all users to have 2 upper case characters, 2 lower case characters, 1 digit, and one ”special” character. Using the previous formula, this cuts the pool of potential passwords nearly in a third, and is graphed on figure 3(b). By way of comparison, this policy restricts that password space such that a personal computer will be able to recover a Kerberos password in one year around November 2011.

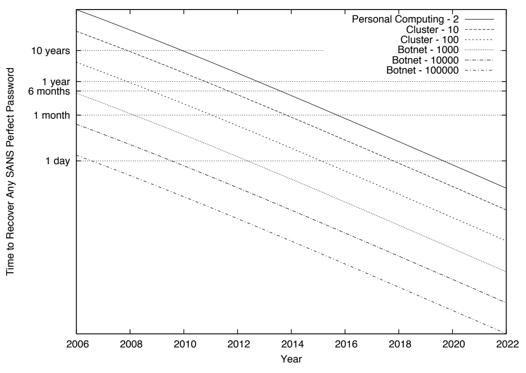
The last set of passwords are those generated with the pwgen utility. pwgen is a Unix utility which generates ”memorable” passwords of user-defined size (default is 8 characters). However, pwgen will only mix alphanumeric characters randomly to form passwords, and will not use symbols. According to the pwgen changelog[5], this was done so that passwords would be ”much more usable.” Obviously, this greatly restricts the number of potential passwords that can be chosen to the size 62^8, down from 95^8. Unfortunately, pwgen is widely used to generate ”random” passwords when secure initial or replacement passwords are needed. Obviously, this policy greatly restricts the pool of potential passwords. Figure 3(c) shows the effect this has on the time it takes for an attacker to recover a Kerberos password in a system using these types of passwords. Unix crypt, which we have shown to be inadequate for protecting password hashes, is much easier to run password guessing attacks against due to its DES-based encryption algorithm. Kerberos, however, is generally much more difficult to crack, as shown in 1. However, compared to the time it takes to recover a perfect Kerberos password, pwgen-generated passwords can be recovered approximately 30 times faster, making them approximately as weak as older Unix crypt hashes without password restrictions. This demonstrates how poor policy decision can negate large benefits in algorithm choice. This policy, the most restrictive of the three, creates a situation in which a single PC can crack a Kerberos password in 1 year around June, 2009.

We conclude that password policy, while useful for eliminating the weakest of passwords, can severely restrict the pool of passwords attackers must search. Requiring users to pick only a subset of potential passwords can drastically reduce the password space provided by perfectly random passwords in that system. The time to crack a perfect password is reduced by up to a factor of 30, as shown in figure 3(c). It is important to note that when password policies are enforced, this sizable restriction is applied to each and every password, no matter how random the user’s password is.

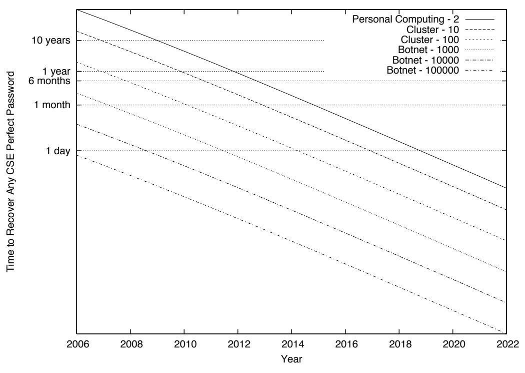
3.2 Real Passwords

Unfortunately, passwords in practice are actually much worse. Users find clever ways to circumvent password restrictions, which allows them to pick words containing very little entropy. They often pick words with obvious letter replacements, like ”0” for O, that can be easily guessed, or misspellings of common words. In short, they still pick passwords that are not truly random and thus are vulnerable to intelligent password guessing attacks. For instance, most password recovery tools today contain fairly sophisticated methods of guessing variations on words out of the dictionary, such as trying words with numbers appended, words with numbers prepended, common symbol-for-letter replacements, and more. Then end result of this is that figure 3 is still much too conservative when applied to actual passwords. While this does not apply to every password, as do the results in section 3.1, we demonstrate that it does apply to a large number of them.

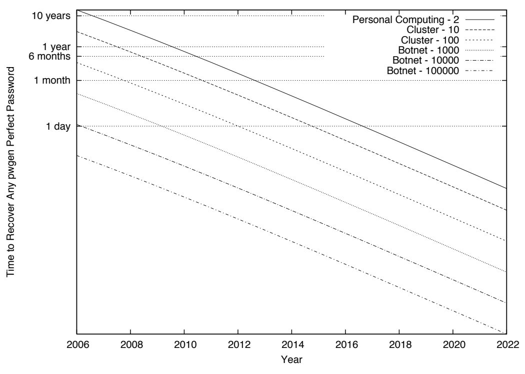
This is partially due to the fact that user passwords not based on dictionary words often still contain little randomness. Users are much more likely to pick certain letters than others, for instance[22]. Therefore, if



(a) SANS Password Policy



(b) CSE Password Policy



(c) pwgen Passwords

Figure 3: Password Recovery with Policy Restrictions

passwords are created using much more commonly used characters like lowercase letters, and do not often contain hard-to-reach letters like "+", they create passwords that contain a great deal less randomness than they could. Unfortunately, password recovery tools have begun to take advantage of this vulnerability in passwords. Some password crackers now have an intelligent brute-force mode which tries all possible passwords, but in order of increasing likelihood. For instance, the string "bgtyae1T" would be tried long before "t,I}&[*v". One way this is done in practice is to take a file representing the character frequency in a given language and using that information to try the more common characters in that language with greater frequency in password guesses.

In order to evaluate the severity of these problems, we grabbed the password file for the entire Computer Science and Engineering Department at Penn State University via "ypcat passwd > passwd," as described in Appendix A. Intuitively, this simply asks the central authentication authority for the password hashes of all users, which are then stored in a file for offline cracking. This netted password entries for 3500 users with valid passwords in less than a second. We then scripted the automatic submission of password cracking jobs to a cluster of 20 nodes with dual Opteron 250 processors. We used 16 of the 20 nodes for brute-forcing passwords based on character frequencies, and 4 nodes for trying passwords derived from dictionary words. John the Ripper was used as the password recovery tool, since it is widely available and has good support for both dictionary and brute-force based attacks. We ran this program for only 5 days, with the number of passwords recovered as a function of time graphed in figure 4.

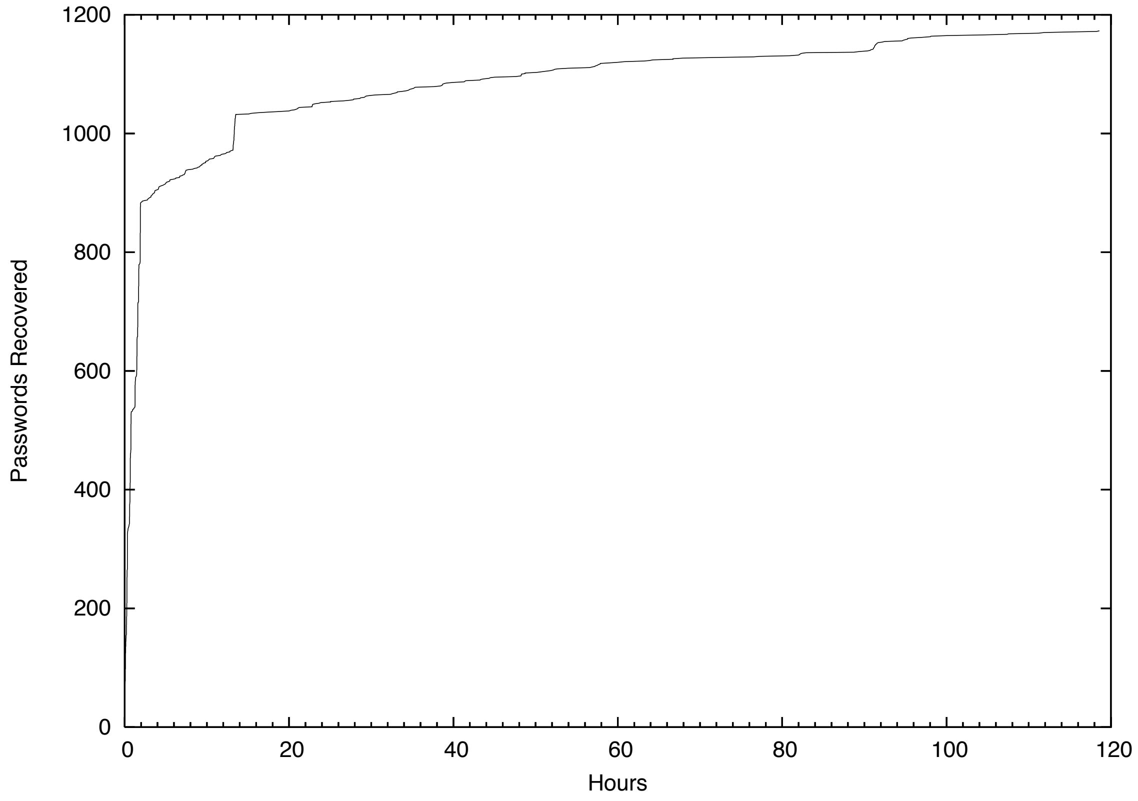


Figure 4: Time to Recover Penn State CSE Passwords

One of the most startling observations that can be drawn from this graph is that approximately 25% of all passwords were cracked in the first 2 hours. This particular password deployment even had password policies in place on some of the passwords, as mentioned in the preceding password policy section. It is particularly interesting to note that only 10.1% of the passwords recovered were recovered as a result of guessing words based on dictionary words. The remaining 1118 passwords were all recovered by the nodes performing a brute-force attack. The brute-forcing method found in John the Ripper, which we used for these experiments, is clearly able to recover a large number of non-dictionary based passwords. This is a somewhat surprising result, as common folklore in the security community seems to suggest that the biggest password problem is commonly used or dictionary-based passwords. Instead, this may reflect growing consciousness about the weakness of those passwords, but a persistent unwillingness or inability of users to pick passwords uniformly from the potential space of all possible passwords.

3.3 Perfect Password Crackers

The reality of password strength may be even worse than this, however. According to the National Institute of Standards and Technology (NIST), passwords in practice are much weaker than the theoretical maximum discussed above[26]. Based on experimental findings, NIST has given some guidelines as to the practical entropy provided by passwords that users choose. We will evaluate the time to crack passwords in practice using a few entropy guidelines from NIST. Each bit of entropy multiplies the possible password space by 2 (e.g., 5 bits of entropy means there are 2^5 possible passwords). NIST specifies that the first character of a password gives 4 bits of entropy, and successive characters give 2 bits apiece in a system where a user may chose any password they like. We will call this the *permissive policy*. In many systems, however, users are forced to choose both upper-case and non-alphanumeric characters. This nets a password with an additional 6 bits of entropy, according to NIST. We will refer to this as the *special-character policy*. If passwords are checked against an exhaustive dictionary of at least 50,000 words (a factor of 10 less than the dictionary searched in the preceding results), 6 bits of entropy are added. Two new policies can be constructed, the *dictionary policy* and the *special-character and dictionary policy*, when the dictionary is applied to the permissive and special-character policies, respectively. The special-character and dictionary policy represents the strongest passwords that the government identifies, as far as we can tell, so it is used here to demonstrate how long actual passwords should take to compromise. Note that all of these measurements assume full 8 character passwords, as the time to compromise with 7 or fewer characters became negligible. As shown in Table 2, in theory, a single machine should be able to recover a standard password in practice.

Table 2: Time to recover passwords as specified by NIST

Password Type	Bits of Entropy	Time to recover for single machine	20-node Cluster
permissive	18	< 1 minute	< 1 second
special-character	24	14 minutes	20 seconds
dictionary	24	14 minutes	20 seconds
special-character and dictionary	30	15 hours	22 minutes

The NIST estimates predict faster recovery than our experimental results show, though not by much. As Table 2 shows, the average special-character and dictionary password takes 15 hours to recover. If our experimental results matched, we would expect to recover half the CSE passwords in 15 hours, indicating that the average password had been recovered. Our experimental results show that we had recovered about a third of these passwords.

There are a number of reasons for this, but the most likely is the software. John the Ripper is a sophisticated program, but it does not do a perfect job of guessing passwords in order of increasing randomness. In other words, a user password may not be very random at all, but due to the imperfect nature of the password recovery software, it may not be quickly recovered. Our experience has confirmed this, as we have seen many passwords (like ”myPword!”) take a long time to crack, while possessing little randomness. This raises a future concern. Software algorithms for guessing passwords have room for improvement, and a brief history of password recovery tools indicates that these improvements will likely be made. This is likely to decrease the time to recover a password on every system in every configuration for every password.

4 Mitigating Password Vulnerability

This section considers ways of mitigating the vulnerabilities of current password systems, now and in the future. We consider two broad approaches to limiting the vulnerabilities associated with passwords: the first is to simply prevent off-line attacks from occurring, and the second is to reduce the effectiveness of the offline attack. Note that increases in computing power continually erode the effectiveness of the latter solutions.

4.1 Preventing Offline Attacks

Obviously, offline attacks require seed information to mount the attack: the password file, a TGT, etc. If, however, you could prevent the adversary from obtaining this password material, you could prevent the attack. The mechanism you use to prevent the attack is related to the means by which the password material is obtained. Password material is obtained passively (as in eavesdropping a network exchange), or actively, e.g., by initiating a bogus login or reading a local file. For example, recent UNIX systems have begun to provide mechanisms to reduce the visibility of password material to all users of the system. Recent versions of Unix introduced the concept of the `/etc/shadow` file, which stores the password hash in a file readable only by root, instead of by all users.

In the wrong configuration, Kerberos can represent a surprisingly easy target for an attacker. In V4 and on servers with the preauthentication feature disabled, the password material is sent over networks to the end server. This makes it much easier for an attacker to recover password material. The vulnerability to password-guessing attacks requires a change to the protocol. A number of solutions have been proposed that would eliminate the possibility of these types of offline password-guessing attacks. The principal idea behind these schemes is to stop encrypting data with a key derived from a password. Such a solution effectively defeats these attacks, which requires that the keyspace to search be manageable. If a key is derived from a password, the keyspace is approximately 10^{18} times smaller than the full 112 bit keyspace that 3DES provides. Therefore, the time to search the entire keyspace provided by 3DES is currently prohibitive to recovering passwords in this manner, while the keyspace created by all possible passwords can be comprehensively searched. Additionally, as we have shown, users will choose passwords much worse than the theoretical maximum that perfect passwords can provide. We present a few proposed solutions and discuss their feasibility.

The Secure Remote Password protocol (SRP) is designed to operate securely over untrusted networks [27]. SRP does that by forcing the user to commit to a choice of password early in the protocol. In this way, an attacker can only make one guess per instance of the protocol, making brute-force password attacks nearly impossible. This is especially true if the authentication server employs some type of rate-limiting. However, an attacker can still make online guesses, so if a password is particularly obvious, as it is in some cases, then an attacker may still be able to compromise a limited number of passwords.

SRP is also particularly noteworthy since it doesn't require any changes in the behavior of users. The user doesn't have to manage public keys, carry around an extra device, or do anything differently. The Kerberos tools only have to be modified to change the preauthentication stage of the protocol to negotiate some type of identifying data and use that to exchange the TGT encrypted with that key. Since the key used isn't derived from the user's password, the encrypted TGT cannot be used as material for an offline password-guessing attack. SRP represents the simplest solution that solves these password guessing attacks while adding little complexity to the protocol. The one downside is that it would introduce an additional communication round while negotiating the TGT, an action occurring only once every 24 hours in most Kerberos deployments. Unfortunately, the SRP protocol does require an expensive exponentiation operation, which may make it impractical for extremely low-power devices.

Two-factor authentication is one solution that has gained recent prominence with the strong recommendation of the Federal Financial Institutions Examination Council that two-factor authentication should be used by 2006 for all Internet financial transactions [11]. As long as the method of two-factor authentication used includes some type of random number or symmetric key, this information could be combined with the user's password to create a key that falls randomly within the keyspace of the encryption method used. This then eliminates the fundamental restriction of passwords: that they only occupy a very limited subset of the keyspace supported by the encryption used.

While the above solutions can fix many protocols, in some instances the protocols themselves must be kept simple. Kerberos, for instance, could be difficult to apply as a solution to authentication to financial web services, due to time synchronization restraints, export restrictions, and network latencies. In some cases, a simpler mechanism is required. In this case, one commonly used way to prevent cracking material from falling into the wrong hands may be to encrypt the link over which credentials are transmitted. Currently, this could be done using VPN or SSL. In this way, a site may make a conscious decision to send passwords out over the network, but it will be a network they choose to trust. Since we have shown that, given cracking material, it should be assumed that an attacker can recover the password, it may not be unreasonable to trust the network with the integrity of passwords.

4.2 Hardening Password Systems

Another approach is to strengthen the password systems as they exist today. One solution often proposed is to increase the minimum number of characters required for passwords. As mentioned before, this is restricted by a fundamental human limitation of remembering no more than 7 random items easily. While some users may be able to memorize random strings of 12 or more characters, many will not be able to, and will be forced to write down passwords or pick passwords will very little randomness. A good password cracker will be able to try such non-random combinations quickly, negating most of the benefit to having a longer password. Additionally, even if users pick longer passwords, they can only remember 7 random items[8]. This means that longer passwords may or may not actually contain more entropy. For instance, they may be simply a concatenation of words taken directly from the dictionary.

Implementation restrictions also make this solution practically difficult. Any system that must interoperate with older crypt() implementations is limited to 8 characters. Many sites today require passwords of no longer than 8 characters because of this. Additionally, since users must remember such a large number of passwords, they often re-use them from site to site. As such, they often pick passwords that will be accepted everywhere they try to authenticate, thus restricting their password choices to 6-8 character passwords conforming to standard password policies.

One interesting possibility is to use pass phrases instead of passwords. In this system, a user would choose a pass phrase of around 7 words (the upper limit on what humans can remember). Since there are around 500,000 words in the English language (under some admittedly arbitrary definition) [4], the potential combination of these words can provide a larger password space. However, in practice, informal studies have shown that it may be the case that pass phrases often provide less randomness than passwords [24]. This is conjectured to be the result of users picking common phrases and words, resulting in less total randomness.

From the discussion of the difficulty of cracking MD5 crypt hashes vs Unix crypt() hashes, it seems the solution to these problems might be simply hashing passwords by a few orders of magnitude more times than we do today. However, this presents a few problems. First, this may, depending on the authentication protocol, put a great strain on the server. In the case of Kerberos, if the KDC must hash Ticket Granting Tickets 1000 times (instead of just once, like today), the computational load on the server would increase considerably. Additionally, this exposes authentication servers to DoS attacks, since an attacker can repeatedly attempt to authenticate in most systems, which would lead to a heavy use of resources. Repeated encryptions would also make it difficult to incorporate legacy systems into new authentication schemes. An older 100MHz Pentium system cannot do 100,000,000,000 MD5 hashes very easily. Moreover, low power devices, also, are much less capable of encrypting or hashing values hundreds of thousands of times. Other solutions, such as hardware-dependent encryption algorithms[23], result in a hardware-cryptography arms race. As hardware increases in speed, cryptography is deliberately slowed to maintain its security.

5 Related Work

Morris and Thompson first addressed the issue of password security in 1979 [21] by describing the challenges faced by the UNIX password system. They observed problems that existed within the system stemming from the availability of the password file and then identified guessing passwords as a general approach that was successful in penetrating the system. However, the time to encrypt each guess and compare the result with the file entries was highlighted as the main challenge in password guessing. They analyzed passwords from 1 to 6 characters long from key spaces of 26 to 128 characters and found that exhaustively searching the key space was beneficial in finding a fraction of a system's passwords given enough time. To simplify the searching task, they also noticed that the users of the system chose short and simple passwords, which greatly reduced the key space.

In order to make cracking user passwords more challenging, Morris and Thompson proposed a list of tips to make stronger passwords. The suggestions were attempts to slow the process of password cracking and included basic ideas like choosing longer passwords, choosing passwords constructed from a larger character set, or having the system create passwords. The authors also proposed password salting, combining the password with extra well-known data, as a technique to make pre-computation impossible and increase the time necessary to crack a password. These defenses became the basis for future password cracking prevention techniques.

Ten years later, a paper was published discussing the claims of Morris and Thompson as well as the progress of password security and cracking [15]. Like its predecessor, the paper examined the performance of key space searches. They looked at the possible times for cracking passwords with the same key space as Morris and Thompson, but examined lengths ranging from 4 to 8 characters. With the addition of password salting, the searches had indeed become more complex. The authors claimed that a large key space of 95 characters "is large enough to resist brute force attacks in software ... It is impossible to use exhaustive search over the large search space..." However, they determined that password cracking was very possible if the search space was limited. This could be done by creating a common password word list to guess passwords from instead of attempting to guess every possible password.

In order to maximize the difficulty of password cracking, [15] discussed execution speed of the hashing mechanism and password entropy. The authors concluded that, because computing speed and power were changing, attacking the problem by increasing the speed of the encryption algorithm was not possible. They also analyzed other solutions like changing the encryption algorithm and better protecting the cracking material. It was concluded that making passwords less predictable was the principal defense against password cracking.

Dictionary attacks are the fastest, easiest way to crack passwords because passwords are commonly chosen from a small set of words. In order to prevent these fast, simple attacks, systems implemented policies that required passwords contain a certain amount of entropy. The policies include rules on minimal length and required password characters. To enforce these policies, password checking software was developed, which determined if a given password had enough entropy to be considered secure. However, dictionary attacks then evolved by exploiting common non-dictionary choices for passwords. The techniques used by these attacks included searching for random capitalization, permutations of dictionary words and usernames, letter and number manipulations, and foreign language words. These attacks continue to evolve by examining and exploiting common policies. Unfortunately, research has shown that despite password policy advice, users still tend towards dictionary words for passwords [18].

Sophisticated analysis of the English language has aided in password guessing. For example, character frequency, once very successfully used as a spellchecker in UNIX, is now being used in password cracking[3]. Analysis of common passwords has also contributed to faster password cracking. Possible passwords are tried in a certain order based on how common the password is. From these advanced methods, we see that password guessing techniques continue to evolve as long as passwords are still in use. As a result, a variety of solutions have been proposed to combat password guessing.

However, twenty five years after Morris and Thompson's paper, modern passwords are still vulnerable to offline cracking attacks. Basic hashes and digests are still used to encrypt these passwords, thus today's cracking material is similar to that available in the 1980s. However, the ability hash passwords, and thus recover passwords, has drastically improved due to developments in software that have quickened the performance of these encryption techniques, sometimes by as much as a factor of 5[9]. These improvements have impacted the speed at which passwords can be cracked, thus increasing the difficulty of preventing offline password cracking.

6 Conclusion

The fundamentally limited amount of protection current passwords can provide is no longer sufficient to protect password-based authentication systems vulnerable to offline attacks from brute force attacks by the rapidly growing computing resources available. Because all passwords will be recoverable, the security of any system based on passwords will depend on the availability of cracking material, not how random passwords are. As such, protocols must be designed to not allow any type of offline attack, and the material that can be used to mount such an attack must be protected with the understanding that its confidentiality is equivalent to the security of the authentication mechanism as a whole.

References

[1] AMD 3-year technology outlook. http://www.amdcompare.com/techoutlook/.

[2] EFF DES cracker project. www.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/.

[3] John the ripper password cracker. http://www.openwall.com/john/.

[4] Number of words in the english language. http://hypertextbook.com/facts/2001/JohnnyLing.shtml.

[5] pwgen cvs changelog for revision 1.3. http://cvs.sourceforge.net/viewcvs.py/pwgen/src/pw_rand.c?rev=1.9&view=log.

[6] Sans institute. http://www.sans.org/aboutsans.php.

[7] SANS password policies. http://www.sans.org/resources/policies/.

[8] The magical number seven, plus or minus two: Some limits on our capacity for processing information, 1956.

[9] Eli Biham. A fast new DES implementation in software. *Lecture Notes in Computer Science*, 1267:260–??, 1997.

[10] Shekhar Y Borkar, Pardeep Dubey, Kevin C Kahn, David J Kuck, Hans Mulder, Stephen S Pawlowski, Justing R Rattner, R M Ramanathan, and Vince Thomas. Platform 2015: Intel processor and platform evolution for the next decade. Technical report, Intel, 2005.

[11] Federal Fiancial Institutions Examination Council. Authentication in an internet banking environment. http://federalreserve.gov/boarddocs/srletters/2005/SR0519a1.pdf.

[12] Larry Cuban. *Oversold and Underused: Computers in the Classroom*. Harvard University Press, 2001.

[13] Manek Dubash. Moore's law is dead, says Gordon Moore. *http://www.techworld.com/opsys/news/index.cfm?NewsID=3477*, 2005.

[14] Magnus Ekman, Fredrik Warg, and Jim Nilsson. An in-depth look at computer performance growth. *SIGARCH Comput. Archit. News*, 33(1):144–147, 2005.

[15] David C. Feldmeier and Philip R. Karn. Unix password security - ten years later. In *CRYPTO '89: Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology*, pages 44–63, London, UK, 1990. Springer-Verlag.

[16] Radhakrishna Hiremane. From moore's law to intel innovation - prediction to reality. *Technology@Intel Magazine*, April 2005.

[17] Ian Jermyn, Alain Mayer, Fabian Monrose, Michael Reiter, and Aviel Rubin. The Design and Analysis of Graphic Passwords. In *Proceedings of the 8th Annual USENIX Security Symposium*, 1999.

[18] Daniel V. Klein. "foiling the cracker" – A survey of, and improvements to, password security. In *Proceedings of the second USENIX Workshop on Security*, pages 5–14, Summer 1990.

[19] Rob Lemos. Passwords: The Weakest Link. http://news.com.com/2009-1001-916719.html, 2002.

[20] Fabian Monrose and Aviel Rubin. Authentication via Keystroke Dynamics. In *Proceedings of the 4th ACM Conference on Computer and Communication Security*, 1997.

[21] Robert Morris and Ken Thompson. Password security: a case history. *Commun. ACM*, 22(11):594–597, 1979.

[22] Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*, pages 364–372, New York, NY, USA, 2005. ACM Press.

[23] Niels Provos and David Mazières. A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track*, pages 81–91, 1999.

[24] Arnold G. Reinhold. Results of a survey on pgp pass phrase usage. http://www.ecst.csuchico.edu/ atman/Crypto/misc/pgp-passphrase-survey.html.

[25] Wayne C. Summers and Edward Bosworth. Password policy: the good, the bad, and the ugly. In *WISICT '04: Proceedings of the winter international synposium on Information and communication technologies*, pages 1–6. Trinity College Dublin, 2004.

[26] W. Timothy Polk William E. Burr, Donna F. Dodson. Electronic authentication guidelines. NIST Special Publication 800-63.

[27] Thomas Wu. The secure remote password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, 1998.

[28] Thomas Wu. A real-world analysis of Kerberos password security. In *Internet Society Network and Distributed System Security Symposium*, 1999.

Appendix

Unix Crypt

The previous work in this field has examined password cracking primarily as it applies to cracking attempts against the Unix/Linux /etc/passwd, so we start by examining this password storage type to give a sense of how modern techniques and equipment compare to what was previously available. This type of authentication system is used to authenticate users to a Unix/Linux system. In the traditional Unix crypt system, hashes of users' passwords are stored in a password file often named either /etc/password or /etc/shadow, with the 2 letter salt prepended to the hash. A user enters their password, which is then combined with the salt in the password file, and then encrypted using a variation of the DES algorithm. The resulting ciphertext is compared with the hash in the password file, and if the values match, the user is successfully authenticated. Newer systems, such as the one first found in modern crypt function, hash the password with MD5 repeatedly (up to 1000+ times), instead of just once[23].

For this type of authentication system, an attacker must somehow obtain a copy of this password hash file. Unfortunately, this can be made available to an attacker in a variety of ways. The simplest of these is if an attacker has root access on a machine, in which case he can simply copy the /etc/shadow file. If the password hashes have not been moved to /etc/shadow, they will reside in the world-readable /etc/passwd file, in which case an attacker with normal user access to the system can simply copy the file. However, amongst other attacks, there is one attack in particular which allows a large number of attackers access to the password file. The Network Information Services, or NIS, is often used to centralize authentication decisions over a large number of machines. NIS provides a utility, ypcat, which allows users to view portions of information about system users. We found ypcat to be often misconfigured in a way that allows any user on any system connected to NIS to simply ypcat the password hash portion of each user in the system. In this way, an attacker can gain access to the credentials of each user on any system tied to NIS.

The actual process of guessing a user's password is very simple. To recover passwords from this password file, an attacker takes candidate passwords, combines them with the appropriate salt, which is well known, and applies the appropriate hashing technique to this value. The attack then checks to see if the result from hashing his guess matches the hash value in the password file.

Kerberos

We also evaluate password cracking as it relates to modern versions of Kerberos. Kerberos, a popular single-sign-on protocol, is widely touted today as a solution to ”the password problem.” It is used to authenticate to a variety of services, including IPsec, Email, Web Services, Directory Services, and many more. Because Kerberos is often used as a single-sign-on service, a compromise of Kerberos credentials is often equivalent to a compromise of the users' credentials to every service in the network. Unfortunately, Kerberos, in every version, is vulnerable to a variety of password-guessing attacks[28].

One of the biggest issues with Kerberos as it relates to password cracking is that as opposed to most Unix/Linux systems, where an attacker must have a valid user account (or have compromised one), all the cracking material necessary to mount an offline attack against Kerberos credentials can be obtained either by anyone who asks or anyone who can sniff Kerberos traffic, depending on the restrictions in place. During a client's initial authentication In the Kerberos protocol, a client sends an authentication request to a server in charge of authentication for the Kerberos realm called the KDC. If the client makes a correct request, the KDC will return a token called a ticket granting ticket (TGT). This token can be used to obtain credentials to any Kerberized service the client can access. Unfortunately, when this TGT is given to the client, it is transmitted over the network, encrypted with a key derived from the user's password. While the user's password itself is never sent in any form, this TGT is still vulnerable to password guessing attacks, as described below.

An attacker has a variety of options for obtaining cracking material (the TGT) required for this attack. In Kerberos v4, a KDC will return a TGT to anyone who asks for it. Thus, in this case, an attacker's job is completely trivial, and he can easily obtain a TGT to crack for each user in the system by simply asking. However, Kerberos v5 introduced the idea of preauthentication. With preauthentication, a user must use the key derived from his password (as described above) to encrypt a timestamp, which is included in the client's request for TGT. The server will only return a TGT if the timestamp received by the server decrypts correctly with the client's key. In this way, the server attempts to insure that a TGT is only sent to the user to whom it belongs.

However, an attacker attempting to crack a Kerberos 5 deployment still has a number of options for recovering a TGT. First, many Kerberos deployments do not have preauthentication required for all users.

In this situation, an attacker may simply ask for TGTs as he did for Kerberos v4. Many deployments, in order to ensure backwards compatibility, still support Kerberos v4, so an attacker may simply ask for v4 tickets for each user. Finally, in any of these systems, the TGT itself is sent over the network in the clear, so an attacker that can sniff the network can trivially recover the TGT.

In order to compromise Kerberos credentials, an attacker first captures the TGT using one of the aforementioned methods. Then, an attacker generates a password guess. This guess is transformed into a key using the Kerberos ”stringToKey” function, which uses both the password guess and information found in the TGT itself, such as the user’s name and the name of the Kerberos realm. Then, this key is used to decrypt the captured TGT. Since each TGT, if decrypted correctly, contains the string ”krbtgt”, it is easy for an attacker to know if the decryption, and therefore the candidate password, was correct.