Research Report

Password cracking in the field

operating systems and database management systems

Research project 1, 2006

System and Network Engineering
University of Amsterdam
Weesperzijde 190
Amsterdam

Authors
Gert Bon
Steffen van Loon

Table of contents

Preface

Password cracking in the field is about the implementations of encryption algorithms. The information on these implementations on several platforms is fragmented. This document tries to provide some grip on the implementation and weaknesses in widely used software like operating systems and database management systems.

Password cracking in the field is written for the Master of Science course System and Network Engineering in co-operation with Jeroen van Beek and Eric Nieuwland from KPMG.

The authors want to thank Jeroen van Beek and Eric Nieuwland for their contribution, support and feedback during this research project.

Gert Bon
Steffen van Loon

Introduction

This document is written for Research Project 1 for the course System and Network Engineering at the University of Amsterdam.

In this document the project goals for this project are described in "*Project goals"*.

In "*Gaining access to systems"* a few methods are briefly described to gain access to systems. Close to gaining access, methods for cracking passwords are discussed in "*Cracking methods*".

Strengths and weaknesses of implementations of encryption algorithms in operating systems and database management systems are described. These are found the chapters "*Implementation in operating systems"* and "*Implementation in database systems*".

After discussing the implementations of password security in operating systems and database management systems conclusions are drawn in chapter Conclusion.

Some elements of our research remained unsolved which remain a matter of future research. These elements can be found in chapter "*Future research"*.

For those who are not common in the world of cryptography, there is an "*Appendix A: General information*" which explains most common terms used in cryptography.

The legal notes are attached in Appendix B - Creative Commons Licence.

The original research report *Password cracking in the field* can be downloaded from http://www.os3.nl/~steffen/rp1/report.pdf. The original project initiation document for this research project can be downloaded from http://www.os3.nl/~steffen/rp1/pid.pdf.

Project goals

This document is written to describe the basic information of algorithms and the implementations of these algorithms in widely used software solutions like operating systems and database management systems.

The primary goal is to provide information about strength and weaknesses of different implementations of encryption algorithms in popular software. We will describe implicit weaknesses of some of these implementations, including the impact and risk of brute force and dictionary attacks and the possible use of rainbow tables.

The secondary goal is to provide a source of information containing advice and references for securing systems and networking containing the implementations as covered in this report. We think this document will give some grip in acquiring more information about particular in-depth information.

This document does not describe the possibilities of cracking different software implementations. This document will also not describe ways of penetrating system and network security, although we will give some references and links concerning this information if appropriate.

Gaining access to systems

To gain access to systems that require authentication, the first thing to do is search for a method to get in. The easiest way is acquiring user credentials that can be found in authentication information. This information could contain user credentials, in a plaintext value or hash value, depending on the used protocol.

The methods of obtaining user credentials or passwords are not relevant for this document. We will focus on cracking earlier acquired hash values, and are at this point not interested in how we acquired them. However, we will name a few methods in the following sections.

Wire sniffing

A passive attack method of gathering authentication information is wire sniffing. Attacks performed using wire sniffing requires physical access to the network. This is needed to sniff the cable and record raw network traffic.

The attacker has to wait until there are some authentication frames passing by, encrypted or not, before he can attack these credentials. Depending on how these credentials are encrypted it is possible to attack these credentials. Some protocols, like FTP and POP, even sends the user credentials unencrypted. In this case, when someone is wire sniffing, the credentials are given away in plaintext. Even if this method is considered, it's relatively hard to perpetrate and usually computationally complex. Tools are widely available, a good example is Ethereal.

Man-in-the-middle and replay attacks

Using online attacks with a man in the middle or replay attack require access to the network. This is needed to sniff the cable and record raw network traffic, from which authentication frames can be obtained.

With a MiM-attack the attacker has to wait for an authentication sequence so the attacker can imitate a proxy. With the acquired information, the proxy can make the connection to the other involving party. This way, the proxy can record all information that's passes him. With a replay attack, an attacker can repeat the captured frames to gain access.

Password guessing

This method is password guessing until one password works. This is made easier by:

Weak passwords (which is the core cause)

Open authentication points

Excessive information from server

Lack of password guessing controls

Consider this will take a long time and requires huge amounts of network bandwidth. It's also easily detected and stopped. Presume using a dictionary would save time.

Unsecured access to systems

One of the most interesting ways to acquire user credentials are systems that are not properly secured. This can be for example an operating system or web server where password files are world-readable or a database server where you can query interesting tables. This also includes exploiting implementation specific flaws, like a DLL-injection to gain access to a system process, or SQL-injection on a database server.

User credentials acquired via this way are generally encrypted somehow, mostly by applying a one-way hash-function. Our main purpose of this document is the decryption of this information.

Cracking methods

The purpose of this document is to describe what the possibilities are when encrypted passwords are found via methods such as those mentioned in the previous sections. There are different attacks possible to obtain unencrypted passwords from these encrypted passwords. These attacks can be performed offline at leisure.

When performing these attacks, there are some prerequisites to obtain the passwords. One has to know how the passwords were encrypted, which encryption method is used, and what the implementation-specific properties are. The structure of a hash has to be clear.

This information can be found in documentation on the used software package or the use of reverse engineering. Most documentation on used encryption algorithms in software packages can be found in books or the Internet. For some software implementations there are weaknesses known and in that case, there are already tools available to crack the passwords.

In the following sections the attack methods to retrieve the original password from the encrypted version are described.

Brute force attacks

A brute force attack is an attack on a password, where all possible character combinations are tried. A major advantage of this method is, that all passwords will be found. However, this way of attacking a password is very inefficient. It is a good and fast method for cracking short passwords, but when password lengths are longer, this method will get very slow.

Due to this characteristic, this attack is most of the time performed with progressive complexity. For example, first thing to do is trying alphanumeric key space, then alphanumeric with upper case, and then expanded with other symbols.

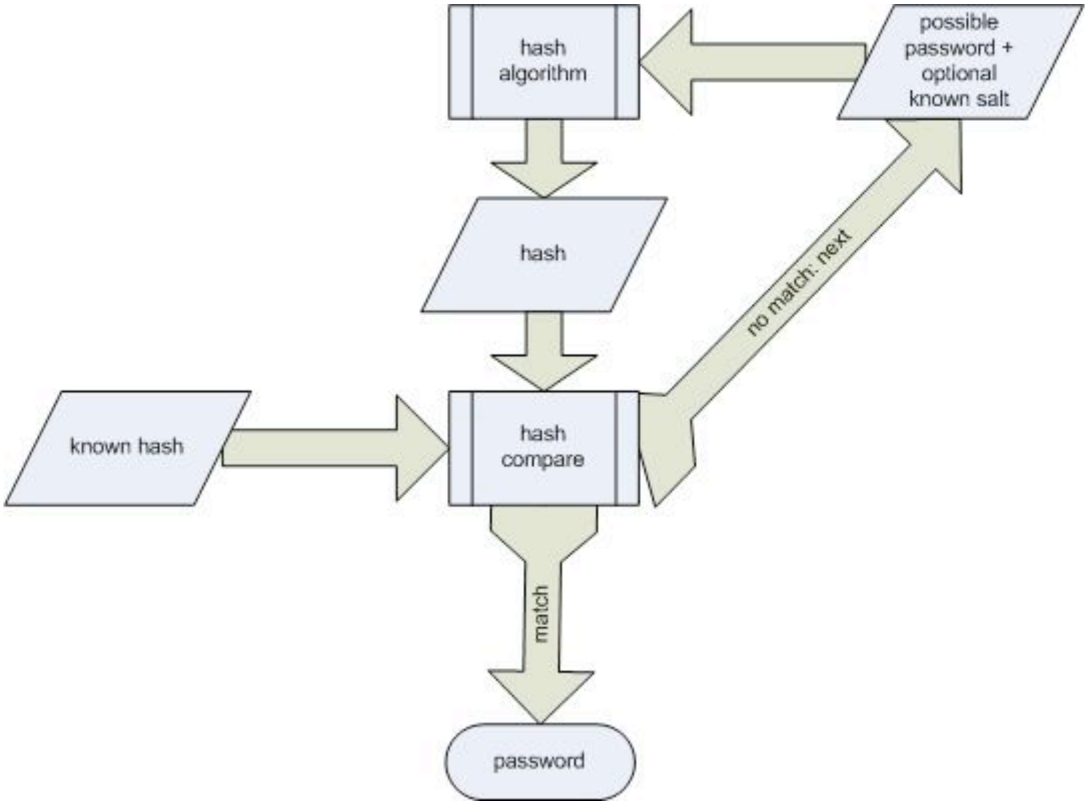


Figure 1 - Brute forcing a hash

Dictionary attacks

When using a dictionary attack all passwords from a specific list are tried. This attack utilizes the characteristic that many people use a word or a word combination as a password. Trying a large number of words can be significantly faster then a brute force attack.

The success factor of dictionary attacks depends on a big list of the chosen words. This list is mostly language-dependent also. The success rate of this method also highly depends on the type of passwords trying to crack.

For example passwords like "B_)r8wy#@3.!2" will not be found, because this password is not based on an existing word and therefore is not found in the wordlist.

Hybrid attack

A hybrid attack is a combination of the dictionary attack and the brute force attack. It combines a wordlist with some mutations on the entropy. This means that this method will generate new passwords based on the used wordlist, by using different techniques. The entropy can be altered by appending a symbol or number at the end of a word, or replacing letters with numbers. For example the dictionary word "password" generates "password31" or "p4$Sw0rd".

The speed of this method is highly dependable on the alterations made on the wordlist. When expanding the word "password" to "password<XX>" with XX in character set [a-z,0-9], the complexity increases almost 1300 times. With more advanced alterations, this number will be even higher. This method is more efficient than a brute force attack. However, in comparison with a dictionary attack this method is more time-consuming.

Both hybrid and dictionary methods attack the main problem with security; weak passwords. Both methods are ineffective against well-chosen long passwords, which are not based on exiting words using a combination of letters, numbers and special characters.

Pre-computation attacks

The first concept of pre-computating passwords using a fast memory trade-off theory was already proposed in 1980 by Martin Hellman1 and was refined by Ronald Rivest in 19822. Recently there was some new development on this technique. This attack method is further described in the section rainbow tables.

Rainbow tables

This method is based on generating all possible hash-values and stores them together with the password in a table. When attacking, only the corresponding hash-value has to be looked up in the database. This technique makes it dramatically faster, up to a factor of over 12 times faster than traditional methods.

Storing all hashes requires huge amounts of storage space (example: all LM-hashes take 166 Terabytes, all NT-hashes less than 15 characters takes 140,959,235,198 exabytes). The solution here is to use a time-space tradeoff, like the one developed by Philippe Oechslin known as rainbow tables. This method generates most hashes and store the commonality, and then brute force within the corresponding set. This attack only works on algorithms that do not use salting.

Other attacks

One of the most common and successful examples of a non-technical attack is "shoulder surfing"; this is when someone is watching while typing a password.

The most common known is "social engineering". This requires some good socials skills and preparation. A good example about social engineering is the chocolate bar example, which can be found at http://zdnet.com.com/2100-1105_2-5195282.html.

Another technical method is "keyboard sniffing". This can be done with hardware and software, which is cheap and hard to detect. These attacks are however out of scope for this document.

Implementation in operating systems

This section describes the implementation of encryption algorithms in widely used operating systems in this section. Links are provided as reference for more in-depth information.

Windows Networking

Microsoft LAN Manager (LM)

For Windows networking (including Active Directory domains) the password is stored in two different ways by default, namely as an LM-hash and an NT-hash. The biggest weakness of these methods is that in neither case a salt is used by calculating the hash-value.

LAN Manager Hash (LM hash) is one of the formats that Microsoft LAN Manager and Microsoft Windows uses to store user passwords. LM hash was developed in the seventies by IBM for use with IBM 360/370 series. The technology was adopted by LAN Manager, which was a joint venture of Microsoft and IBM, in the eighties. The algorithm is still included in recent versions of Windows for backward compatibility and activated by default.

The LM-hash is based on DES, and can be is easily subverted. In LM authentication the password is case-insensitive, restricting each character to either one of the 26 letters or a special character. Long passwords, up to 14 characters, are divided into 7-character chunks. The combination of a small character set and the password division results in a relatively small key space.

Due to the small key space successful brute force attacks are possible on LM hashes. For example a character set [A-Z] + [0-9] and a password length of 5 characters will result in a key space of only 36^5 = 60,466,176. If the character set was chosen out of [a-z] + [A-Z] + [0-9] it would result in a key space of 62^5 = 916,132,832. Notice that by increasing the length of the character set the key space grows exponentially. Because dividing the passwords into 7 character chunks, the total key space for LM Hash is 69^7 ≈ 7.4 trillion. This is not consequently for all passwords with a length up to 14 characters. More detailed information can be found in Peter Mudge's rant[3].

An example

Let's calculate the time it takes to brute force 1 password with a maximum length of 7 characters. The second part of the hash in this case is always the same because this part is padded with NULL. Assume we can crack 3,000,000 crypts per second on a 1.4 GHz machine with 1024 MB RAM running John. As we have calculated before; the total key space 7,446,353,252,589, and use 0.5 as success factor indicating a random password. On average the password will be found when we are halfway through all possibilities.

Character set (C)	69 characters
Crypts per second (A)	3,000,000
Key space (K)	7,446,353,252,589
Success factor (S)	0.5

T = time in seconds
$(K / A) \cdot S = T$
$(7{,}446{,}353{,}252{,}589 / 3{,}000{,}000) \cdot 0.5 = 1{,}241{,}058.88$ seconds.

This is only 14,36 days which is far less than most passwords expiration times. In the worst case it takes 28.72 days which is still less than most passwords expiration times.

In many cases only an alphanumeric character set is used to compose passwords. This will reduce the attack time significantly. The following calculation has a smaller character set and shows how much time it saves.

K = key space
C = character set
N = password length
$K = C^N$

$36^7 = 78{,}364{,}164{,}096$

$(78{,}364{,}164{,}096 / 3{,}000{,}000) \cdot 0.5 = 13{,}060.69$ seconds.

This is only 3.6 hours and in the worst case 7.2 hours.

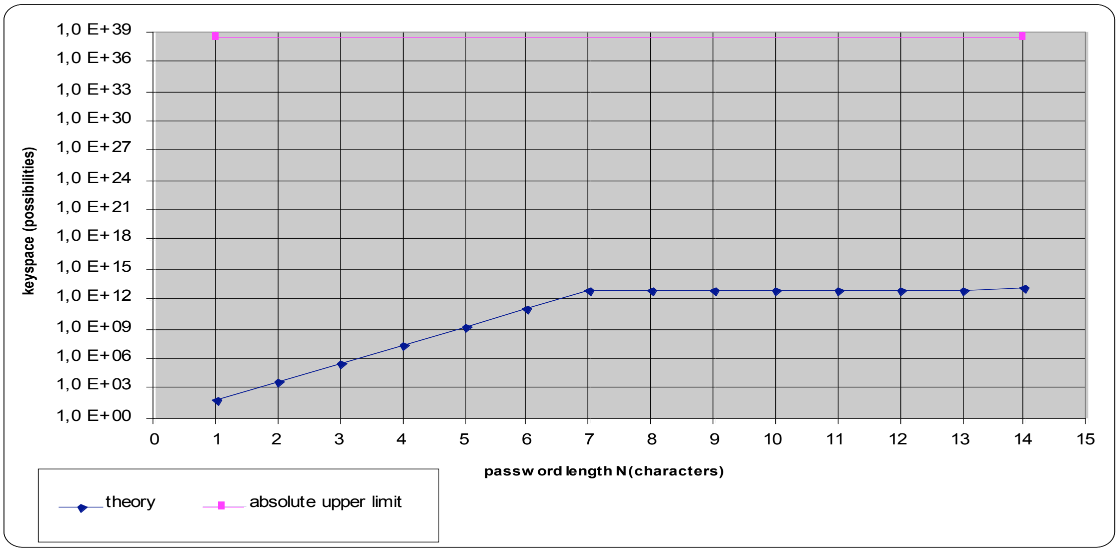
Summary

Design

Name:	LAN Manager hashing.
Applications:	Microsoft Windows 95 series, 98 series, ME, NT series, 2000 series, XP series, Server 2003 series.
Platforms:	See applications.
Used for:	Hashing Windows user passwords.
Algorithms used:	DES.
Algorithm input:	Password; in uppercase and when less then 14 characters the password is padded with NULL to 14 characters.
Salting:	No.

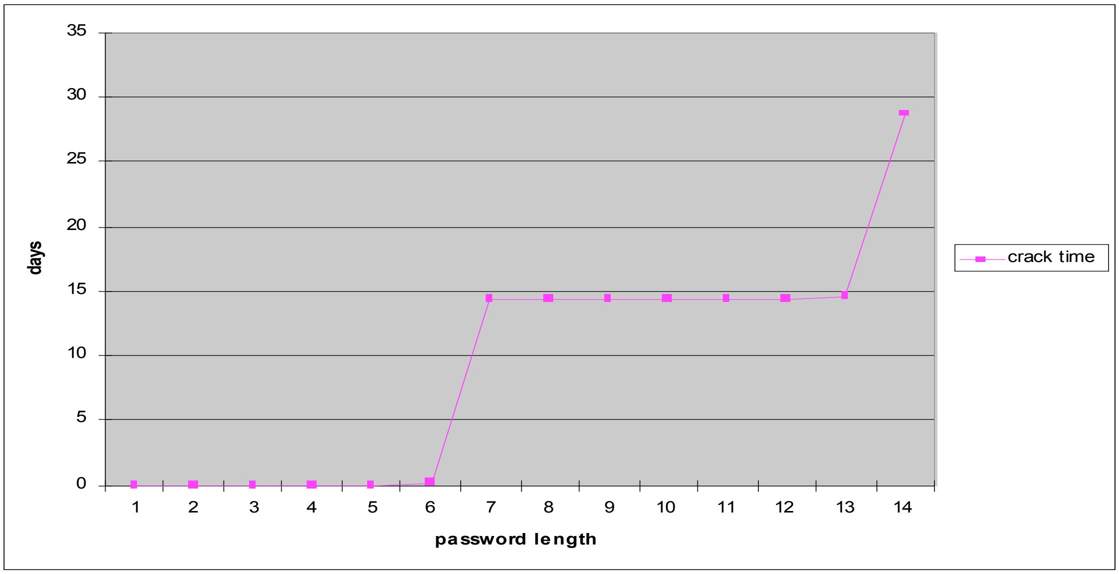
Implementation

Character set:	69 characters: [A-Z][0-9] and 33 special ALT characters: [!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/space].
Limitations for character set:	Standard character set: Length of password is >= 0 and <= 14
Hash storage	Location: "%WINDIR%\system32\config\sam" (SAM database). Size: 16 bytes: 8 bytes for first 7 characters and 8 bytes for characters 8 to 14. Format: Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0::: <username><rid><uppercase hash><hash>
Theoretical key space:	$K = C^N$ $7,446,353,252,589 = 69^7$
Key space limitations:	A hash collision will occur after 2 ^ (storage bits) passwords as documented in "Hash storage". $= 2^{128} = 3.40 \cdot 10^{38}$



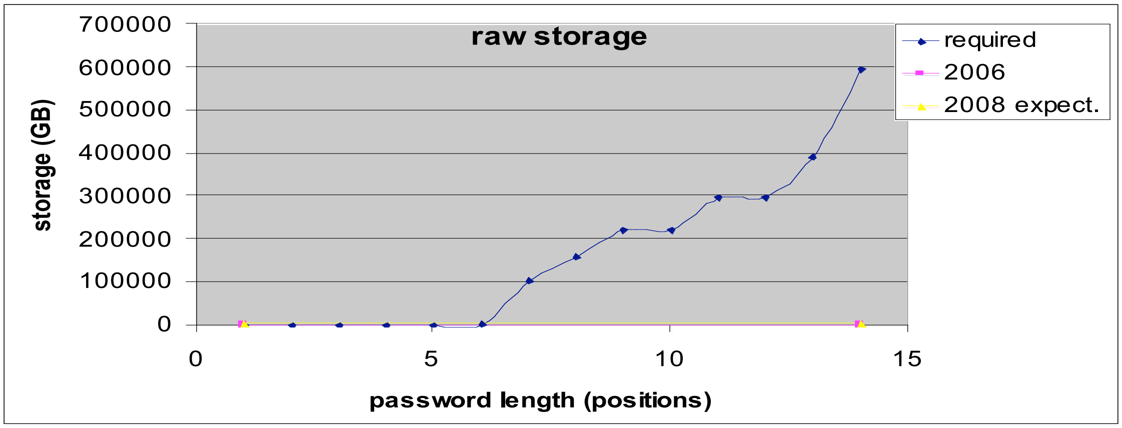
Key space for LM

The graph above shows the key space based on a 69-long character set.



Attack time for LM

The graph above shows the attack time based on a 69-long character set.



Storage for LM

The above graph is logarithmic and shows the raw disk storage for storing both hashes and passwords based on a 69-long character set.

NTLMv1

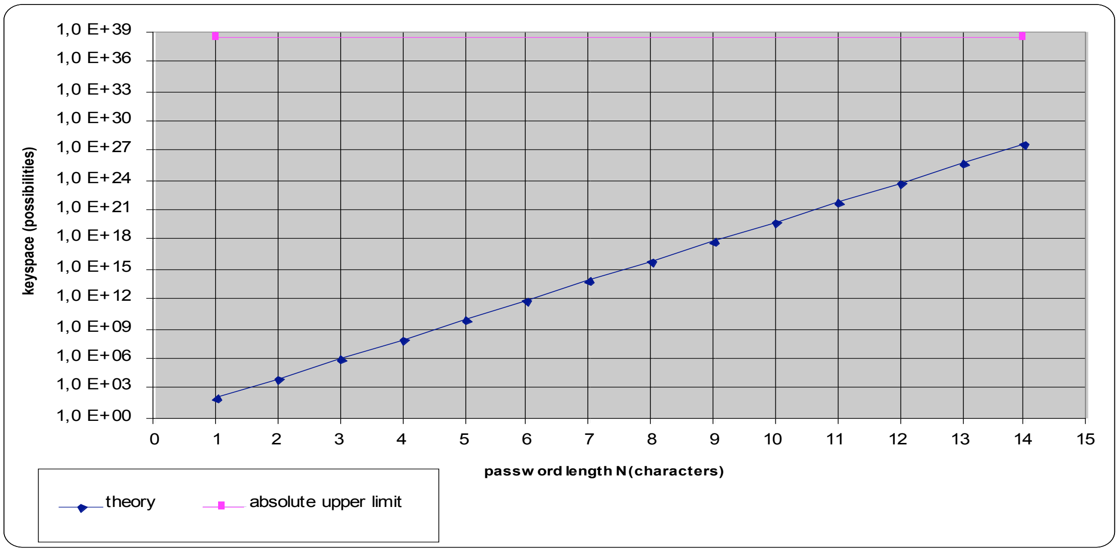
Microsoft recognized the vulnerabilities in LM Hash, and introduced the significant improved NTLM Hash. These vulnerabilities included the relative small character set, which implicit a smaller key space, and the division of the password. The key space grows exponentially because the effective password length was increased. The NT-hash is used for authentication by domain members in for example Windows NT 4, 2000, XP, 2003 and Active Directory Domains. The NT-hash is generated from the password with the MD4 algorithm, which creates a 16-byte one-way hash.

Dictionary attacks on NTLM are still a very good method to crack weak passwords, but brute force attacks are a lot harder now in comparison with LM hash. The weakness of this protocol is that it does not offer any signing or encryption of the exchange of messages between clients and servers. The protocol is vulnerable for "chosen plaintext" attacks, using message injection techniques by an attacker.

Summary

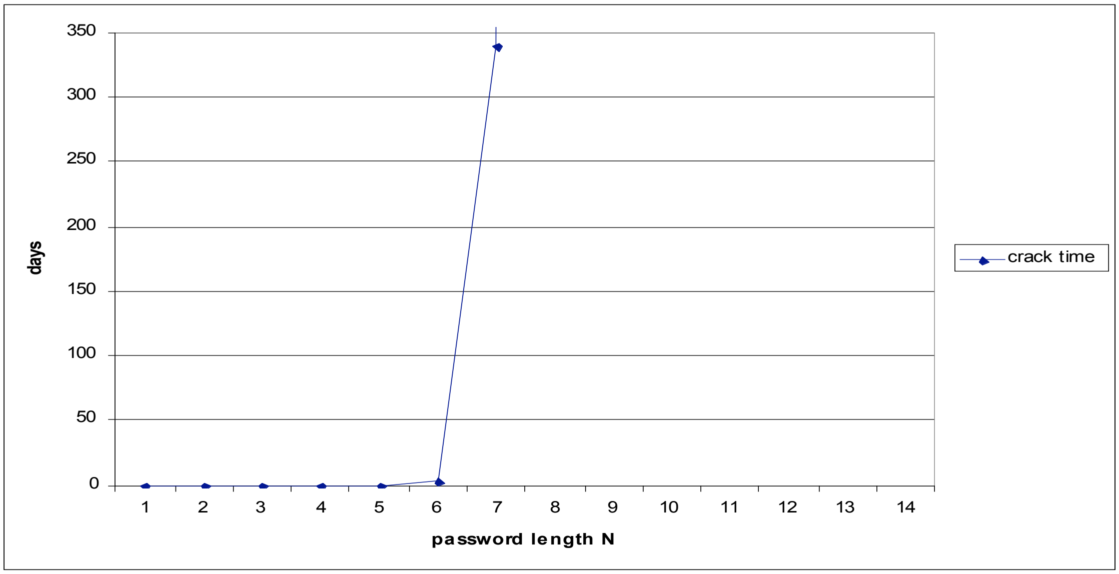
Design	
Name:	NT LAN Manager hashing version 1
Applications:	Microsoft Windows 95 series, 98 series, ME, NT series, 2000 series, XP series, Server 2003 series.
Platforms:	See application.
Used for:	Hashing Windows user passwords
Algorithms used:	MD4.
Algorithm input:	Password.
Salting:	No

Implementation	
Character set:	65,536 characters.
Limitations for character set:	Length of password is >= 0 and <= 128
Hash storage	Location: “%WINDIR%\system32\config\sam“ (SAM database). Size: 16 bytes Format: Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0::: <username><rid><uppercase hash><hash>
Theoretical key space:	$K = C^N$ $3.232 \cdot 10^{616} = 65{,}536^{128}$
Key space limitations:	Theoretically, a hash collision will occur after 2 ^ (storage bits) passwords as documented in “Hash storage” $= 2^{128} = 3.40 \cdot 10^{38}$



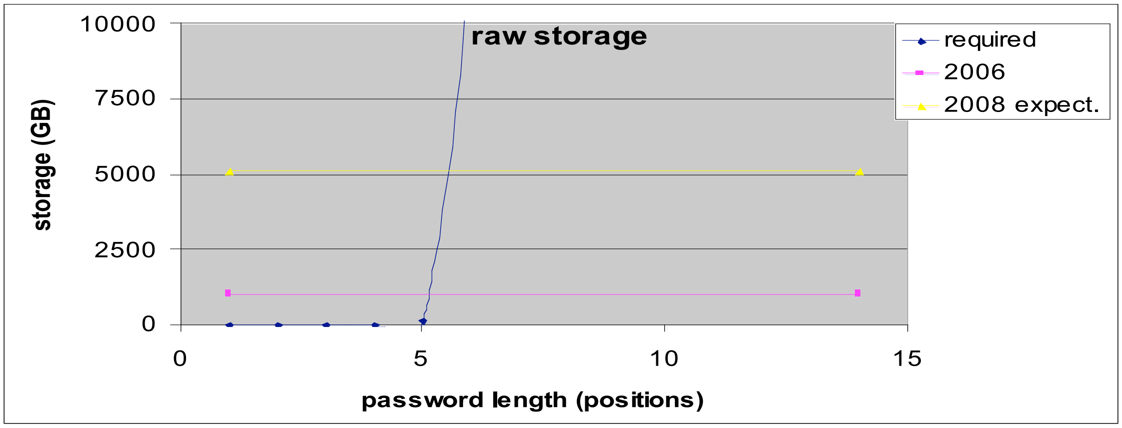
Key space for NTLMv1 & NTLMv2

The graph above shows the key space based on a 95-long character set.



Attack time for NTLMv1 & NTLMv2

The graph above shows the attack time based on a 95-long character set.



Storage for NTLMv1 & NTLMv2

The graph above shows the raw disk storage for storing both hashes and passwords based on a 95-long character set.

NTLMv2

After a weakness was identified in NTLMv1 Microsoft introduced NTLMv2. NTLMv2 added another enhancement to NTLMv1, which makes it more secure: the challenge / response mechanism was improved. NTLMv2 provides enhanced session security negotiation. It provides separate keys for message integrity and confidentially. The client inputs the challenge to prevent chosen plaintext attacks and makes use of the HMAC-MD5 algorithm (see RFC 2104) for message integrity checking.

Summary

Design	
Name:	NT LAN Manager hashing version 2
Applications:	Microsoft Windows 95 series, 98 series, ME, NT series, 2000 series, XP series, Server 2003 series.
Platforms:	See application.
Used for:	Hashing Windows user passwords
Algorithms used:	MD4.
Algorithm input:	Password.
Salting:	No

Implementation	
Character set:	65,536 characters.
Limitations for character set:	Length of password is >= 0 and <= 128
Hash storage	Location: "%WINDIR%\system32\config\sam" (SAM database). Size: 16 bytes Format: Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0::: <username><rid><uppercase hash><hash>
Theoretical key space:	$K = C^N$ $3.232 \cdot 10^{616} = 65{,}536^{128}$
Key space limitations:	Theoretically, a hash collision will occur after 2 ^ (storage bits) passwords as documented in "Hash storage" $= 2^{128} = 3.40 \cdot 10^{38}$

Concluding on LM, NTLMv1 and NTLMv2 hashes

The weakness of Windows and most other software using the LM, NTLMv1 and NTLMv2 protocols is that they are enabled by default. To secure the systems and networks using these protocols, the recommendation is to use NTLMv2 only. This will prevent attacks on LM or NTLMv1-hashes. This is however not always an option, especially when backwards compatibility is required.

There are many references about this topic available; here are some we found useful:

MIT – Authentication and Security (http://web.mit.edu/ist/topics/windows/server/winmitedu/security.html)

Microsoft TechNet Security Management column – FAQ about passwords (http://www.microsoft.com/technet/community/columns/secmgmt/sm1005.mspx)

Implementing CIFS (http://www.ubiqx.org/cifs/)

AIX

AIX 5L (abbreviation for Advanced Interactive eXecutive) is IBM's[4] latest operating system based on UNIX System V. The AIX abbreviation was originally Advanced IBM Unix.

AIX 5L uses the UNIX default user password encryption function *crypt_unix*. Crypt uses the *DES* algorithm for generating passwords which are locally stored in */etc/security/password*. Username and password are stored in a format like: sueuser:XgYgQDm8Itpe2. The standard character set for passwords has a total of 128 characters. In practice, this is only 95 characters because control characters are not used.

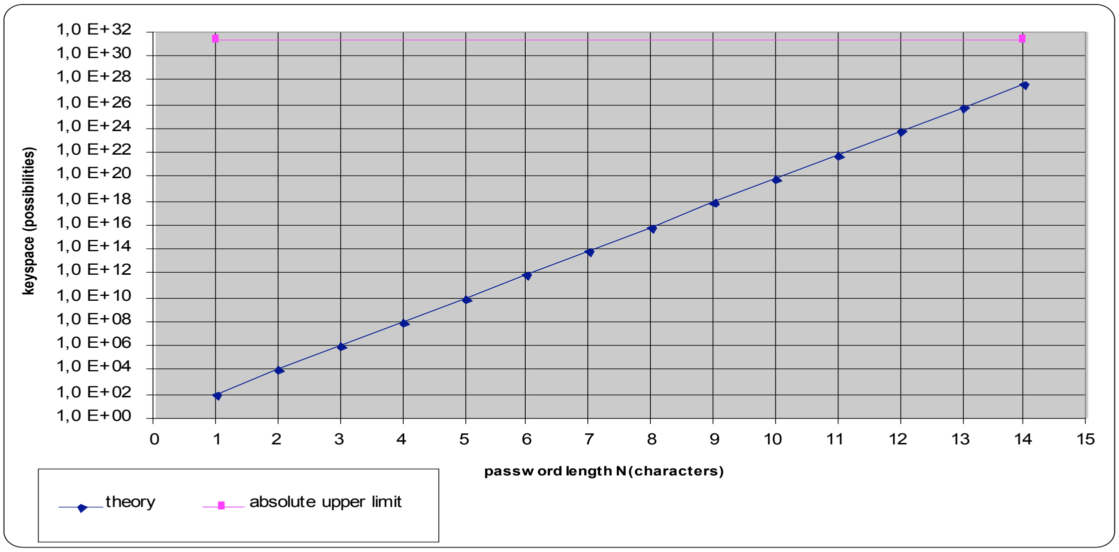
Input for the algorithm is *silently* the passwords first 8 characters and a salt from the function *crypt()*. Passwords longer then 8 characters are cut down to the first 8 characters. The 2 character salt is generated by *crypt_gensalt()* from the character set [a-zA-Z0-9./]. When the non-default algorithm MD5 is used there is no salt unless the rounds for the generation of the salt are specified.

Because salts are used it is currently impossible to use pre-generated rainbow tables as an attack due to the huge amounts of storage it requires. Generating a rainbow table costs a lot of disk space. When the non-standard MD5 algorithm is used rainbow tables can't be used either, unless it is known how many rounds are used for the hash generation. A brute force attack seems most efficient. In 1998 EFF[5] showed 'Deep Crack' known as EFF's DES Cracker. The cracking machine contained hundreds of custom chips and could brute force a DES key in 3 days.

Summary

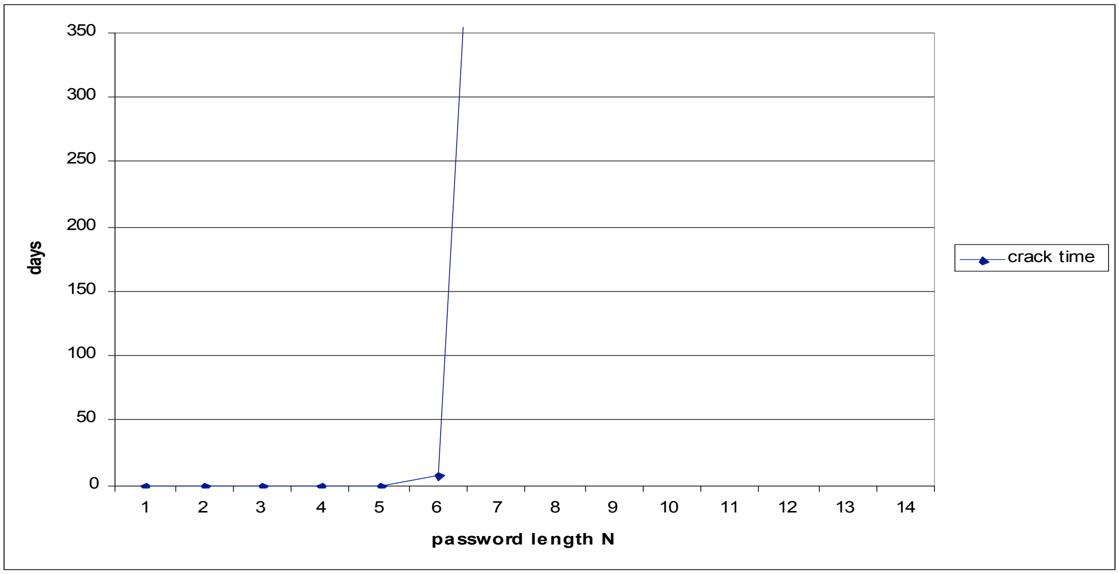
Design	
Name:	AIX Password hashing
Applications:	AIX 5L
Platforms:	AIX
Used for:	User password encryption
Algorithms used:	Default crypt_unix. (crypt = DES) Optional MD5
Algorithm input:	The first 8 characters of the password + salt from crypt().
Salting:	The crypt() function calls crypt_gensalt(3c) to generate the 2 character salt chosen from the set [a-zA-Z0-9./]. MD5; specify rounds used for generation of the salt.

Implementation	
Character set:	95 characters.
Limitations for character set:	Length of password is >= 0 and <= 8
Hash storage	Location: /etc/security/passwd Size: 13 bytes (default crypt() based) Format: sueuser:XgYgQDm8Itpe2 (default crypt() based)
Theoretical key space:	$K = C^N$ 95^8 = 6,634,204,312,900,000
Key space limitations:	Theoretically, a hash collision will occur after 2 ^ (storage bits) passwords as documented in "Hash storage" = 2^{104} = 20,282,409,603,651,670,423,947,251,286,016



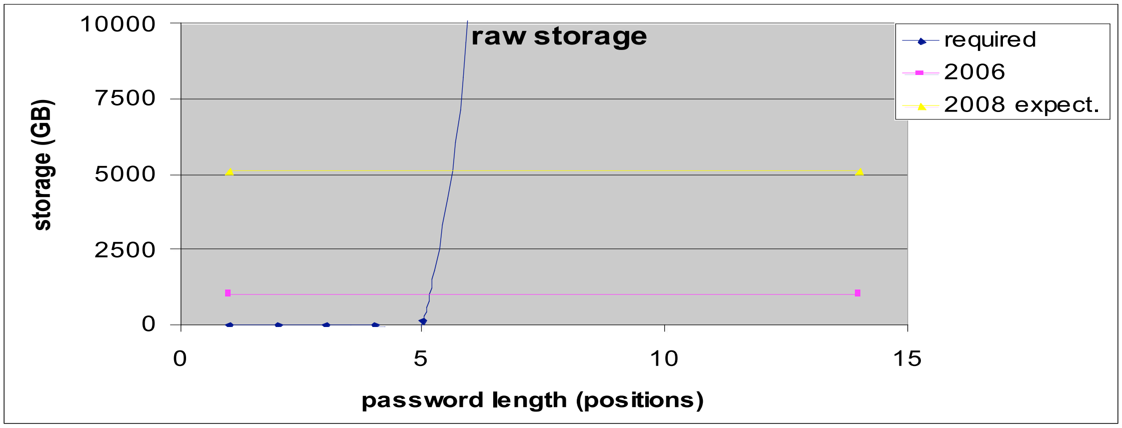
Key space for AIX

The graph above shows the key space based on a 95-long character set.



Attack time for AIX

The graph above shows the attack time based on a 95-long character set.



Storage for AIX

The graph above shows the raw disk storage for storing both hashes and passwords based on a 95-long character set.

Solaris 10

Solaris is an operating system developed by Sun Microsystems and is a certified version of UNIX. Solaris 10 is based on UNIX System V .

Solaris 10 uses, like AIX, the UNIX default user password encryption function *crypt_unix*. The *DES* algorithm is used in the *crypt()* function for generating encrypted passwords which are stored locally in */etc/security/passwd*. An US ASCII character set (128 characters) is used or eventually an international character set (256 characters). In practice this is only 95 characters due to the fact that control characters are not used.

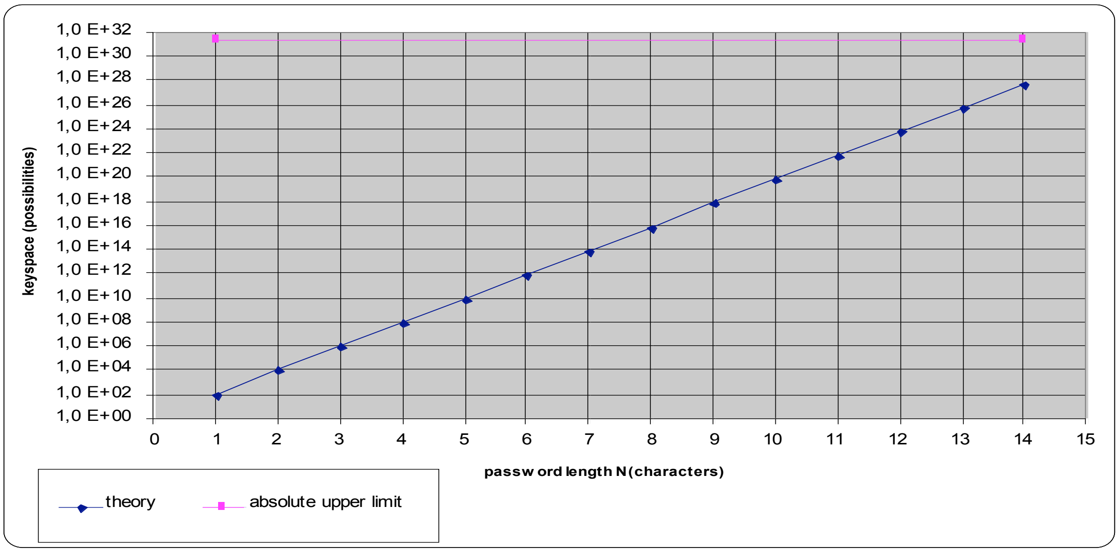
The input for the *crypt()* function are the first 8 characters from the password and a 2 character salt which is generated by *crypt_gensalt()*. This salt is generated from the character set [a-zA-Z0-9./]. Stronger password encryptions like MD5 and blowfish are also supported. Besides it is possible for MD5 to specify the number of rounds which are used to generate the salt it is also possible to use passwords with a length up to 255 characters.

The salts make it currently impossible to use pre-generated rainbow tables as an attack due to the huge amounts of storage it requires. When the non-standard MD5 algorithm is used rainbow tables can't be used either, unless it is known how many rounds are used for the hash generation. A brute force attack seems most efficient. In 1998 EFF[6] showed 'Deep Crack' known as EFF's DES Cracker. The cracking machine contained hundreds of custom chips and could brute force a DES key in 3 days.

Summary

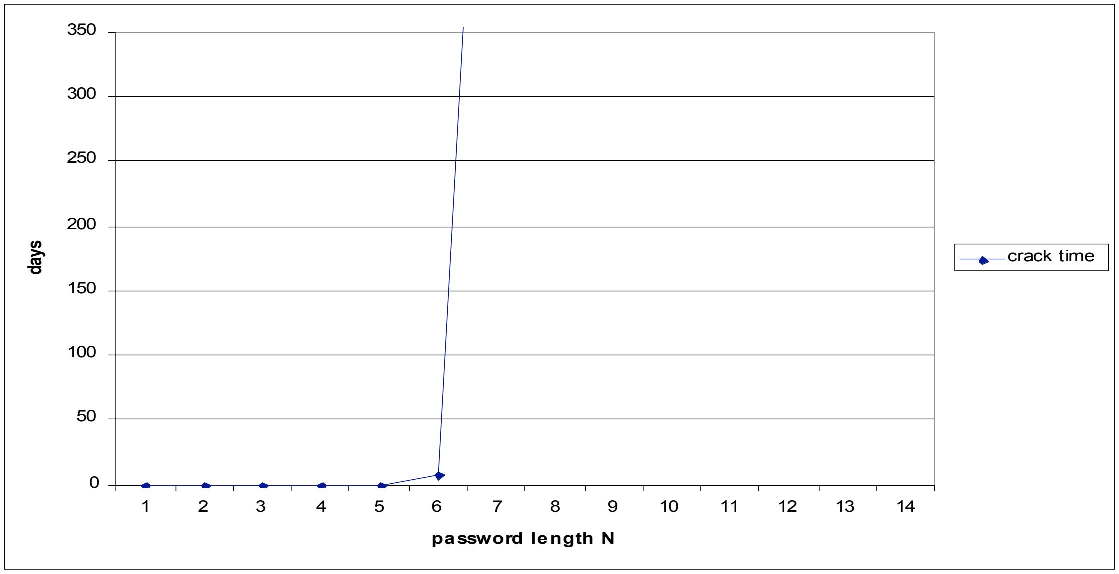
Design	
Name:	Solaris Password Hashing.
Applications:	Solaris 10.
Platforms:	Solaris 10.
Used for:	User password encryption.
Algorithms used:	Default crypt_unix. Also stronger password encryption options like MD5 & blowfish. (crypt = DES)
Algorithm input:	The first 8 characters of the password + salt from crypt().
Salting:	The crypt() function calls crypt_gensalt(3c) to generate the 2 character salt chosen from the set [a-zA-Z0-9./]. MD5; specify rounds used for generation of the salt.

Implementation	
Character set:	128 characters for US ASCII 256 characters for the international set 95 characters in practice
Limitations for character set:	Crypt has a maximum length of 8 characters MD5 has a maximum length of 255 characters
Hash storage	Location: /etc/security/passwd Size: 13 bytes (default crypt() based) Format: sueuser:XgYgQDm8Itpe2 (default crypt() based)
Theoretical key space:	$K = C^N$ In practice: 95^8 = 6,634,204,312,900,000 US ASCII with crypt(): 128^8 = 72,057,594,037,927,936 International with crypt(): 256^8 = 18,446,744,073,709,551,616 International with MD5: $256^{255} = 1.26 \cdot 10^{614}$
Key space limitations:	Theoretically, a hash collision will occur after 2 ^ (storage bits) passwords as documented in "Hash storage" = 2^{104} = 20,282,409,603,651,670,423,947,251,286,016 Limitation for MD5: 3.40 · 1038 = 1632 (32 long and hexadecimal)



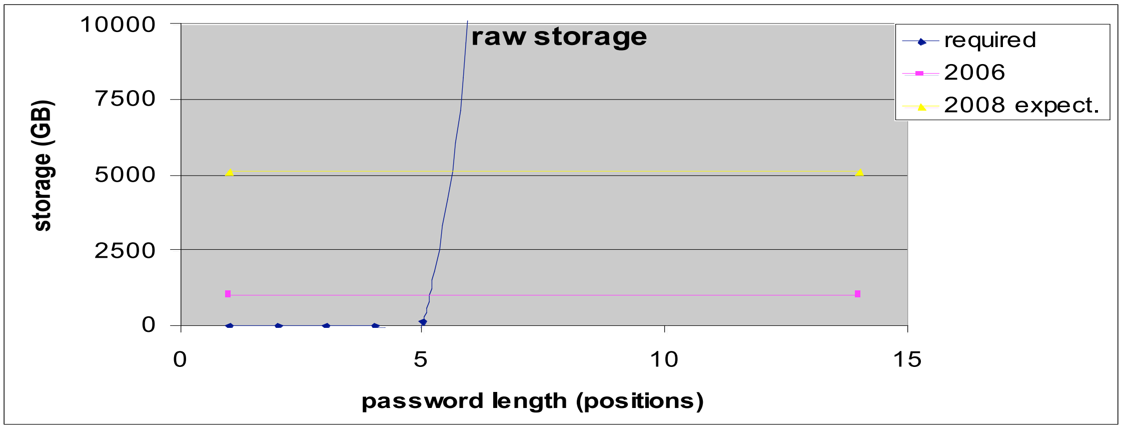
Key space for Solaris

The graph above shows the key space based on a 95-long character set.



Attack time for Solaris

The graph above shows the attack time based on a 95-long character set.



Storage for Solaris

The graph above shows the raw disk storage for storing both hashes and passwords based on a 95-long character set.

Cisco Internetworking Operating System

Cisco password hashing is used in common Cisco hardware i.e. routers and switches. The algorithms used to encrypt user passwords is Base64 encoded MD5 hashes (type 5) or a weak Cisco two-way algorithm (type 7). Passwords are based upon a default 7 bit US ASCII character set and hashes of these passwords are stored in the configuration file. Optionally it is possible to select an 8 bit international character set. This means 128 (7 bit: 27) or 256 (8 bit: 28) different characters can be used. These passwords are compatible with hashes used by OpenBSD or FreeBSD.

The input for the encryption algorithm of type 5 is a password and optionally a secret (set with *enable secret*). For example we have the Base64 formatted MD5 hash $1$6Je2$MurE4FTzoZjQShRW4Ui9H0. The $ in the MD5 hash separates the fields into three parts. $1 is the version, $6Je2 is the salt and the final part $MurE4FTzoZjQShRW4Ui9H0 is the Base64 encoded password hash. We assume a maximum password length of 100 characters. This maximum was advised by Cisco regarding a password buffer overflow attack.

Type 7 passwords look like 075F314940470E171E060E.

Brute force, eventually with a dictionary, attack would be the best option to crack passwords when *enable secret* is set and password is type 5. Otherwise it would be possible to use a rainbow table attack for type 5 with pre-computated MD5 hashes which are Base64 encoded.

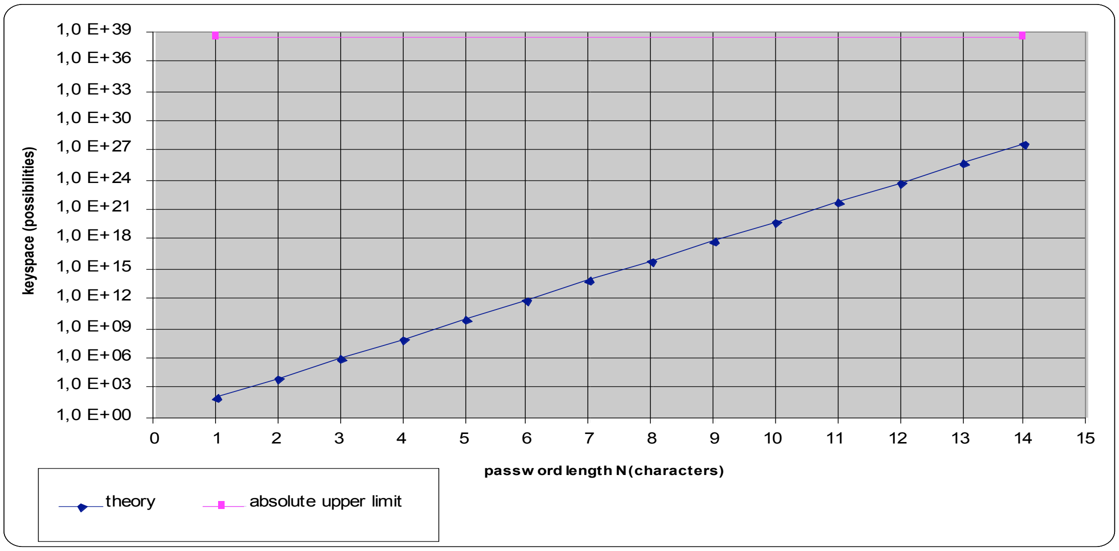
Type 7 hashes can be decoded easily since they are not encrypted but encoded. A number of tools is available to perform decoding: for example a Perl implementation can be downloaded from http://www.insecure.org/sploits/cisco.passwords.html

For Cisco hardware it's safest to use type 5 passwords with *enable secret* set.

Summary

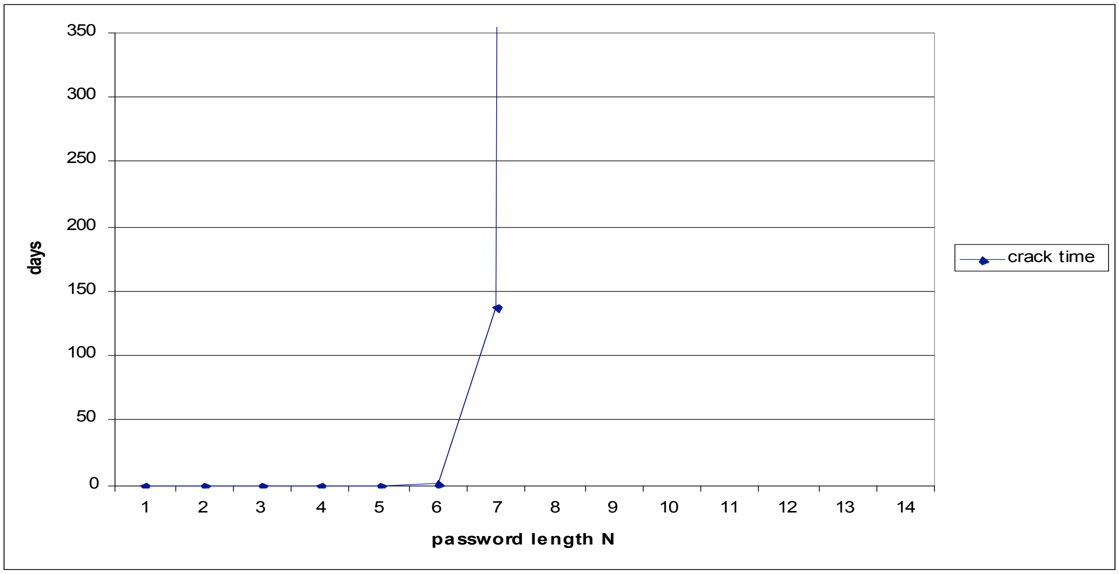
Design	
Name:	Cisco IOS password hashing.
Applications:	Cisco IOS.
Platforms:	See applications.
Used for:	Password access encryption.
Algorithms used:	Base64 encoded (MD5 hashes). (type 5) Or a proprietary weak Cisco algorithm (type 7) enable secret 5 $1$6Je2$MurE4FTzoZjQShRW4Ui9H0 The $ in the MD5 hash separates the field into three parameters: <version><salt><base64(MD5 password hash)>
Algorithm input:	Password, optional a secret.
Salting:	Optional.

Implementation	
Character set:	US-ASCII 128 characters International 256 characters
Limitations for character set:	
Hash storage	Location: Configuration file Size: 16 bytes Format: $1$6Je2$MurE4FTzoZjQShRW4Ui9H0
Theoretical key space:	$K = C^N$ $128^{100} = 5.26 \cdot 10^{210}$
Key space limitations:	Due to limitations of MD5 this is 3.40 · 1038 (hexadecimal & 32 characters long) or hash collisions occur.



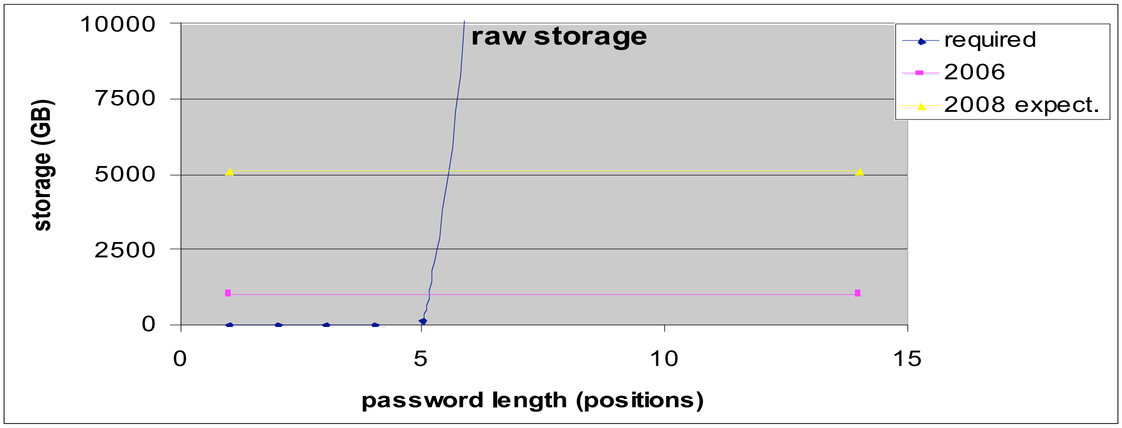
Key space for Cisco IOS

The graph above shows the key space based on a 95-long character set.



Attack time for Cisco IOS

The graph above shows the attack time based on a 95-long character set.



Storage for Cisco IOS

The graph above shows the raw disk storage for storing both hashes and passwords based on a 95-long character set.

Implementation in database systems

Implementation of encryption algorithms in widely used database systems

In the past years, focus on security was a big item. Operating systems are hardened and user applications are audited, but database systems were untouched. This is a strange development, because database systems are the pillars of all important company processes and a lot of important data is stored here.

Oracle

One of the main database software manufacturers is Oracle. Oracle is developing database products since 1979 (on a PDP-11), and is used by many multinationals. Oracle database systems are recently under attack, according to different news sites, like:
http://news.com.com/Oracle+password+system+comes+under+fire/2100-1002_3-5918305.html?tag=nefd.top.

The Oracle password algorithm was not public until recently. The SANS-institute published *"An Assessment of the Oracle Password Hashing Algorithm"*[7], which can be found at http://www.sans.org/press/oracle_pass.php. This paper releases details on how passwords are encrypted before being stored in the Oracle database.

A renowned security expert and researcher at SANS named Joshua Wright released details on how to breech the password hashing algorithm used by Oracle. He demonstrated an attack tool he wrote that recovers plaintext passwords from even very strong, well written passwords within minutes.

In order to abuse the weaknesses described in papers, an attacker needs to have knowledge of the password hashes of a database user. Obtaining this information is out of scope for this document, but can be done in a number of ways, like SQL injection, access to the host operating system, access to backup tapes, etcetera.

The released documentation and recent postings on a number of mailing lists discusses the techniques Oracle uses to store an encrypt user passwords in the database. This highlights a number of weaknesses in these techniques, such as a weak hashing algorithm and the lack of case preservation.

Hashing

Oracle can handle passwords up to 30 characters long. All these characters are converted to uppercase before the hashing algorithm starts. This algorithm creates an 8-byte hash using a modified DES encryption algorithm. The username is used as a salt for this algorithm, by concatenation of the username with the password before this is feed to the algorithm.

The Oracle password hashes are stored in the database, in the table SYS.USER$ - Password

However, the hashes can also be found on other locations, like the Oracle Password File, a Data File of the system tablespace, export files and archive logs.

Cracking

There are a number of password crackers available for Oracle, which uses brute force and dictionary attacks. Before the publication of the algorithm, cracking was done with PL/SQL based crackers. Now there are some C-based crack tools available. The most popular is probably "Orabf"[8], it is a pretty fast tool which does an average of 1,100,000 hashes per second on a standard Pentium 4.

Other issues

One of the biggest problems of Oracle is the use of default passwords, sometimes even needed by bad software implementations based on an Oracle database. There are multiple wordlists available with standard username and password combinations, like the one from Pete Finnigan (at http://www.petefinnigan.com/default/default_password_list.htm).

Another big issue is that user rights are not managed strictly enough. Limiting user rights is the first step in preventing that hashes become available for cracking purpose.

There is a lot of information available to secure Oracle systems. Oracle has released a paper about Oracle Database Hardening[9]. Some references can be found at SecurityFocus[10] and the website of Pete Finnigan[11].

Rainbow tables

In the paper SANS released, Josh Wright already mentioned the use of rainbow tables on Oracle databases. He created a proof of concept and describes his findings.

Since the release of his paper, no other publications concerning the use of rainbow tables on Oracle showed up. Because of that, we contacted Josh Wright. His reaction on our question why there where not any further publications regarding his findings, he answered that he did not released the source code of the path on the RainbowCrack tool. Reason is that he is concerned about the negative effect the release of his code would have on the security status of the Oracle databases. His concerns are that Oracle customers do not have an alternative available yet. Oracle's position is that they are planning to address a new password hashing algorithm in the future, but they do not have a solution available yet.

The potential abuse with the use of rainbow tables is enormous he said, cracking a password is reduced to a 100% hit ratio within minutes. Generating tables is quite practical, and it works well due to the weaknesses in the hashing algorithm. Due to the fact that there is no case preservation and the character set used is quite small, tables can be generated easily. Only drawback is the use of username as salt, making the generated tables only applicable on the specific user account. However, generating a table with the username SA or SYSTEM, grants you all rights.

For now, we can not give more information about this topic. It is now wait and see when Josh releases his source, or when someone releases an alternative patch to the RainbowCrack tool.

Advice

Even with the potential risk of rainbow tables used on Oracle, there are a number of thinks you can consider to do:

Do not use the same username on multiple machines. Instead concatenate the system name, or something other variable like the database name, to the username. The password hash will be different, due to the change of the salt. This reduces the change that if a password is broken, the attacker gains access to multiple machines.

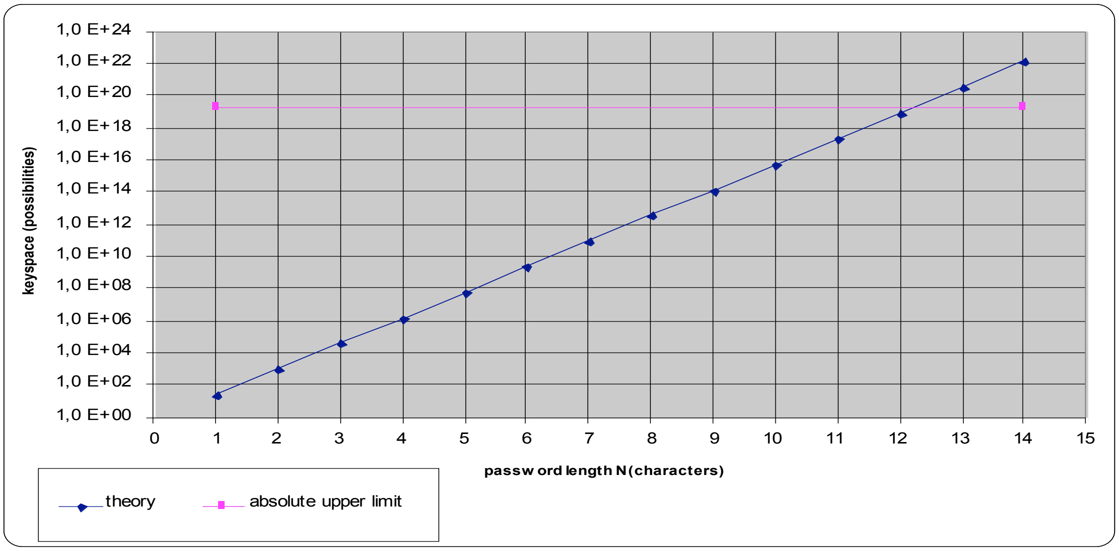
Important account names which can be found on all systems, like SA or SYSTEM, are first subject to (precomputation) attacks. The obvious solution for this is that these accounts should have very strong passwords, including a mix of digits and punctuation marks and at least a password length of 10 or more.

Also disable all accounts that are not used, especially accounts with more rights, and change default usernames and their corresponding passwords.

Summary

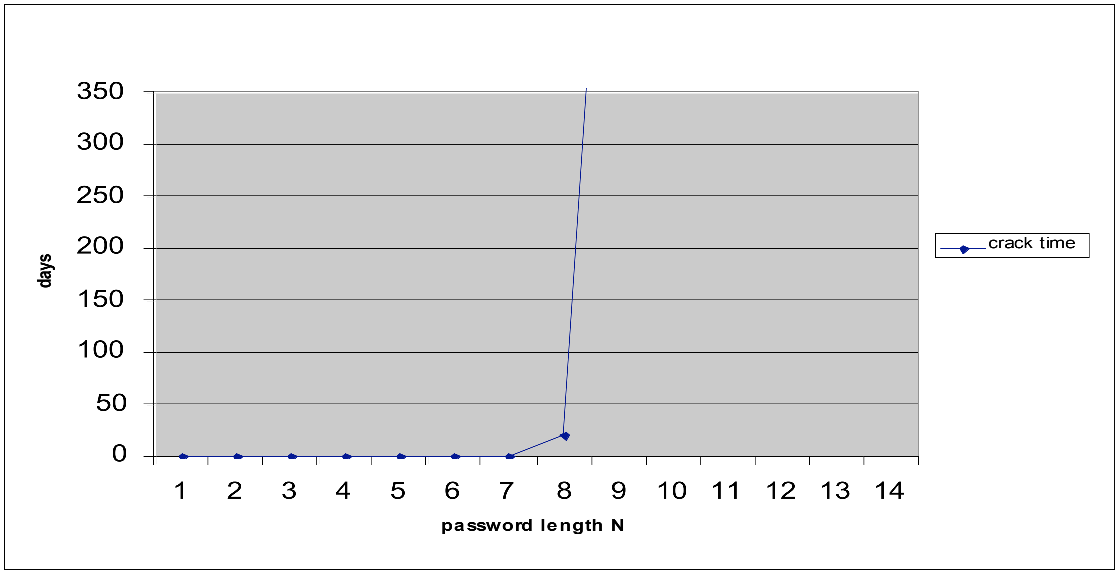
Design	
Name:	Oracle password hashing
Applications:	Oracle Databases version 8, 8i, 9i, 10g
Platforms:	All platforms supported by Oracle (HP-UX, MS Windows, SUN Solaris, …)
Used for:	Hashing passwords of database users
Algorithms used:	DES in CBC mode
Algorithm input:	Username, password
Salting:	Account specific (username is salt, password hash for <user> and <pass> is the same for all systems

Implementation	
Character set:	ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789#$_ (39 characters)
Limitations for character set:	Username and password MUST start with a alpha character [A-Z] Length of both username and password is > 0 and <= 30
Hash storage	Oracle table DBA_USERS, column PASSWORD, 8 byte (= 64 bits) field
Theoretical key space:	(Limitation · character set ^ length) $3.61 \cdot 10^{47} = 26/39 \cdot 39^{30}$
Key space limitations:	Theoretically, a hash collision will occur after 2 ^ (storage bits) passwords as documented in "Hash storage" $18{,}446{,}744{,}073{,}709{,}551{,}616 = 2^{64}$



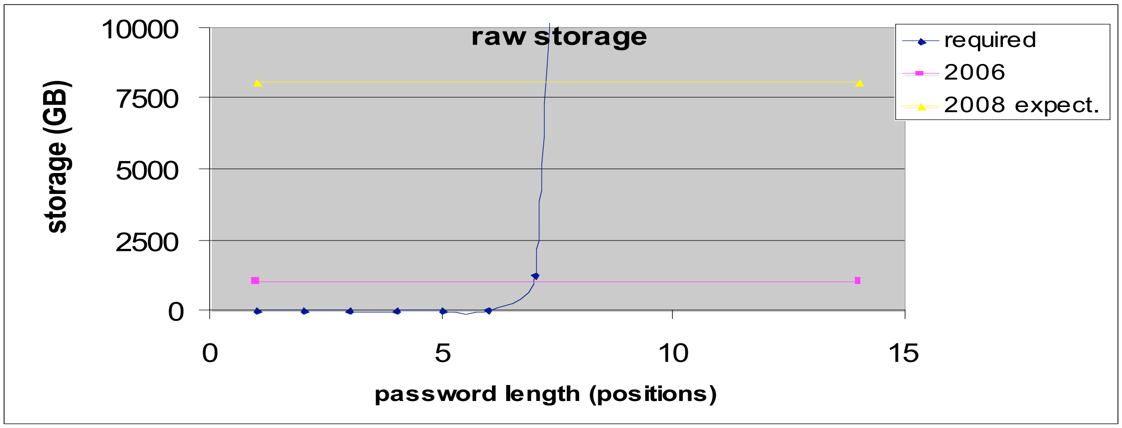
Key space for Oracle

The graph above shows the key space based on a 39-long character set.



Attack time for Oracle

The graph above shows the attack time based on a 95-long character set.



Storage for Oracle

The graph above shows the raw disk storage for storing both hashes and passwords based on a 95-long character set.

Microsoft SQL Server 2000

Microsoft SQL Server 2000 is a relational database management system and supports Microsoft's version of Structured Query Language (SQL). It is widely used by companies ranging from small- to medium-sized databases and in the past five years some large enterprise databases.

MS SQL Server 2000 uses the SHA1 algorithm for hashing passwords of database users. The input the first time for the algorithm is a salt and the password in UNICODE. The second time the input is a salt and the uppercase version of the password in UNICODE. The salt is generated from two calls to rand() (one call for the original password and one call for the uppercase password) which result in 8 numbers. In UNICODE there are 65535 characters are possible. The maximum password length is 128 characters.

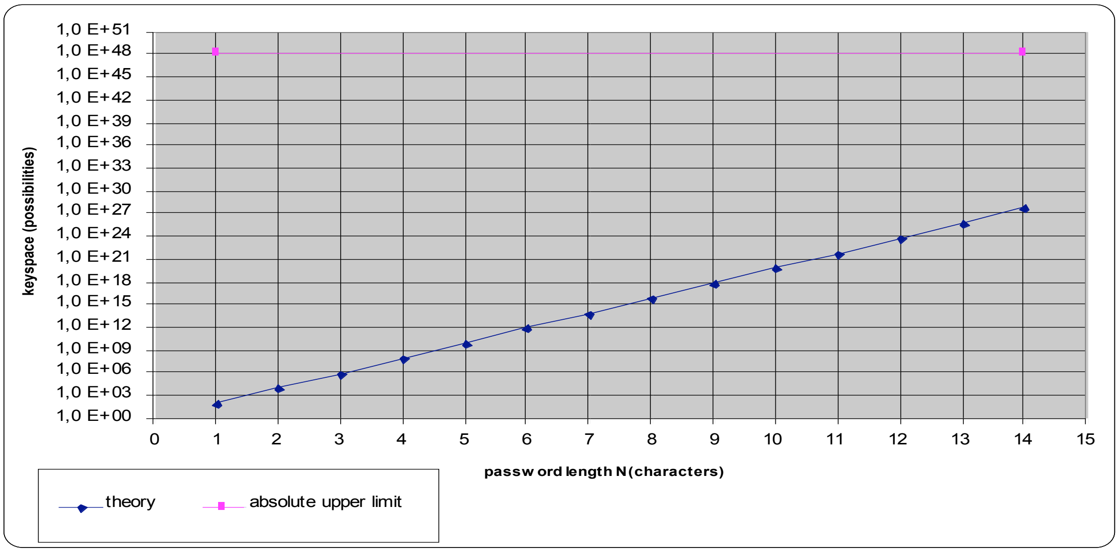
Free password auditing utilities like *NGSSQLCrack* or *SQLdict* are available to verify the complexity of SQL passwords.

An interesting tool is SQLPAT http://www.cqure.net/wp/?page_id=16. This tool should be used to audit the strength of passwords offline. The performance in brute force or dictionary attack mode on a 1 GHz Pentium with 256MB RAM is 750.000 guesses / sec. To perform an audit one needs the password hashes from the *sysxlogins* table, column *password*. The hashes are easy to retrieve although you need a privileged account like *sa* to do so.

Summary

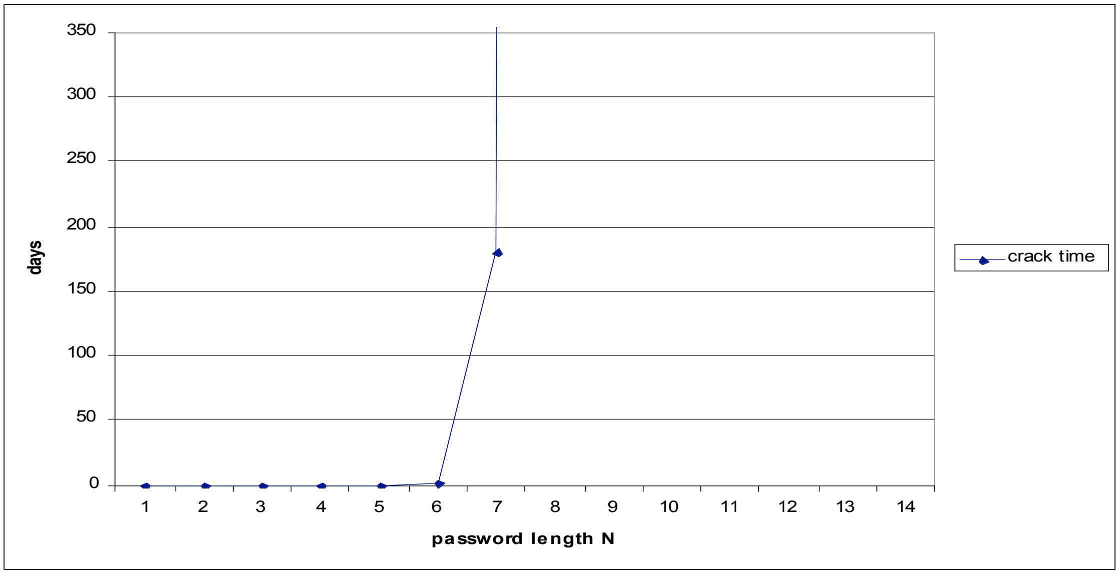
Design	
Name:	MS Sql 2000 password hashing.
Applications:	MS Sql 2000.
Platforms:	Windows.
Used for:	Hashing passwords of database users.
Algorithms used:	SHA1.
Algorithm input:	First round (mixed case): salt + password in UNICODE. Second round (upper case): salt + uppercase(password in UNICODE)
Salting:	Salt from two calls to rand() which result in 8 bytes. One salt (4 bytes) for the original password and one salt (4 bytes) for the uppercase password.

Implementation	
Character set:	65,535 characters.
Limitations for character set:	Length of password is >= 0 and <= 128 Password is uppercased.
Hash storage	Location: "select * from master.dbo.sysxlogins;", column "password". Location: HKLM\security\policy\secrets\SQLSERVERAGENT_HostPassword\currval (accessible by Windows LocalSystem account, Administrators can take ownership and give themselves permissions to these keys) Size: 20 bytes Format: 0x0100F612916E596524EC954399F27089FA416 046C6DA07D04B8845E41ED1A655CD5F6E23F86E573B A550BA17D21C <tag(2)><salt(4)>< sha1 mixed case password hash (20)>< sha1 upper case password hash (20)>
Theoretical key space:	$K = C^N$ $65{,}535^{128} = 3.23 \cdot 10^{616}$
Key space limitations:	Theoretically, a hash collision will occur after 2 ^ (storage bits) passwords as documented in "Hash storage" $1.46 \cdot 10^{48} = 2^{160}$



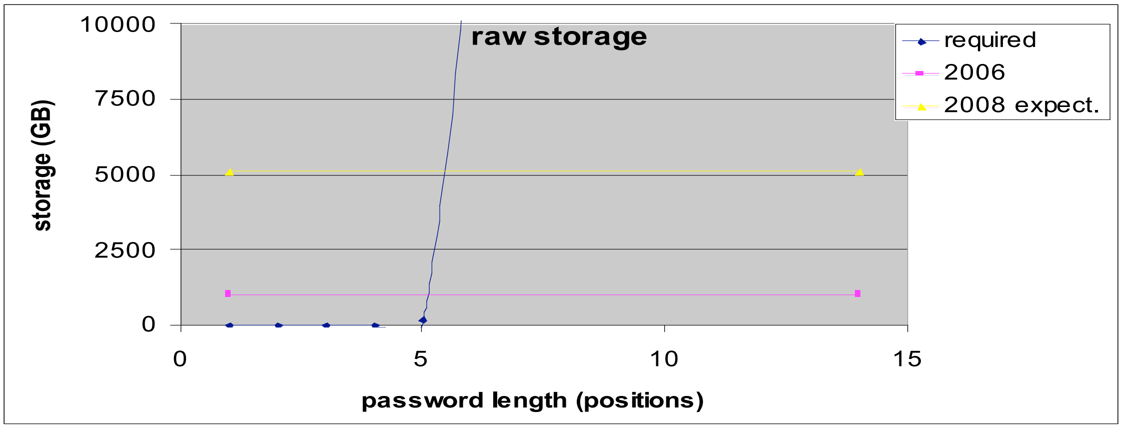
Key space for Microsoft SQL Server 2000 & 2005

The graph above shows the key space based on a 95-long character set.



Attack time for Microsoft SQL Server 2000 & 2005

The graph above shows the attack time based on a 95-long character set.



Storage for Microsoft SQL Server 2000 & 2005

The graph above shows the raw disk storage for storing both hashes and passwords based on a 95-long character set.

Microsoft SQL Server 2005

Five years after the introduction of Microsoft SQL Server 2000 Microsoft introduces Microsoft SQL Server 2005. This new product Microsoft is relatively secure by default, there are several security flaws removed from MS SQL Server 2000. These included the removal of the uppercase encrypted password and no more *sa* blank passwords.

The ability to manage SQL Account passwords is new. Now it is possible to enforce password complexity, password expiration and account lockout. Another addition is the ability to encrypt and decrypt data with certificates. No third party tools are required. The encryption algorithm for passwords didn't changed and is still SHA1. Also the maximum password length is still 128 characters.

Summary

Design	
Name:	MS Sql 2005 password hashing.
Applications:	MS Sql 2005.
Platforms:	Windows.
Used for:	Hashing passwords of database users.
Algorithms used:	SHA1.
Algorithm input:	Salt + mixed case password in UNICODE.
Salting:	Salt from call to rand() which result in 4 bytes.

Implementation	
Character set:	65535 characters.
Limitations for character set:	Length of password is >= 0 and <= 128
Hash storage	Location: "select * from sys.sql_logins;", column "password_hash". Size: 20 bytes Format: 0x0100B42B151AEE6F82D0317AB58C9B5CF5BAC39932E2B82360FE <tag(2)><salt(4)><sha1 mixed case password hash(20)>
Theoretical key space:	$K = C^N$ $65,535^{128} = 3.23 \cdot 10^{616}$
Key space limitations:	Theoretically, a hash collision will occur after 2 ^ (storage bits) passwords as documented in "Hash storage" $1.46 \cdot 10^{48} = 2^{160}$

Conclusion

Real life

In real life the limiting issues in the security of systems are mostly implementation oriented. As we take a look at the tables for key spaces we can conclude that the real maximum is crossed below the length of a password of 14 characters for NTLMv1, NTLMv2, Oracle, Microsoft SQL Server 2000 and Microsoft SQL Server 2005.

When we look at the crack times, we see that they vary a lot. LM hashes can be cracked easily, while NTLMv2 and Microsoft SQL Server 2005 take relatively much longer.

Although we didn't had the time to incorporate al the time-space trade-off systems for all algorithms, we still can conclude that the storage amounts for LM hashes, Oracle, Solaris and AIX are relatively low in contrast to NTLMv2 and Microsoft SQL Server 2000 & 2005. The more storage it takes how longer it takes to prepare a rainbow table attack.

Limiting the possible success of threats can be established in many ways depending on the implementations. A number of ways are possible to defend a system and or network against threats.

Combination of policy and technology

The combination of policy and technology can extensively increase the security of ones system and or network. When defining password policies for deceasing the crackability of passwords, consider these points :

- Require a minimum password length
- Require pass phrases
- An expiration date
- Force the use of special characters (complex passwords)
- Account lockout (where appropriate)
-

Disabling technologies like LM hashes (see KB 299656 and KB 828861) or defining strong encryption algorithms (i.e. blowfish) with salting for user passwords on UNIX would increase a systems security. It is advisable to analyse and test this before applying it widely. Some applications might depend on these rather old technologies.

Pass Phrases vs. passwords

Pass phrases are long strings, for example: "http://www.google.com is my favourite search engine". Pass phrases come along with several advantages:
- Very strong protection against attacks
- Easy to remember, a bit longer to type
- Sometimes break older applications

Passwords are short complex strings, for example: "P4S$w0Rd". Passwords come along with several disadvantages:
- Hard to remember
- Often difficult to type
- Not resistant against current attacks (Obvious substitutions are quickly broken)

To summarize: Long easily-remembered pass phrases are better than short complex passwords. This is still of course depending on the implementation and the used encryption algorithms.

The final conclusion

How secure a password is, is depending on a large key space, the used algorithm and the size of the stored hashes. A large key space is mostly realised using a long-length password. A huge character set simply isn't feasible because users refuse to use it and the keyboard doesn't allow easily more then 95 different characters.

Future research

Unfortunately, due to the lack of time caused by improper planning, there was no time left to research and incorporate WPA-PSK, HP-UX, LDAP, PAM, MS Cache, other Linux distributions and many other implementations in this document. Including these in this document would give a better view of password security on different operating systems and database management systems.

Time-space tradeoffs based on Philippe Oechslin principle were not used as input for the diagrams also due to the lack of time. This would give an even more realistic view for password attacks. Time-space trade-off mechanisms are relatively dangerous for the future of password security. When applicable, we however mentioned this risk at each implementation.

All items which are mentioned above rest for the future research.

Appendix A: General information

Widely used terms in cryptography

There are a lot of terms used in cryptography. The most common terms are discussed briefly in the following section. It will be out of the scope of this article to discuss every term completely. This section forms a base for understanding this document. The terms are listed randomly; however we tried to sort them from basic to in-depth information. A more extensive description or explanation of the topics can be most of the time found by entering the topic in, for example, the Wikipedia (on: http://en.wikipedia.org/wiki/).

Cryptography

Cryptography is the science of writing in code. Cryptography is an ancient art and science, the first documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe used non-standard hieroglyphs in an inscription. A newer form of cryptography arises after the widespread development of computer communications. When communicating over an untrusted medium, like the internet or any other network, the use of cryptography is imperative. More information about this subject, including an interesting description of the history, can be found on the Wikipedia[12].

Encryption

This is the process of scrambling information to make a message unreadable for those who do not have the decoding key. Encryption has been used to protect communications for centuries and can be used to ensure secrecy. Techniques are needed for secure communications to verify the integrity and authenticity of a message.

Cryptanalysis[13]

This is the study of a cryptographic system with the purpose of finding weaknesses. In practice this means breaking the code which is used to encrypt the message without knowing the secret key.

Cipher

A cipher is an algorithm for performing encryption.

Plaintext

Plaintext is the original information which is the input of an encryption algorithm.

Ciphertext

Ciphertext is the encrypted form of a plaintext, which is the output of an encryption algorithm.

Block cipher

A symmetric key cipher operates on a fixed-length group of bits (blocks). A block cipher takes for example a 64 bits block of plaintext and a secret key as input, and generates a 64 bits block ciphertext as output

Stream cipher

A stream cipher, also known as a state cipher, is a symmetric cipher in which plaintext digits are encrypted one by one and in which the transformation of digits varies during the encryption.

Symmetric key

Symmetric-key is also known as private-key or single-key. A symmetric-key algorithm uses trivially related cryptographic keys for both encryption and decryption.

Asymmetric key

This form of cryptography allows parties to communicate securely without having prior access to a shared secret key. A pair of keys, known as a public key and a private key is used for communication. The public key is distributed widely while the private key is kept secret. The sender uses the public key from the receiver to encrypt the message. The receiver uses his private key (secret) to decrypt the message.

Key schedule

A key schedule is the algorithm for computing the sub keys for each round in a product cipher from the encryption (or decryption).

S-box

Short for Substitution box is a basic component of symmetric key algorithms. An S-box is used to obscure the relationship between the ciphertext and plaintext. The input is a number of bits which are then transformed into a number of output bits. Fixed tables are normally used as lookup tables.

Hexadecimal

Hexadecimal, also known as base-16 or simply hex, is a numeral system with 16 written symbols using also the symbols A-F in addition to the usual symbols 0-9.

Base64

Base 64 is a positional numbering system using a base of 64. This is the largest power of two base that can be represented using only printable ASCII characters. All well-known variants use the characters A-Z, a-z and 0-9 in that order for the first 62 digits but the symbols chosen for the last two digits vary.

Exclusive OR

Exclusive OR (XOR) is a logic operator which results in true if one of the operands, but not both of them, is true.

Hash function

A hash function examines input data and produces a hash value as output. The input has a limited variable length in contrast to the output length which fixed. The input is a user password combined with an optionally a salt, depending on the used encryption algorithm. It is unlikely that two different inputs hash to the same output. One-way hashes are named one-way because they cannot be reversed.

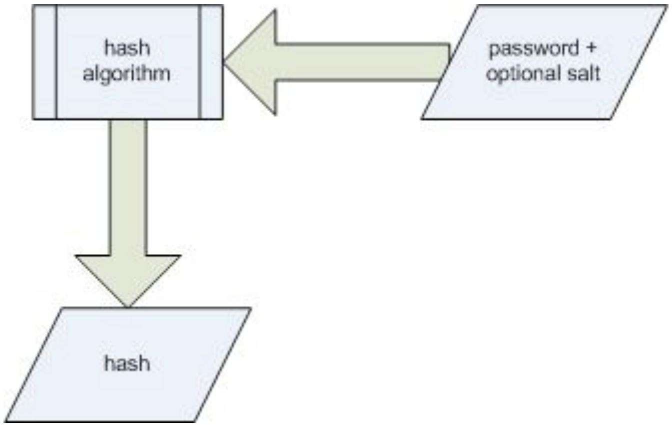


Figure 2 - A hash function

Salting

The use of salts in cryptography is to make a key derivation function produce random output. Salts mostly consist of random bits, which are used as an input to the cryptographic algorithm, next to the password.

Salting was first introduced early Unix-systems. Users, who had access to the passwd-files, could view the usernames and hash-values. Because different usernames had the same hashes, they could conclude this user had the same password. This was because the input on the algorithm was the same, so the result was also the same. This is why a 12-bit salt was added to the input, which resulted in randomized output. In modern implementations bigger salts are used.

Most hash functions use a salt value, because salting makes it more difficult to compare hashes generated with the same password. The salt value may or may not be protected as a secret. Most of the time, the salt value is stored with the hash value. Adding a salt makes it more difficult to conduct a dictionary attack using dictionary or pre-encrypted entries, as each bit of salt doubles the amount of storage and computation required.

Key spaces

The key space of a cryptographic algorithm refers to all possible keys that can be used to initialize it. The use of a large key space makes it computationally infeasible to check each possible key by brute force.

For example, a 20-bit key would have a key space of 1,048,576 (2^{20}). The Rijndael algorithm allows a key of up to 256 bits, which is over 1,15792 x 10^{77} (2^{256}).

Cracking a hash value

Password cracking is performing an attack to retrieve a users password. There is a difference between normal password cracking, like just trying passwords on an online password field and get a go or no-go, or performing an offline attack by cracking the hash value to get the text used as the input of the hash.

This document describes the last one, performing offline attacks. This is not actual password cracking, but an attack on the cryptographic hash value. The purpose is to find an input value that generates the same hash as the hash value of the password we already have. If a value is found that generates the same hash as the one we are comparing with, than we have found a value that can be used as correct password to login.

There is a possibility that this value is not the correct original password but just some nonsense input. If this is the case, we have found a hash collision. The change that this will occur is highly unlikely and dependent on the algorithm used.

Cracking speeds

Cracking speed indicates how many times per second an input value is used to create a hash value and compared against the original hash. The cracking speed of an attack, often measured in crypts per second, primarily depends on the algorithm used to perform the crypts and the speed of the hardware used performs the crypts.

Ciphers in cryptography

Ciphers are used very often in popular software these days to obscure information. These ciphers vary from weak to strong algorithms. In this section we will briefly discuss often-used ciphers in popular software. This section forms the base for understanding the terms which are used in the chapters.

DES

The Data Encryption Standard (DES) is a method for encrypting information. This symmetric-key encryption method was developed in 1975 and standardized by ANSI[14] in 1981 (ANSI X.3.92). DES is a block cipher method which breaks text into 64-bit blocks and encrypts them. A 56 bit key is used to customize the transformation.

3DES

3DES, also referred to as Triple DES, is a block cipher formed from the Data Encryption Standard by using it 3 times. 3DES with three different 56 bit DES keys has a key length of 168 bits. Together with the parity bits it takes up a total of 192 bits. In general the first encryption is encrypted with the second key, and the resulting ciphertext is encrypted with a third key.

Advanced Encryption Standard (AES)

AES, also known as Rijndael is a block cipher and was adopted by National Institute of Standards and Technology (NIST)[15]. AES uses a block size of 128 bits, key sizes of 128, 192 or 256 bits and 10, 12 or 14 rounds (for the respective key sizes). AES operates on a 4x4 array of bytes; the *state*. For encryption each round consists of 4 stages;

1. SubBytes; a non-linear substitution where each byte is replaced with another byte according to a table lookup.
2. ShiftRows; a transposition step, each row of the *state* is shifted cyclically a certain number of steps.
3. MixColumns; mixing on the columns of the *state*, combining the four bytes in each column using a linear transformation.
4. AddRoundKey; each byte of the *state* is combined with the round key; each round key is derived from the cipher key using a key schedule.

Feistel cipher

The Feistel cipher is a block cipher with very similar encryption and decryption operations, requiring a reversal of the key schedule. The plaintext is split into two pieces. The round function *f* is applied to one half using a sub key and the output of *f* is exclusive-ored with the other half. Then the two pieces are swapped, each round follows the same transformation except for the last round where there is no swap.

Blowfish

Blowfish is a keyed symmetric block cipher designed by Bruce Schneier in 1993 as an alternative to existing encryption algorithms, such as DES. Blocks have a size of 64 bits and the key sizes are between 32 and 448 bits in steps of 8 bits. Blowfish is a 16-round Feistel cipher which uses large key-dependent S-boxes.

Twofish

Twofish is a symmetric key block cipher with a block size of 128 bits. Key sizes are up to 256 bits. Twofish is related to the Blowfish cipher but has some distinctive features such as a pre-computed key-dependent S-box and a complex key schedule.

MD2

Message Digest 2 is a hash function developed by Ronald Rivest of MIT[16]. A 128 bit hash value is formed from any message by padding it to a multiple block length and adding a 16 byte checksum to it. The hash represents typical 32 digit hexadecimal numbers.

MD4

Message Digest 4 is a hash function also developed by Ronald Rivest. The 128 bit (16 byte) hashes are typically representing 32 digit hexadecimal numbers.

MD5

Message Digest 5 is a widely used hash function with a 128 bit hash value. The Internet standard (RFC 1321[17]) MD5 was designed by Ronald Rivest. It has been employed in a variety of security applications and is also used to check the integrity of files. MD5 processes a variable length message into a fixed-length output of 128 bits.

SHA-1

Secure Hash Algorithm 1 is a set of related hash functions which is implemented in a variety of popular security applications and protocols. The SHA algorithms were designed by the National Security Agency (NSA)[18] and published as a US government standard. SHA-1 produces a 160 bit digest from a message with a maximum size of 2^{64} bits and is based on the principles similar in the design of MD4 and MD5. The algorithm is slower but creates a larger message digest, which makes it more challenging to brute force attacks.

Appendix B - Creative Commons Licence

Attribution 2.5 1

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED. BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions
   a. "Collective Work" means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.
   b. "Derivative Work" means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.
   c. "Licensor" means the individual or entity that offers the Work under the terms of this License.
   d. "Original Author" means the individual or entity who created the Work.
   e. "Work" means the copyrightable work of authorship offered under the terms of this License.
   f. "You" means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

2. Fair Use Rights.
   Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.
3. License Grant.
   Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

   a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;
   b. to create and reproduce Derivative Works;
   c. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works;
   d. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audit transmission Derivative Works.
   e. For the avoidance of doubt, where the work is a musical composition:
      i. Performance Royalties Under Blanket Licenses. Licensor waives the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work.
      ii. Mechanical Rights and Statutory Royalties. Licensor waives the exclusive right to collect, whether individually

or via a music rights agency or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions).

f. Webcasting Rights and Statutory Royalties. For the avoidance of doubt, where the Work is a sound recording, Licensor waives the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions). The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. All rights not expressly granted by Licensor are hereby reserved.

4. Restrictions.
The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by clause 4(b), as requested. If You create a Derivative Work, upon notice from any Licensor You must, to the extent practicable, remove from the Derivative Work any credit as required by clause 4(b), as requested.

b. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or any Derivative Works or Collective Works, You must keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and in the case of a Derivative Work, a credit identifying the use of the Work in the Derivative Work (e.g., "French translation of the Work by Original Author," or "Screenplay
based on original Work by Original Author"). Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Derivative Work or Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer
UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability.
EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Derivative Works or Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You distribute or publicly digitally perform the Work or a Collective Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.

b. Each time You distribute or publicly digitally perform a Derivative Work, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.

c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.

e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

References

[1] A cryptanalytic time-memory trade off – IEEE Transactions of Information Theory, IT-26:401-406, 1980

[2] Cryptography and Data Security, page 100. Addison-Wesley, 1982

[3] Windows NT rantings from L0pht - http://www.packetstormsecurity.org/Crackers/NT/l0phtcrack/l0phtcrack.rant.nt.passwd.txt

[4] http://www.ibm.com/us/

[5] http://www.eff.org/

[6] http://www.eff.org/

[7] http://www.sans.org/rr/special/index.php?id=oracle_pass

[8] Orabf – A brute force/dictionary tool for Oracle - http://www.toolcrypt.org/index.html?orabf

[9] Oracle paper – Oracle Database Hardening - http://www.oracle.com/technology/deploy/security/pdf/twp_security_checklist_db_database.pdf.

[10] SecurityFocus - http://www.securityfocus.com/infocus/1689

[11] Website of Pete Finigan - http://www.PeteFinnigan.com

[12] Wikipedia, the free encyclopedia – http://en.wikipedia.org/wiki

[13] http://en.wikipedia.org/wiki/Cryptanalysis

[14] http://www.ansi.org/

[15] http://www.nist.gov/

[16] http://www.mit.edu/

[17] http://www.ietf.org/rfc/rfc1321.txt

[18] http://www.nsa.gov/