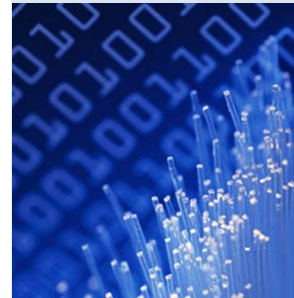




## Everyday Password Cracking

An IRM Research White Paper by  
Thorsten Fischer



## **IRM Research**

Information technology constantly changes and advances. IRM is dedicated to keeping pace with new technology and continuing to innovate in the field of information security. This ensures that we are well informed of new issues and technologies, expanding our knowledge and providing world class services to our clients.

## Everyday Password Cracking

We present a view on what password cracking currently looks like in the context of the everyday work of a security consultant, discuss a number of approaches and provide advice to make password cracking more successful in practice. We also make some suggestions regarding potential improvements to password cracking techniques.

### **Introduction**

Password cracking is anecdotally described as a straightforward process. While the general idea is indeed easy to grasp, one of the biggest challenges faced by security consultants is time constraints, as their authority to carry out a piece of work is usually very narrowly defined. It is therefore imperative that password cracking is performed in a way that offers the highest chance of success within the given amount of time. This paper is structured as follows: Section 2 (Methods and Approaches) first makes the case for cracking password and then discusses methods and the most common approaches to password cracking, including their applicability to certain situations. Section 3 (Concerning Password Policies) discusses a few issues with password policies and how they might make some theoretical cracking approaches easier, rather than making them more difficult. Section 4 (Conclusion) then presents the conclusion that the preferable approach to cracking a password depends on the occasion and the desired goal. Also, suggestions for improvements in techniques and tools will be made in this section.

### **Methods and Approaches**

A number of methods for password cracking exist. The following sub-sections discuss these approaches and their applicability. Note that this paper is only concerned with the actual cracking of passwords, not with defending against such attack attempts. Other authors have dealt with the creation of dependable systems in a much more exhaustive way [6] than is possible here.

### **The Case for Password Cracking**

First, why discuss password cracking at all? For the author, the main reason is because it plays an important role in his everyday work as a technically focused security consultant and penetration tester. The classic literature approaches password cracking either for demonstrating the low quality of most passwords chosen by users [10] or as a study in order to suggest improvements to password quality assurance, for example by employing quality checks at the time passwords are chosen [8].

Username/password combinations are the most commonly encountered means of authentication for computer systems, networks and applications today. While a username identifies an account, the password is intended to authenticate the user to the system. Usernames are rarely secret and should be considered public knowledge for the purpose of assessing the security of a system, whereas passwords are meant to be pieces of sensitive information, because knowledge of the password associated with a victim's username will effectively allow an attacker to impersonate that victim.

We are therefore approaching password cracking in one of two different ways. First, it can be part of a penetration test, where a consultant gains access to a system where encrypted credentials are stored. Given sufficient privileges, these can be copied and cracked offline, potentially providing further access to other systems or to user accounts with additional privileges. Or, the evaluation could specifically focus on password quality. In this case, the client would provide the consultancy with a copy of a database with encrypted passwords and maybe define what resources are perceived to be available to an adversary. The consultancy then employs these resources for a defined time span and determines the password's susceptibility to attack. In either case, different approaches to cracking passwords can be employed, depending on the goal of the cracking attack. The technique on the other hand is always the same, regardless of the approach: a suspected cleartext password is chosen. The same encryption algorithm used

by the password mechanism is applied to the password and the result is compared with the encrypted password. If there is a match, the correct original password has been found. If there is no match, the next password is chosen. This process is repeated until wither the original cleartext password has been discovered or until the allocated time runs out.

## Breaking the Encryption Scheme

It is paramount that passwords are stored in an encrypted form. Other kinds of scheme encrypt a fixed string using the username as the encryption key. When a user wants to log into a computer, the same mechanism is applied to the password supplied. If they match, the correct password was supplied and the user is granted access.

An attack on an encryption scheme is usually of interest to cryptographers if it is faster than an exhaustive search for the key. Attacking cryptographic schemes, as exciting as it may be in theory, is not part of a penetration tester's time constrained assignments. This approach is therefore not discussed in this paper.

## Password Guessing and Default Passwords

Many software packages come pre-installed with default credentials, which are usually well documented in vendor manuals, installation tutorials and discussion forums. Two well known examples in database software include the default blank passwords for SQL Servers' sa account and MYSQL's root account. Collections of default passwords are available on the Web [4]. Much could be said in favour of "trusting one's intuition" when guessing passwords. Especially when the number of attempts is limited, then using the username itself, the string "password" and maybe one other string that appears obvious in the given situation can yield surprising results. However, the main focus of this paper is to discuss more structured password cracking approaches.

## Brute Force

Maybe more correctly described as exhaustive search, this is arguably the best known and somewhat obvious approach to password cracking. Starting at any one password in the whole password space, every possible password is attempted in order.

With its rather grand sounding name and the fact that many people have heard of the term "brute force" makes a good phrase for penetration testing reports. The technique is easy to implement and reasonably fast, since not pre-computation is involved in determining the passwords to test. Also, since the search is usually linear, it is easy to interrupt the cracking process and pick it up at a later point. This is useful at a client site, where an initial cracking attempt can be carried out using locally available resources; if this is not successful, the interrupted session can be picked up with more powerful resources offsite. However, there are a number of issues that might render this approach rather impractical. First, in many cases there is the sheer size of the password space to take into account. The following simple sum describes the number  $p_i$ , of different possible passwords (size of the password space) for a character space of  $n$  different characters of maximum length  $i$ :

$$p_i = \sum_{k=1}^i n^k$$

Figure 1

For a character set which contains only the 26 alphabetic characters from "a" to "z" and which has a maximum length of 2 characters, there will be  $26^1 + 26^2 = 702$  different passwords to consider. For LM hashes, which are still widely used in Microsoft Windows networks (a character set of 69 characters and a maximum length of 7 characters) there are 7555858447479 (roughly  $7.5 \times 10^{12}$ ) different possible

passwords. Note that for a length of 7 alone, 7446353252589 different passwords are possible. The actual amount of time spent is not only determined by the password space, but also by the algorithm used. A number of different password algorithm implementations have been evaluated in [7]. While the complexity of searching through the password space is of course always the same. The actual algorithm implementation can dramatically change the time required in a real world cracking scenario by several orders of magnitude. In the case of Jon the Ripper, for example, one benchmark for cracking LM hashes showed more than 4.9 million attempted passwords per second, whereas on the same machine a password encrypted using the MD5-based scheme used in many modern Unix implementations could be evaluated at little more than 4000 attempts per second. Probably the most considerable disadvantage of the brute force approach is that it does not take password policies or user preferences into account. If an attacker first attempts all combinations of lowercase letters even though there are technical controls in place that demand at least one of the characters to be an uppercase letter, a lot of the attacker's time will go to waste. The same consideration applies in case of character sets that may or not be applicable in a particular language environment.<sup>1</sup>

## Word Lists

Word list attacks are based on the assumption that users tend to use passwords which can be found in dictionaries, or which can be guessed and categorised by other means. This categorization would ideally be tailored to the environment at hand and in many cases include the usernames themselves, common words in the language used at the target site, company specific, and other terms. A detailed look at how a word list attack can be constructed can be found in [8].

This approach tends to work well in practice; mostly because users share a tendency to ignore all the advice on choosing passwords that they are given. This attack is fast to perform, as the passwords to be attempted can be created in advance and also quite easy to implement.

Form a security consultant's perspective, this approach is of particular value because a client can easily reproduce the password cracking steps, because traversing even large dictionaries tends to be relatively fast, which makes it easier for clients to understand the attack. This in turn increases the chance that the problem of weak passwords that are cracked via word list attacks is actually addressed by the client.

Compression Algorithm	Compressed Size	Compression Ratio
gzip	26095821	63.39%
bzip2	8274521	88.39%
lzma	1779646	97.50%

Table 1: Compression ratio of a highly structured, sorted word list using different tools

A number of word lists can be created well in advance and also be used in the rules-based approach described later in the paper. The size of such lists is obviously dependent on the character set used. The larger the set, the longer the lists; compressed storage may be required. For practical use, the attacker should also be aware that some tools, for example John the Ripper [3], are currently not capable of handling input file larger than 2 Gigabytes.

For practical purposes, word lists come in two flavours. The first is a list of words in the natural language sense, for example all English words from a well known dictionary. Numerous such word lists can be found

---

<sup>1</sup> Note however that password encryption is usually carried out on a certain number of bytes represented in a computer's memory. What characters these bytes eventually represent after having been mapped to a "codepage" should be of no concern to the encryption algorithm. This could lead to seemingly different cleartext passwords, yielding the same encrypted password string.

on the Internet. However, after obtaining word lists some care needs to be taken to make them actually usable. Many lists include comments, references, HTML tags and other data that has no relevance for the actual password cracking process. They may also contain a large number of characters that are not even in use in the target character set. Blindly concatenating these files before using them can lead to impressive claims on the size of a list, but the only achievable result will be a lot of wasted computation time. This is especially true for duplicates in the list.

The other flavour of word list is one that contains structured data, for example all possible passwords for a given password length within a given character set. The advantage of such lists is that they can guarantee that a certain subset of character space has been covered. When compressed, they also require very little disk space. Consider the data in table 1 which describes compression ratios of all five-character alphabetic passwords. The original file size was 71288256 bytes.

As can be seen, substantial savings in disk space can be realised with decent compression tools, making the storage of potentially large amounts of pre-generated and highly structured data feasible. In a similar vein, consider table 2 which describes the disk space requirements for storing a complete list of all possible passwords for a number of different character sets. The table shows, for example, that all possible 7-character LM passwords from a 69-character character set (the size typically assumed) can be stored in about 56 Terabytes.

Length	Num(10)	Alpha(26)	Symbol(33)	Alnum(56)	Alsym(59)	Alnumsym(69)
1	n/a	n/a	n/a	n/a	n/a	n/a
2	n/a	n/a	n/a	n/a	n/a	n/a
3	n/a	n/a	n/a	n/a	n/a	1
4	n/a	2	5	8	57	108
5	n/a	67	232	343	4096	8949
6	6	2062	8621	14531	281585	720431
7	76	61277	325152	597871	18986904	56811166
8	858	1792377	12071301	24213780	1260255754	4409966821

Table 2: Storage requirements in Megabytes for all possible words of lengths 1 to 8 in a number of different character sets. "n/a" denotes less than a Megabyte.

Making the bold assumption that the same compression ratio as described above can be achieved on other highly structured and sorted files, a cracking approach could be derived that stored only the passwords themselves and sorts them into files according to, for example, the first characters of the corresponding LM hash. These files would then be compressed. A lookup in this table would then involve opening the file that corresponds to the first characters of the given hash. The plaintext passwords in this file would then need to be hashed one after another until the correct hash is encountered and the password is therefore identified.

While this approach is obviously naïve in its assumptions, it would guarantee a 100% success rate on lookups. Lookups should also be faster than a search in rainbow tables, for example. More research would be required to explore the possibilities and limitations of this approach.

## Rules-based Approaches

The “rules-based word list” approach is the approach that appears to be the one most commonly discussed in literature on password cracking (which overall appears to be sparse). Two of the most commonly cited examples, [8] and [10] both elaborate on how to construct rules based on usernames, commonly encountered passwords, dictionaries and permutations of words. The password cracking tool John the Ripper (JtR) [3] is entirely based on permutations of password candidates according to a set of configurable rules and character sets.

The order of these rules is important and in the case of JtR, greatly benefits from the experience of the program’s author, Solar Designer. However, one has to consider that the author is probably working in a particular cultural environment, where one particular language is predominant. While English is certainly the most common language in IT, there is nothing that guarantees that normal users will choose English passwords. It is therefore important that password cracking attempts are aligned with the most common language of the environment. In the author’s experience, cracking passwords using language-specific dictionaries yield a significantly higher success rate than mere reliance on English word lists. This may also be of interest when determining the quality of passwords in an assessment of an offshore division, for example.

## Lookup Tables

A lookup table is effectively a pre-calculated attack. The hash for every password to be attempted is generated in advance and stored in a way that allows for quick lookups. This has the advantage that the actual hashing or encryption steps do not have to be carried out more than once, and simple table lookups are generally very fast – at least much faster than encrypting or hashing.

Constructing a lookup table is easy to do, but requires a huge amount of storage space. Also, they do not take password policies into account. This means that a lookup table that is derived from simple enumeration of the password space is unlikely to meet the cracking requirements.

There are a number of lookup tables currently accessible on the Web, and some of them compromise projects that have carried out research into the effectiveness of lookup database creation. However, at the time of writing only very basic lookup tables were reachable on the web, which means that a satisfying conclusion regarding these tables could not be included in this paper.

A targeted lookup table is perceived to be of more use. This approach is discussed in the section below.

## Rainbow Tables

A rainbow table, following an approach developed in [9], is a pre-calculated lookup table with more clever storage and less convenient lookup times. Most rainbow tables obtainable are advertised using a certain success probability value. Success rates in the range of 99% or higher can be found frequently and are well founded in the maths described in the paper cited above. However, in practice the success rates appear to be much smaller than that. The author took a sample of the first 100 LM password hashes from his own collection and attempted to crack them using the rainbow tables available from [5], which amount to a compressed size of more than 30 Gigabytes. Of the 100 passwords, not a single one was cracked using

these tables using the rcrack tool on Linux. It would appear that the success rates are much lower in practice than the theory surrounding the avoidance of chain collision and merging would suggest.<sup>2</sup>

Because of the number of additional calculations carried out, rainbow tables require much longer search times than straight lookup tables. They are also more likely to completely ignore password policies because of their restriction to a particular character set, which may make them unsuitable in scenarios where policies are enforced. Chain generation functions which adhere to a particular policy may make it easier to generate more “targeted” rainbow tables, though.

Rainbow tables also require a long time for calculations in advance, as they are essentially pre-calculated lookup tables. If time constraints make it unfeasible to generate a straight lookup table for a particular hash and character set combination, they will also make it unfeasible to create a rainbow table meeting the same requirements.

As an additional subjective note, it appears that the performance of rcrack deteriorates unnecessarily when the number of hashes to crack increases, which in the past has made it unsuitable for attacks against large lists of hashed passwords.

## Previously Cracked Passwords

A more refined approach to lookup tables is the practice of storing previously cracked passwords. The degree of success will depend on the correctness of the following assumptions:

- The same user will tend to use the same password in different places;
- The same user will be unlikely to change their password between two different tests for the same client, or at least unlikely to change it to something substantially different;
- Two different users will not differ too much in their choice of passwords.

The first assumption can be taken as granted, as penetration testing experience shows. Also, the literature often highlights this particular weakness in users [6].

The confirmation for assumption number two again comes from practical experience. Tools like pwdump6, which is part of the fgdump suite of tools [2] often shows that users tend to change their passwords e.g. by only incrementing an appended number. Another approach is to include the current month’s name in the password [6].

When it comes to the differences between passwords of different users, it is too early to make a useful prediction. The author currently holds a repository of 75000 previously cracked LM hashes, about 80000 NTLM hashes and several thousand encrypted representations of passwords from other systems and software installations. Using these lists, cracking newly encountered hashes using a simple lookup is still very rare, though such attacks are occasionally successful. The author believes that a large sample of encrypted “in-the-wild” passwords will be required to make meaningful predictions about the success rate of attacks using this approach.

Note that a list of previously cracked passwords can also be of great value when conducting a test of an infrastructure that has been evaluated before. If even one single previously cracked password has been left unchanged, cracking it will be a matter of a simple table lookup.<sup>3</sup>

---

<sup>2</sup> Note that the author currently does not have any recorded data to support this claim; especially the original data that the above claim is based on is unfortunately not obtainable anymore. The experiment would obviously need to be repeated a number of times to show whether the results substantiate the author’s claim or whether it would need to be retracted.

Anyone who has ever set out to search the Internet for freely available lists of previously cracked passwords will have arrived at the conclusion that this kind of resource is somewhat scarce. The only useful results when searching for "john.pot", for example, appear to come from publicly accessible user home directories that have created a JtR test installation. Therefore, a penetration tester who is going to spend time with more password cracking would be well advised to create their own collection of previously cracked passwords.

## The Approach to Chose

The approaches described above are useful in different scenarios. We would like to proceed by describing two different goals and show which password cracking approaches may be most suitable to reach each goal.

## Overall Password Quality Evaluation

In this case, the idea is to determine the susceptibility of a set of encrypted passwords. For such an evaluation, the recommended approach would be as follows:

- Use the list of previously cracked passwords to determine if the password has been cracked before. This attack may lead to success independent of password quality.
- Perform a word list attack using increasing word lengths. Depending on policy, small word lists may not be worthwhile to try. Ensure that context-specific terms are part of the dictionaries used.
- Perform a word list attack, again using increasing word lengths, but this time with rules for word mutations. Note that a list of previously cracked passwords with mutation rules applied might also be worthwhile.
- Use a linear brute-force attack or an incremental approach like JtR's to attack the remaining hashes.

Users of JtR will note that this is similar to the default mode of operation for this tool. However, if the attacker is interested in keeping separate statistics on each of these steps, it is advised to carry them out in distinct sessions.

JtR offers functionality to make such an approach manageable. A wrapper script can then be used to carry out the required steps in succession. Following this well-defined approach, and assigning a pre-determined amount of resources and time for the job, will also result in a quantifiable evaluation for a client.

## Dedicated Cracking of One Particular Password

The approach for cracking one particular password is slightly different. A prime example for such a password is the password for a Windows domain administrator account; such elevated privileges usually warrant the dedication of all available resources to the cracking process. The author suggests the following steps:

- Use the list of previously cracked passwords to determine if the password has been cracked before. This attack may lead to success independent of password quality.
- Perform an attack using rainbow tables if success can be expected. This is likely to be the case for LM hashes, for example.

---

<sup>3</sup> This only applies to unsalted passwords of course. However, even if a password has been "changed" to be the same again, leading to the application of a different salt and hence a different encrypted representation, a very fast attack can be constructed by using the list of previously cracked passwords as input to a word list attack.

- Perform a word list attack using increasing word lengths. Depending on policy, small word lists may not be worthwhile to try. Ensure that context-specific terms are part of the dictionaries used.
- Perform a word list attack, again using increasing word lengths, but this time with rules for word mutations. Note that a list of previously cracked passwords with mutation rules applied might also be worthwhile.
- Use a linear brute-force attack or an incremental approach like JtR's.

Note the targeted use of rainbow tables which was not part of the approach in the previous section, because of the performance shortcomings outlined above.

The main lesson to be learnt from this is that different goals in password cracking warrant at least a different order of steps, offering a higher chance of success depending on the scenario.

## Choice of Hardware

Often, a decision is made to obtain dedicated hardware for password cracking. This makes sense, as the requirements for CPU power to crack an important password may not be available otherwise, as most computers in a consultancy company will be tied into everyone's day-to-day work.<sup>4</sup> In addition, the time constraints put on most security consultancy assessments mean that additional CPU power is the only way to increase the likelihood of success. Table 3 summarises a number of benchmarks taken using the JtR program on various PC platforms running Linux. In more detail, the platform benchmarked were:

- Intel Pentium III, 733MHz;
- Intel Pentium M 1.5GHZ;
- Intel Core Du T2300 1.66GHz;
- AMD64 Dual Core 4200+;
- Intel Xeon 2.8GHz.

Algorithm	Pentium III	Pentium M	Core Duo	Dual Core	Xeon
Traditional DES	239936	574681	638620	1036K	579715
NT LM DES	1793K	4463K	4954K	7385K	5667K
NT MD4	543352	1285K	1335K	2658K	1487K
MS Cache Hash	366108	910863	915204	1658K	1134K
BSDI DES	8022	18867	20908	34874	19774
FreeBSD MD5	1821	3414	4190	8703	7796
Apache MD5	1808	3814	4185	8704	7869
Post.Office MD5	535234	1324K	1518K	1699K	1680K
Raw MD5	769660	1950K	2080K	2932K	2152K
IPB2 MD5	462678	1010K	1191K	1551K	1458K
Raw SHA1	523156	1435K	1317K	2179K	1780K
Kerberos v5 TGT	8112	20915	19430	29360	15306
Netscape LDAP SHA	502845	1309K	1242K	2123K	1667K
OpenBSD Blowfish	123	254	283	365	399
Eggdrop	2399	7214	4228	20940	13487
Kerberos AFS DES	210278	202700	497590	889702	433672
MySQL	397953	1125K	1234K	2382K	784862
Lotus 5	60960	128497	46260	114950	188524
More secure Internet	36476	80809	85985	57073	125202

<sup>4</sup> Though a company-wide way to make use of idle CPU cycles on internal computers could certainly be devised

Password					
----------	--	--	--	--	--

Table 3: John the Ripper benchmarks on different hardware architectures, running version 1.7.2 on Gentoo Linux

The benchmarks were taken using JtR's `-test` option. A more realistic benchmark could probably be obtained by carrying out an actual cracking process using a number of input hashes that are known to require a specific computation time on a reference platform, then comparing other results with that time. However, for the sake of speed comparison, the results in this table should be deemed sufficient.

The most surprising (or disappointing) result was that the benchmark on the AMD64 processor outperformed the Intel Xeon processor, even though the latter was 15% more expensive at a popular online store at the time of writing. This shows that tests should be carried out before investing money in a dedicated password cracking infrastructure. When choosing hardware, it should also be considered that the most commonly used tools like JtR do not support more than one processor. Note that the MPI patch for this tool was not considered here, because it tended to collide with the patches already applied to the local JtR installation.

The same is true for multiple machines. While a number of solutions exist for distributing password cracking jobs, the most promising approach for cracking any one given set of encrypted hashes appears to be a clustering approach that aggregates any number of machines into one single, high-speed processor. An implementation of such a clustering approach may result in a less expensive setup that offers the same cracking performance as a single expensive processor.

The lack of parallelisation capability warrants another note. A commonly chosen approach when carrying out a wordlist crack on a large set of password hashes is to split up the list of hashes into multiple chunks, then cracking the smaller lists using the same dictionary on different computers to which the chunks were distributed by hand. However, this approach ignores the way the bulk calculations are carried out during password cracking.

To recall: With an unsalted password scheme, the password candidate is first hashed, and the hash is then compared with the hashes to crack. However, hashing is an operation that is far more computationally expensive than the comparison involved. Therefore, when splitting up the list of hashes but retaining word lists, the computationally extensive of hashing the whole word list is doubled. Instead of this approach, with an unsalted password scheme it is therefore faster to split up the word list and distribute the pieces, then carry out the cracking attempts on the whole list of password hashes on each computer.

## Concerning Password Policies

One of the recommendations most commonly written down by security consultants is something along the following lines:

*"[Consultancy] recommends that [client] enforce a stronger password policy. Passwords should have a minimum length (possibly 8 characters or more) and include uppercase and lowercase characters, numbers, and special characters"*

And a well-intentioned recommendation it is. However, in practice such recommendations still largely appear not to be followed. And if misunderstood, password policies can do more harm than good. Complexity requirements are based on the assumption that an attacker will carry out a linear brute-force attack on passwords, walking through the password space in a particular order. These requirements are thus perceived as an increase in the number of characters  $n$ , resulting in a much larger sum from the equation (Figure 1).

However, this is not the complete picture. Common thinking makes the following assumptions:

1. That a brute-force attack is carried out in an incremental fashion;

2. That a brute-force attack will, on average, be successful after  $P_i/2$  steps, and therefore
3. That an increase in  $n$  will result in a larger  $p_i$  according to the equation (Figure 1), resulting in an (ideally unfeasibly) long time required for cracking a given password.

Let us assume that a user has the opportunity to enter a new password. He is free to choose whatever password he wants from uppercase and lowercase letters as well as from numbers; 62 possible characters altogether. And so he would, but there is one catch: at least one of the characters has to be an uppercase character. From the complexity requirement point of view, this may appear sensible. However, the possible password space is now effectively **reduced**, because instead of having the ability to choose from 62 characters for all positions, one position is now reduced to 26 characters, resulting in a total password space that is smaller than the original one.

Therefore the requirement to choose  $j$  of all characters from a character space of size  $m$  results in the following observation:

$$p_i = \sum_{k=1}^i n^{k-j} m^j < \sum_{k=1}^i n^k$$

Figure 2

Where  $j < i$  and  $m < n$ , with  $i$  and  $n$  as described in the first equation (Figure 1)

The sum changes and the result becomes smaller accordingly for requirements like "at least one uppercase letter and one number", for example. This policy will make it a waste of time for the attacker to search through all passwords that only consist of lowercase characters. And a simple linear search will do exactly that. Password cracking tools like Cain [1] will walk through the characters space from any given starting point defined by a "starting word".

It would appear that we have encountered a requirement for a formal description of password policies and for a definition of an order on the resulting cleartext password space, which is required for a structured generation of password candidates that adhere to a policy. There may be some difficulties with that; consider the password Aaaa as the "first" password that adheres to the example policy mentioned above that required at least one uppercase character. It is easy to iterate to the 26<sup>th</sup> password of Zaaa. However, what would be the next password to attempt? Some examples are AAaa, aAaa, Aaab, and there are any more possible. The first of these examples should certainly be tried after the others, since it can be commonly observed that users will choose their password to meet the policy as closely as possible and use the definition as a memory aid. Additional requirements will complicate the generation further.

Some policies are outright harmful. For example, the author once encountered a policy for an application that used six-digit PINs for authentication. The two requirements for the PINs were as follows:

- No two digits in a row must be the same number. For example "11" was not allowed as a substring of the PIN.
- No two digits were allowed to be consecutive numbers. For example "23" was not allowed as a substring of the PIN.

This limited the PIN space dramatically. Where a simple PIN of six digits may be one of  $10^6 = 1000000$  numbers, the restricted version came to only  $10^1 \times 8^5 = 327680$  different possible PINs<sup>5</sup>. This is a password policy that should make the average brute force attack more than sixty percent faster.

There is yet another catch – the above numbers implicitly assume an attacker carrying out a random search over the password space. However, it is the author's experience that users who are faced with any kind of password policy tend to meet the policy in an as narrow manner as possible.<sup>6</sup> For the above PIN example, the easiest number to remember appeared to be 135246. As it turned out, this was the number most widely used amongst users. This means that password policies need to be carefully designed to ensure they do not have an effect contrary to what was intended.

The whole area of password policies and potential ways of taking advantage of them requires more research. If a password search would be carried out in a more structured fashion, brute force attacks on well chosen passwords could be on average a lot more successful in the future than they are today, because the typical linear brute force approach wastes a lot of time in subsets of the password space where passwords are by policy guaranteed not to occur at all.

## Current Ways of Dealing with Policies

Currently, password cracking programs lack functionality to properly take password policies into account. Consider for example the following "external filter" for JtR. These filters are snippets of C code (yes, JtR comes with its own stripped-down C interpreter) that approve or reject a password candidate.

The filter below ensures that a password candidate matches the Windows 2003 Server default password policy. This policy requires that there is at least one character each in the password from at least three of four different character sets: lowercase letters, uppercase letters, special characters, and digits.<sup>7</sup>

---

<sup>5</sup> Slightly more in fact, as the number 9 has no disallowed subsequent number. This does not however change the final number considerably.

<sup>6</sup> We assume that this is done because that way, the policy itself can be used as a memory aid for remembering the password itself.

<sup>7</sup> Note that the minimum password length of eight characters is not taken into account, mostly because there are different ways to deal with that in JtR.

```
[List.External:Filter_w2k3_default]
void filter()
{
    int i, c, s, n, u, l, x;
    i = 0;
    s = 0; // number of special chars
    n = 0; // number of digit
    u = 0; // upper case
    l = 0; // lower case
    x = 0; // counter
    while (c = word [i++]) {
        if (c >= '0' && c <= '9') {
            n++;
        } else if (c >= 'a' && c <= 'z') {
            l++;
        } else if (c >= 'A' && c <= 'Z') {
            u++;
        } else {
            s++;
        }
    }
    if (s !=0) x++;
    if (n !=0) x++;
    if (u !=0) x++;
    if (l !=0) x++;
    if (x < 3) {
        word = 0;
    }
    return;
}
```

Figure 3 – JtR external filter for the Windows 2003 Server default password policy

While this approach works, it has one big disadvantage: it has no influence on the creation of candidate passwords. Instead of creating a targeted list of password candidates, candidates are created according to rules which are then filtered (hence the term “external filter”). Ideally, password candidate creation would already take a policy into account. In a default setup, the above filter spends a substantial amount of time rejecting password candidates that really should not have been created in the first place.

It is acknowledged that the generate() function in a JtR external filter may be of use to modify the generation of password candidates in the manner described. However, the author is convinced that a more general way of describing password policies and using then as input into password cracking is called for, without requiring the user to write and debug C code, or any other programming language, for that matter.

## Conclusion

Password cracking is part of a security consultant's everyday work. Depending on the scope of the work and the goal to be attained, a number of different approaches to choose from present themselves. Scenarios like cracking one particular password of a highly privileged account may require a different approach than the bulk, overall evaluation of the quality of a database of hashes.

Also, several ways exist to determine which passwords to try and in what order. Brute force attacks, dictionary (word list) attacks, rainbow tables and lookup tables are all viable in different scenarios. Targeted lookup tables comprising of previously cracked passwords also appear to be very promising. And choice of hardware dedicated to password cracking also requires careful consideration.

Password policies are often recommended by security consultants as a blanket remediation for authentication weaknesses. However, as has been illustrated in this paper, they can be counterproductive and sometimes outright harmful to society.

We conclude that despite the common notion that password cracking is simple to perform and that a universal approach fits all scenarios, there is still plenty of scope for research to make password cracking attacks more targeted and subsequently more successful in the future.

## **Notes**

All tests were carried out on a laptop with Intel Core Duo T2300 1.66GHz processor unless stated otherwise.

## References

- [1] Cain & Abel website. <http://www.oxid.it/cain.html>
- [2] fgdump website. <http://www.foofus.net/fizzgig/fgdump>
- [3] John the Ripper website. <http://www.openwall.com/john>
- [4] Phenoelit default passwords website. <http://www.phenoelit.de/dpl/dpl.html>
- [5] The Shmoo Group Rainbow Tables website. <http://rainbowtables.shmoo.com>
- [6] Ross J. Anderson. Security Engineering: A Guide to Building Dependable Distributed Systems, chapter 3: Passwords. John Wiley & Sons, Inc., New York, NY, USA, 2001.
- [7] Gert Bon and Steffen van Loon. Password cracking in the field. 2006
- [8] Matt Bishop and Daniel V. Klein. Improving System Security via Proactive Password Cracking, 1992.
- [9] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off.
- [10] Robert Morris and Ken Thompson. Password security: a case history. Commun. ACM, 22(11): 594-597, 1979.

## About the Author

Thorsten Fischer is a Senior Security Consultant at Information Risk Management Plc (IRM) where he performs a range of consultancy services, including providing advice to clients about password quality evaluation, infrastructure architecture and IT security policies.

## About IRM

Information Risk Management Plc (IRM) is a vendor independent information risk consultancy, founded in 1998. IRM has become a leader in client side risk assessment, technical level auditing and in the research and development of security vulnerabilities and tools. IRM is headquartered in London with Technical Centres in Europe and Asia as well as Regional Offices in the Far East and North America. Please visit our website at [www.irmplc.com](http://www.irmplc.com) for further information.