Password Cracking Techniques
Think your passwords are secure? Think again
www.hackthepc.com
www.hackpconline.com

Password Cracking Techniques

Think your passwords are secure? Think again

Legal Information

This book and the software fragments included present techniques thanks to which both IT environment can be protected by its user as well as other systems can be attacked.

That's why we would like to draw your attention to the fact that this book and software included can be used only to protect your IT environment. Conducting an attack on other IT system without the permission of its respective owner is penalized by the federal **Computer Fraud and Abuse Act** (if you live outside the United States, please refer to your local law).

While being an administrator and using the tools decried in the book in order to protect your own IT environment, or getting to know the hacking techniques presented, you always have to be sure that your actions are fully legal.

Table of Contents

Introduction

Password Cracking! Yah! I hope you are as excited about the subject as I am. If not, you must be at least somewhat interested since you are reading this right now, unless of course you pick up random things and read them because you have nothing better to do. In that case, I'll get you interested!

Whether you are a computer security hobbyist, Certified Ethical Hacker, or just want to know how to crack your girlfriend's password because she's "cheating" on you, you will enjoy and get something out of this book. It covers the very basics of password hacking and cracking.

If you are really serious about learning all about hacking, then not only will you read everything in this book, but you will also DO. DO every example I show you because no matter how much you read, you will never really know something until you DO it. So DO it Damnit!

Password Cracking Techniques

Passwords

A **password**, also known as a PIN, passcode or secret code, in its simplest form, is just a secret word or phrase used for authentication, to determine whether you are who you say you are. Nowadays when you hear the word password, you automatically assume they are talking about a website or something related to computers and other electronic devices, but computers haven't always been around, passwords have.

Where did Passwords originate?

Passwords have been around since before you or I can remember. In ancient times, as recorded in literature, they were used by sentries to challenge people who wished to enter their territory. If the approaching people knew the "watchword", then they were allowed to pass. If not, then they were shot in the throat with a couple arrows, left to die, and then brought in for food. Actually, I don't know about the last part.

After some research I have yet to figure out where passwords originally originated from, but I would assume it's safe to say they have been around since cave men time. Keeping cavemen out of the caves they don't belong in.

How are Passwords used?

Passwords are a big part of our daily life. We have passwords to protect our email, voicemail, phones, ATM cards, lockers, online banking, wireless networks, encrypted data and I can go on forever! Well maybe not forever, but you get the point.

As you can see, we need to keep them secure because if they fall into the wrong hands bad things will happen. Let's take a look at some possible outcomes that could happen if such a thing happened.

- Your bank account could be emptied to fund a large purchase of socks for Judith's sock fetish.
- Your sensitive email could be read by the stalker next door.
- You could be visited by the FBI after your wireless network was used to surf child porn.
- Your gym locker could be emptied, except for your pubic lice shampoo that was left out for everyone to see.
- Your love for your friend's grandmother could be exposed after your Facebook messages were read and made public.

As you can see, none of the above situations are delightful to experience. This is why it is extremely important to know how to secure and create strong passwords.

After reading the above, you are probably sweating profusely, trembling in your chair and thinking to yourself, "Oh no! I don't know how to secure and create strong passwords!?" but don't worry! As you continue reading this book you'll find out.

Password Cracking

Password cracking is the act of recovering passwords through unconventional and usually unethical methods from data that has been stored or sent through a computer system.

Password cracking is a very popular computer attack because once a high level user password is cracked, you've got the power! There's no longer a need to search for vulnerabilities and all that other mumbo jump needed to take over a system that we won't be discussing in this section.

Also, everyone is susceptible to a password cracking attack. Unless you live in a remote, technology absent area, you have a password for something, and there's usually something to gain from obtaining your password.

To show you how real and popular this form of attack is today, here are a few recent happenings.

- Password cracking was used to take over a few high-profile twitter accounts, including President Barack Obama, Britney Spears, Kevin Rose, and Rick Sanchez.
- Wal-Mart was a victim of a security breach where sensitive information was taken. Password cracking was one of the many methods used to gain entry.
- 10,000 cracked Hotmail passwords were publicly posted, and every day crackers continue to post new lists on forums all over the internet.

- phpBB.com was hacked and their 200,000+ username/password database was dumped and made publicly available to anyone willing to download it. Of those passwords, over 80,000 were reported to have had been cracked.

What is Password Cracking used for?

Password cracking can be used for both good and evil. If I forgot my password for a certain system or program, I might try cracking it before I completely give up on it. Now if it's for any other reason, then it probably has an evil basis and is most likely illegal as well.

Notice how for my legitimate reasons I didn't mention cracking services. Services are usually things like your ISP (Internet Service Provider), email, social networking and other related passwords. The reason why I didn't mention these is because even if I legitimately forgot my password for a site like Facebook or Yahoo, it is still against their TOS to attempt to crack those passwords. Why? Because you will be attempting 100's of password/second over the internet which could put a strain on their system and cause a DOS (Denial of Service) attack. Also, if not done properly, most systems would detect it as an attack and lock you out, sometimes even blocking your IP address completely so that you have absolutely no access to the website from your current ISP given IP address. Even though it is possible to change your IP address, you don't want to keep doing that. No matter what your reasons are for attempting to crack a password from a service site, it will always be seen as a malicious attack because the websites provide methods for the owner to retrieve their forgotten password. With that said, cracking service site passwords is still very possible and in some cases very easy.

Password Cracking Methods

There are many different types of password cracking methods, and I will introduce you to each one of them. Below is a list of the methods you will soon become a pro in:

1. **Dictionary Attacks**
2. **Brute Force Attacks**
3. **Hybrid Attacks**
4. **Rainbow Tables**

Password Hash

Passwords are most often stored in their plaintext format or in their **hashed value format** in a file system or in a database. If your password was "password" and it was stored as just "password" this would be an example of your password stored in its plaintext form. So if you could extract the password list from your victim and the passwords were stored in their plaintext form, then you have no need to crack anything because you already know the passwords. Da tu du! But if you extracted the list of passwords or dumped the database of passwords, and they were stored in their hashed values, then it's crackin' time! But before we go any further, let's look at the basics.

What is a password hash?

A password hash is the password after it has gone through a one-way mathematical process, or algorithm, producing a completely different string. So let's say your password is "**password**" and you run it through the MD5 algorithm, one of the many cryptographic hash functions out there, your final outcome will be **5f4dcc3b5aa765d61d8327deb882cf99**. There is now no possible way of changing that back to the word "password". The only way to reproduce that key combination is to either know the word and run it through the same hash function, or by trying to crack it, which is essentially the same thing.

The Login Process

Before you even go to login to one of your many password/username protected websites, you must first create your login details. So what happens when you create your login details and hit submit? It's pretty simple. Most websites run your password through a cryptographic hash function like the one mentioned above and then store it in a database. Here is an example of how a PHP script would hash your password before it is stores it in a database.

```
$Password = MD5($_POST['password']);
```

In the above PHP line, the script takes the password you submitted via $_POST and runs it through the MD5() cryptographic hash function, which transforms the submitted password into its MD5 hash value. Then the hash is stored in the variable $Password, which is later stored in the database.

Now that you have your login details created, next time you go to login, the PHP script will take the password you submitted, run it through the hash function, and compare it to the hash stored in the database. If the two hashes match, it means that the password submitted is the same password stored in the user database, so the website will log you in. Here's an example in pseudo-code.

```
If (md5($Submitted_Password) == $Stored_Password_Hash) Then
Login()

Else
```

Display_Wrong_Login_Details_Message()

What is a password salt?

No, it’s not the type of salt that stings your eyes when you open them in the ocean because you thought you saw some sort of sea creature next to your legs and then find out it’s just a shell until you get your head out of the water and that “shell” starts chomping on your big toe causing you to scream like a three year old girl and splash around like a dying fish on the shore. True story. Password salts are completely different, even if they have the same affect on password crackers.

A password salt is a string that is added on to a user’s password before it is encrypted. This string could be anything, the user’s username, the exact time the user signed up, or something completely random.

The point of a password salt is to make a password more secure by making it much harder to crack. It does this by making the password longer, and by making each password hash different from every other, even if the password is the same.

For example, if the password was “**123456**”, the final hash would be MD5(“random-salt”+”123456), so even if someone else used that same password, their salt would be different, which would result in a different password hash. This way, if the attacker cracks a password, he wouldn’t be able to find every other user with the same password because their hashes would be different.

We'll get more into salts once you learn more about password cracking.

Online vs. Offline Password Cracking

When performing a password cracking attack, it is either an **online** or **offline** attack. Let's look at each method in detail.

Online Password Cracking

Online attacks are necessary when you don't have access to the **password hashes**.

When performing an online attack, you are usually presented with a web form asking for a username and password combination. There, you could try to guess the password, but that usually won't get you anywhere. Instead, you could create or use an available automatic password guessing tool. Luckily for those of you who can't program, there are already hundreds of these tools freely available online.

The downside of performing an online attack is that it can be very noisy, extremely slow and sometimes just not feasible.

Many login forms have a lockout feature that locks you out after a certain number of failed login attempts. For example, one of my cPanel hosting accounts will completely block my IP address if I fail to login after five attempts. When this happens, I am forced to contact customer support to have my IP address manually unblocked so that I could access the site. Another example is if I fail to login into my online banking after multiple tries, my account will be locked for 20 minutes.

If the target websites doesn't have a lockout feature, that doesn't mean you're golden. Online password cracking attacks are very noisy, and when you are throwing random wrong passwords at a system, its log file will grow tremendously. It looks very suspicious when there are hundreds of wrong password attempts logged to the same IP address.

To get around these factors, you might try to cover up your IP address via a proxy, use a different proxy for every 5 to 10 guesses, or even attempt a few guesses every 30 minutes so it looks less suspicious. Many of the password cracking programs out there have these features available.

Online attacks can be very slow because the speed of the attack depends on the speed of your internet connection and the speed of the target server. Because of this, the best and really the most effective type of attack is a dictionary related attack. So if you have a fairly secure password you will most likely not fall victim to an online password cracking attack.

Offline Password Cracking

Offline attacks are only possible when you have access to the password **hash(es)**. The attack is done on your own system or on systems that you have local access too.

Unlike an online attack, there are no locks or anything else to stop you on an offline attack because you are doing it on your own machines. The only thing that could hold you back is the limits of your computer hardware because an offline attack takes advantage of its machine's processing power and its speed is dependent on the speed of the actual machine. So the better the processor and nowadays even graphics card, the more password guessing attempts you can get per second.

Now that you know the difference between online and offline attacks, I'm sure you'll agree with me that you should try to use offline attacks whenever possible. This obviously won't be possible most of the time, so we will look at real world examples of both methods later.

Password Cracking

-Dictionary Attack-

A **dictionary attack** is password attack where every word from the dictionary is attempted against a password hash. Good dictionary attacks use wordlists with dictionaries of other languages (depending on the target), the most commonly used passwords (many of which aren't words in the dictionary), and order the wordlists with the most commonly used passwords on top to save cracking time.

For those of you who are visual learners, a dictionary attack is like approaching a woman or man using a pickup line from a list in you pocket, being shot down and kicked in the face, trying again, being shot down and smacked in the face, until finally one of the pickups on your list work and you have yourself a date.

When should you use a dictionary attack?

When performing a password cracking attack, dictionary attacks usually are, and should be the first attack type used. Why? Because most people create shitty passwords due to the "huge" effort it takes to remember and type in a bit longer and more complex password. Due to this laziness factor, dictionary attacks can usually crack a good percentage of the hashes they are run against. Dictionary attacks are also the first and many times the only type of attack used in online attacks. This is because, as you’ve learned before, online attacks can be very slow and noisy.

Password Cracking

-Brute Force Attack-

A **brute force attack** is a password attack where every possible combination in a range of characters is generated and used against the password hash.

For those visual learners, a brute force attack is presented pretty well with a Rubik's Cube. The brute force attack would be the act of your hand turning the cubes in every possible direction to create different combinations until finally the Rubik's Cube is solved and you have matching colors, the password.

When selecting a range of characters to use for a brute force attack, you have a few options. Below are the ones available in the popular cracking program, **Brutus**.

- **Numerical** – Use any numbers from 0-9
- **Lowercase Alpha** – The lowercase alphabet
- **Uppercase Alpha** – The uppercase alphabet
- **Mixed Alpha** – Both lowercase and uppercase
- **Alphanumeric** – The lowercase and uppercase alphabet plus digits 0-9
- **Full Keyspace** – Everything above including all the special characters on your keyboard.
- **Custom Range** – If you have an idea of the characters included in the password(s) you can create a custom list of characters to use.

Each range option yields a different amount of possible password combinations. Let's look at how many combinations there are for a password of 0-6 characters in length.

- **Numerical** – 1,111,110 Passwords
- **Lowercase Alpha** – 321,272,406 Passwords
- **Uppercase Alpha** – 321,272,406 Passwords (obviously the same because it's the same amount of characters)
- **Mixed Alpha** - 20,158,268,676 Passwords
- **Alphanumeric** – 57,731,386,986 Passwords
- **Full Keyspace** – 697,287,735,690 Passwords

For a six character password, we are hitting over 697 Billion combinations for the full keyspace! And by just adding one more character to the password making it 0-7, the number of combinations jumps to 65,545,047,154,955, that's over 65 Trillion! As you can see, it makes a big difference having an idea of what types of characters are being used in the password(s) and how long it is.

Download Brutus - Remote Password Cracker

Calculating Number of Combinations

You now have an idea of how the number of combinations can grow with the addition of a new character or an extra character in the password, but how are these numbers calculated? Simple.

If you are doing a Numerical attack on a 6 character long password, that means there are 10 possible different characters (0-9) that you can use. So the equation to calculate the number of different combinations is:

$$\textbf{\# of different possible characters}^{\textbf{password length}}$$

So the expression for our example would be:

$$\mathbf{10^6}$$

Which, once calculated, comes out to 1,000,000 combinations?

Wait! Didn't I state that there are 1,111,110 possible combinations for the same character set before? Yes, but it was for passwords that consisted of 0 to 6 characters long, not just 6. If you don't get what I mean, look at it this way. When I'm looking for all the possible combinations of a password that is of length 0 to 6, I need to account for the combinations of all the 6 character length combinations, 5 character combinations, 4 character combinations and so on. If you were doing it out, this is what it would look like:

$$\mathbf{10^6+10^5+10^4+10^3+10^1=1111110}$$

This would get pretty tedious if you had to do it manually for long numbers, so here's a simple **C script** I made for you. Put together and that does it out for you.

```
#include
#include

main(){

int n = 10; // number of possible characters
int a = 6; // length of the password
unsigned long long int x = 0; // this will hold the answer, its set to unsigned long long int so that the variable x can
// hold the largest possible number

while (a >= 1){ // keep going until a is 1

x+= pow(n,a); //take n to the power of a and add it to x
a--; //subtract 1 from a

} //do it again until a is lower than 1

printf("The number of possible combinations is %lld.\n",x); //finally print the answer

}
```

Backwards Brute Force Attack

A backwards Brute Force attack is a brute force attack against usernames. So instead of using the brute force attack to create and try a bunch of password combinations, you will be most likely be using one password and using the brute force attack to generate all possible usernames in a range of characters, trying that password(s) against it.

When should you use a Brute Force attack?

Only use a brute force attack when a **Dictionary** and all other options fail. A brute force attack takes a lot of resources and a lot of time to perform. Depending on how big the password is, the range of characters being used and the resources available, a brute force attack can take **years** to fully complete as you'll see later on.

Password Cracking

-Hybrid Attack-

A **hybrid attack** is a mixture of both a **dictionary** and **brute force** attack. That means that like a dictionary attack, you would provide a wordlist of passwords and a brute-force attack would be applied to each possible password in that list.

A hybrid attack is like the beginning of an MMORPG where you choose your character design. Your figure stays the same but you have the choice to change your clothes, hair and color until you have the look you want, a badass Schwarzeneggar or a medieval hooker.

On my first day as a freshman in high school, I was given a username and password for the school's computer network. Everyone's password was set to the first initial of their first name, their last name and birth date. So if my name was Bob Sagat and I was born on May 22, 2010, my password would be "**bsagat052210**". This wasn't a great way to distribute passwords, but we did have to change it after we first logged in. Can you see why a hybrid would be an effective attack in this case? What I could have easily down was get a list of every freshman student in the school and apply a brute force attack to the end of each name. The rule would look something like this:

(first initial of first name)(last name)([0-9] [0-9] [0-9] [0-9] [0-9] [0-9])

In this case, a hybrid attack would have enabled me to crack every single student's password within a few minutes.

When should you use a Hybrid attack?

Use a hybrid attack whenever you have an idea of how a password is formatted. For example, if you dump a database of password hashes from a website, and after trying a dictionary attack against it you are left with many uncracked passwords, then take a look at the password requirements for that website. Many websites require a password to be made a certain way. For example it may require a password to have at least two numbers and a special character. Knowing how people like to make things as easy as possible for themselves, you can safely guess that many people used exactly two numbers and one special character. Armed with this knowledge you can go back to your dictionary file and apply a brute force attack to it (making it a hybrid attack), trying the following combinations:

([0-9] | SC) ([0-9] | SC) ([0-9] | SC) (password)

or

(password) ([0-9] | SC) ([0-9] | SC) ([0-9] | SC)

*Where **SC** = Special Character (ex. !,@,#,$) and (| = or).*

Password Cracking

-Rainbow Tables-

Rainbow tables are lookup tables that contain almost every possible combination of passwords in a given character set. It is simply a completed brute force attack that you can reuse as many times as you wish without having to regenerate all those possible password combinations. This can reduce password cracking time by up to 99%! Of course, to generate a rainbow table it can take longer that it would take to crack one password, but afterwards you will be able to crack others in a few minutes compared to hours. Now that you have a basic idea of what rainbow tables are, let’s look at exactly how they work.

If you recall, I said that rainbow tables store most of the possible hashes in a given character set. This is true, but not in the way you might think. Depending on the character set and length of the password, a rainbow table can get extremely large. Below is an example of tables from the **RainbowCrack Project**:

table id	charset	plaintext length	success rate	table size
md5_numeric#1-12	numeric	1 to 12	99.9 %	8.75 GB
md5_loweralpha#1-9	loweralpha	1 to 9	99.9 %	28 GB
md5_loweralpha-numeric#1-9	loweralpha-numeric	1 to 9	96.8 %	40 GB
md5_ascii-32-95#1-7	ascii-32-95	1 to 7	99.9 %	64 GB
md5_mixalpha-numeric#1-8	mixalpha-numeric	1 to 8	96.8 %	80 GB

MD5 Rainbow Tables

If the rainbow stored every hash for every plaintext word in a given character set, it would use more memory then you have or can imagine. Instead, rainbow tables use a time-memory trade off technique, known as chains.

To understand these chains, you must first understand the “reduction function”. As you’ve learned, **hash** functions map plaintext words to hashes, well, the reduction function maps hashes to plaintexts. You might be thinking, didn’t you say that you couldn’t reverse a hash function and get the plaintext from it? True, the reduction function doesn’t inverse the hash function acquiring the original text. No, instead it gets another plaintext from the hash. For example, if we have a set of plaintexts that are made up of 7 numbers [0-9], using the md5() function a possible hash could be ***MD5(1234567)*** **-> fcea920f7412b5da7be0cf42b8c93759**. In this case the reduction function could be as simple as taking the first seven numbers from the hash to generate a new plaintext. So once the reduction function was applied to the hash, it would result in the new plaintext, “**9274125**”.

And that's what the reduction function does, generates a new plaintext from a given hash.

These chains that make up rainbow tables are made up many combinations of the one way hash function combined with the reduction function, making a chain. For example, if we were to continue the chain from the first example, it would look like this:

***Md5(1234567)* -> fcea920f7412b5da7be0cf42b8c93759 ->**

***Reduction(fcea920f7412b5da7be0cf42b8c93759)* -> 9274124 ->**

***Md5(9274124)* -> ed7db1cf7fc4fbd0169f00c37a0165ab ->**

***Reduction(ed7db1cf7fc4fbd0169f00c37a0165ab)* -> 7174016 ->**

The above chain would keep repeating until you or the rainbow table maker decides to stop it, usually after millions of hashes are created. Now here's how rainbow tables save memory. Instead of storing every one of these hashes, the rainbow table only stores the first plaintext and the last hash in that chain. Then millions of more chains are created, each representing millions of password hashes. If a table was made up of the above chain, then it would be made up of its first plain and final hash looking something like this: **1234567 -> ed7db1cf7fc4fbd0169f00c37a0165ab**.

Once enough tables have been generated (millions), you will want to actually use the final rainbow table to crack a password hash by checking to see if it is inside any of the generated chains. Here is the algorithm:

1. <loop>Check to see if the hash matches any of the final hashes. If so, break out of the loop because you have found the chain that contains its plaintext.
2. If the hash doesn't match any of the final hashes in the tables, use the reduction function on it to reduce it into another plaintext, and then hash the new plaintext.
3. Go back to step 1.</loop>

Now that you know which chain contains the plaintext, you can start with the original plaintext of that chain and start hashing and reducing it until you come to the password hash you are trying to crack and its secret plain text. There you have it! The quickest and simplest explanation of rainbow tables that you will ever find!

When should you use a rainbow table?

If there are rainbow tables available for the type of password hash(es) you have, then use it! Chances are it'll work. If you have to download the tables, then while that's going on, trying a **dictionary attack** won't hurt.

If the passwords are salted, then you probably will not want to use rainbow tables because you will need to create a new rainbow table for every password. Why? Well, if you recall, when you salt a password, you add a random or unique string to it and then run it through the hashing function. So if the password was "**password**" and the script attached the user's username to it, the final password would be **MD5("username"+"password")**. Since everyone has a different username, the word "password" will never look the same once hashed.

Salts

I promised that once we learned more about password cracking we would come back to **salts**.

If you remember, I mentioned that one of the ways salts made passwords harder to crack was by making them longer. As you've learned, every additional character exponentially increases the amount of possible character combinations. Once the hacker finds out what the salt is, this is no longer the case. The attacker can edit any dictionary or brute force password cracker code to add the salt to the current word before running it through the hash function. The attacker can now run normal password lists and brute force attacks as if the salt wasn't even there. This can also applied on a larger scale. If the attacker found out that the salt was the user's username, he could easily automate a password cracker by editing the code to attach the user's username to the password. So as you can see, it is important to create good salts and store them as securely as possible.

Random Salts vs. Unique Salts

So which is better, having random generated salts, or unique salts like your username or email address? It depends on how you store it. If it is in the same database as your username and password hash, then it doesn't matter if it's random or unique, because it's being stored either way. Once the hacker gets access to the database and dumps the username/password database, to figure out what the salt is, all he would need to try attaching every stored value (username, email, name, etc..) to a possible password until he cracked the password. He would then know what the salt was for every other user. The attacker could also just choose to try and crack the password hash as is, and if successful he would see the salt and password in plaintext. The attacker would then compare the plaintext with the database values under that user and see where it matches up, finding the salt. This would probably take much longer or wouldn't work depending on how long the salt and password combination is.

This would be a different situation if the salt was stored in a different server because if the attacker had access to one database, he might not have access to the other. In this case, using a random salt would make sense because the attacker would still be able to guess a unique salt like a username, but not a random hash stored elsewhere.

For even greater security, in addition to using a salt that is stored in the database, you could add to it in the actual source code of the register/login script. This way, the attacker would need to have access to both the database and the source code to be able to get the salt.

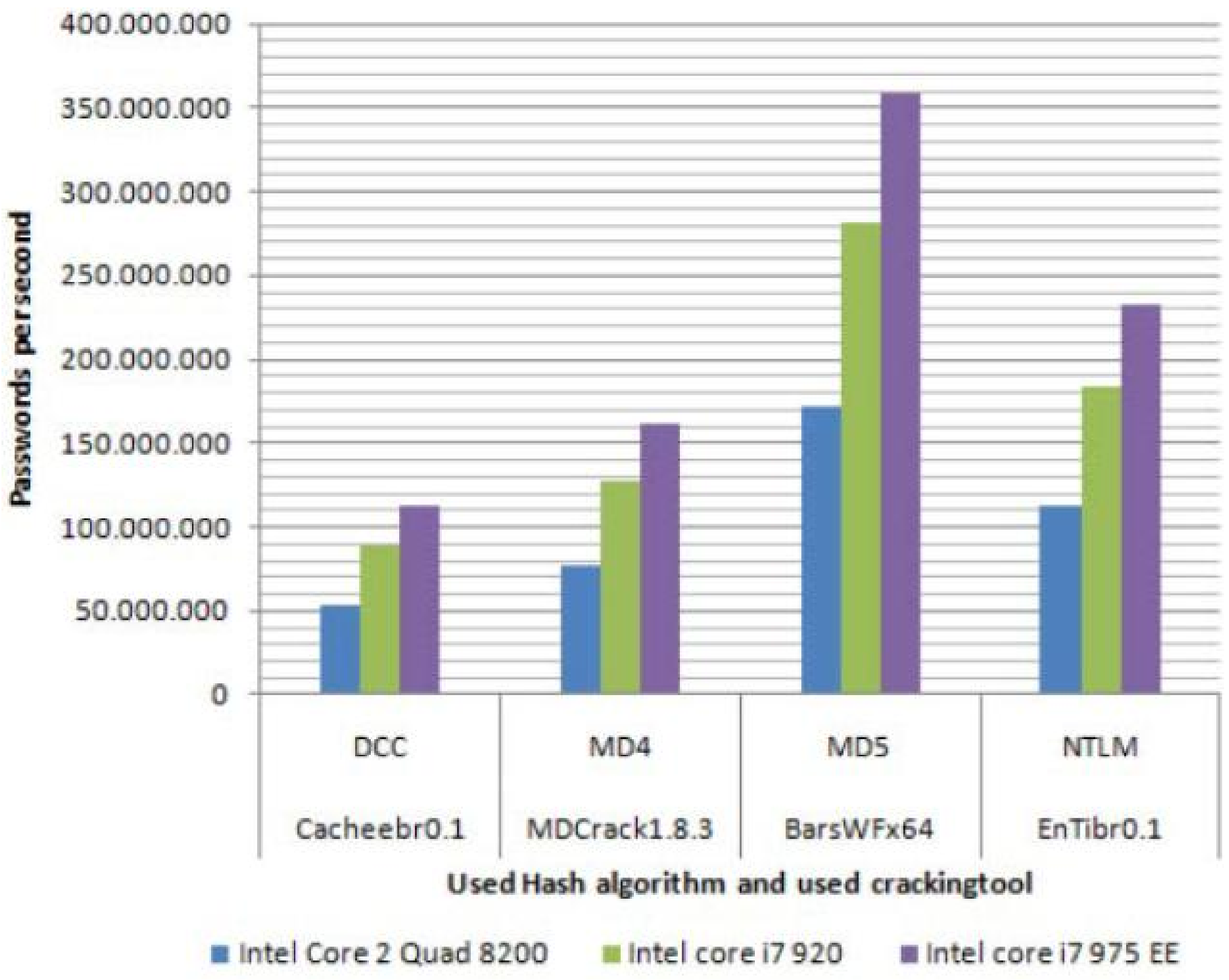
How To Crack Faster!

Password cracking can be a very time consuming process. The speed, or passwords per second, depends on the hardware that you are using. Most importantly your CPU (Central Processing Unit), GPU (Graphics Processing Unit, and the amount of machines you have dedicated to that one task a.k.a. computer cluster. Let's look at each one of these in greater detail.

CPU (Central Processing Unit)

The CPU is the brains of your computer. It is responsible for carrying out all of the instructions of a computer program, in our case, a password cracking program. This obviously means that the faster the CPU, the more passwords you can try per second and the faster you will crack the password.

CPUs today are very fast and thanks to Moore's law, the technology continues to improve and become faster. I'm sitting here typing away on an Acer netbook with a tiny 1.3 GHz Intel Atom processor that can brute force just over 1 Million passwords per second! Now, depending on the hash algorithm, the program being used and the processor you are using this can go up to the hundreds of millions of passwords per second. Below is an example.



Source: http://staff.science.uva.nl/~delaat/sne-2009-2010/p34/report.pdf

GPU (Graphics Processing Unit)

You might have been surprised when I mentioned a graphics card was important for cracking passwords. You might be even more surprised to hear that a graphics card can be much faster than a high end CPU at cracking passwords.

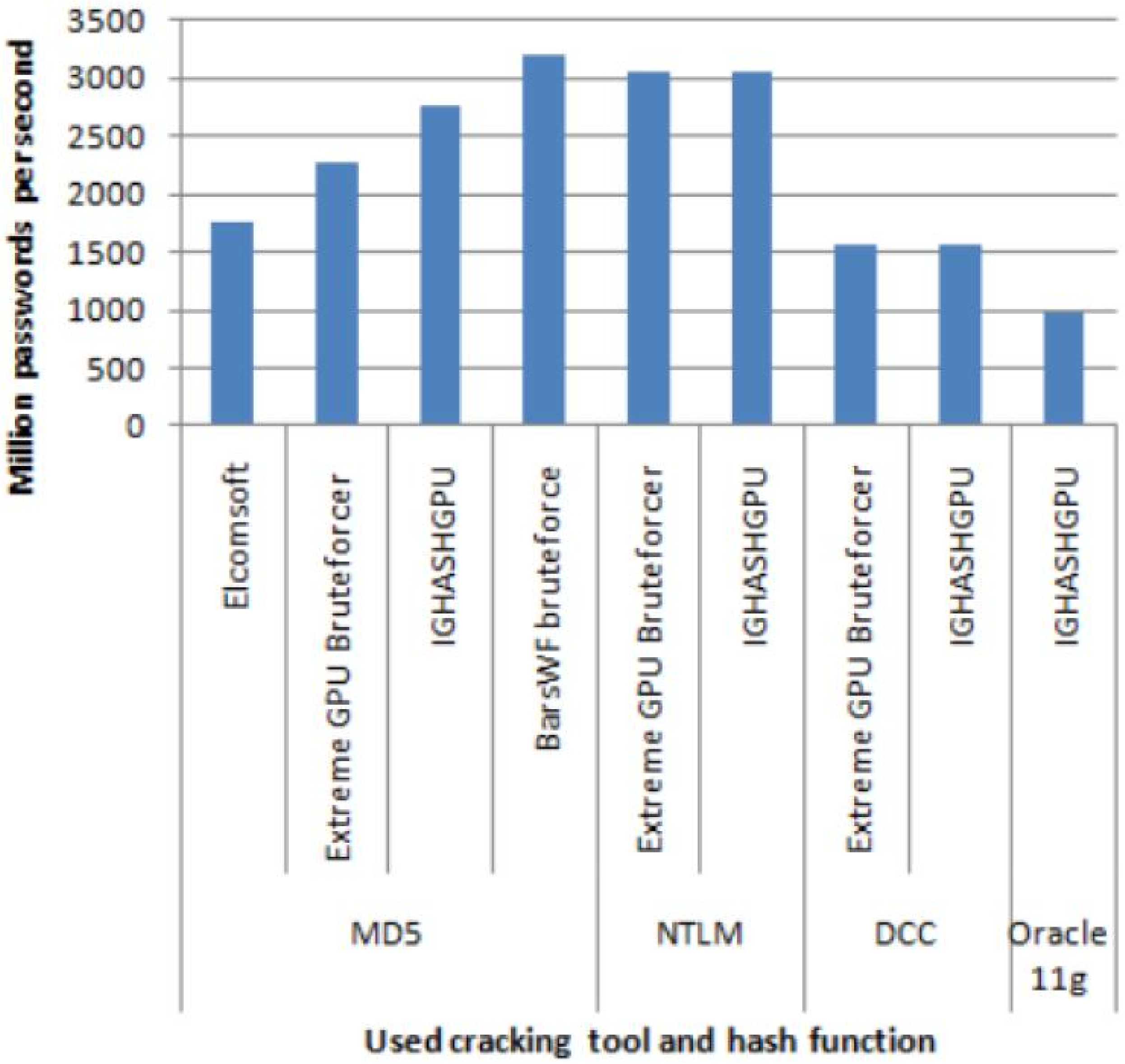
In late 2007, Elcomsoft, a software company based in Russia, came out with the first password recovery program to take advantage of the nVideo GeForce 8800 GPU and increase password cracking speed by 25 times over a CPU. They boldly claimed that an eight character Windows password that normally took two months to brute force on a machine using a CPU, took only two to five days with their software and a high end GPU. That is a huge difference! Let's look at how this is possible.

The reason why GPU's are perfect for password cracking is because of their parallel architecture. Andrew Humber, nVidea's spokesman, described it as "A normal computer processor would read a book starting at page one and finishing at page 500, a GPU would take that book tear it into a 100,000 pieces, and read all those pieces at the same time."

CPU's today usually have two, four or even eight cores whereas GPU's have hundreds of internal processing units known as stream processing units. This is what makes them great for password cracking and when looking for a GPU for password cracking, remember that the more streams the better.

Programming an application for a GPU is much more difficult compared to traditional programming. To make it simpler, nVidea came out with the CUDA architecture and the OpenCL framework. These were created to give developers the ability to use standard programming languages like C and C++ to develop GPU applications, and is what was used to create the password cracking programs you will be using later on in this course.

To show you how much faster password cracking is with GPU's, examine the graph below. In the test, two nVidea GTX295 cards were used with a few different programs and password hashes. As you can see, they passed the billion passwords/second mark.



Source: http://staff.science.uva.nl/~delaat/sne-2009-2010/p34/report.pdf

Computer Clusters

It doesn’t matter if you have the fastest CPU today or the greatest video card, brute forcing a password can still take years even at a rate of 3 billion passwords/second as we saw in the graphics card data. This is because the amount of possible password combinations can get extremely high. So how is it still possible to crack passwords that have an unbelievable amount of possible combinations? The answer is: computer clusters. Computer clusters are a bunch of interconnected computers that split a job amongst themselves to considerably shorten the time needed to complete it. These computers could be all on a LAN or they could be spread across the whole internet.

There are many different types of computer clusters, but to keep it simple, I will describe how most of them work.

In a computer cluster, there is usually one computer (server) whose sole purpose is to oversee the rest of the computers (clients). The server’s job is to divide a password cracking job into many pieces and assign each piece (work unit) to each computer in the cluster.

In a password cracking job, each unit would be a range from one string to another. For example, one work unit may be AAAA BBBB. This means that client would need to try every password combination from AAAA to BBBB against a password hash.

Once a machine finishes its assigned work unit, it sends the server its results. If the password was found, the server would tell every other machine to stop;

otherwise it would send that machine a new work unit. This process is repeated until the password is cracked.

Using a password cracking cluster can slash the time it takes to crack passwords in half, so if you have access to multiple machines, use them.

How to Secure and Create Strong Passwords

In the past few sections, we have concentrated on **cracking passwords**; now let's talk about how to **create strong passwords** that are nearly impossible to crack for 99% of all password crackers.. no.. crackerers.. yah that seems right.

I could write a few pages on the different methods of creating secure passwords but that would bore the both of us, so I've chosen to show you the method that works for most people. Phrases.

Phrases

I'm sure by now you realize that for your password to be strong, it should be made up of most if not all of the following:

- Length (shoot for at least 14 characters)
- Lowercase characters
- Uppercase characters
- Symbols
- Numbers

What’s the best way to incorporate all of those and at the same time be easy to remember? That’s right, Phrases. Damn you’re good!

If you search for a secure password creator, you will come up with a lot of tools that automatically generate random password that contain all of the above. The problem with these generated passwords is that they are not easy to remember, and what happens to things that you can’t remember? You mark them down somewhere. If someone comes by it, it is now useless. This is why I have always used phrases. Let’s look at some examples:

- IKnowWhatYouDidLastSummer!0
- what would Jesus do?77
- Never gonna give you up! Never gonna let you down!11

Simply pick a phrase that you can easily remember, capitalize some of the letter, add in some punctuation, pop on a number and you got yourself a darn good password. That’s wassup.

Facebook Hacking Course

I know most of you might be wondering to know how to hack Facebook account passwords? Before you try to hack any Facebook password, it is necessary to understand the real ways of hacking that actually work and also those that are simply scam and don’t work.

Thanks to the media, the word “hacker” has gotten a bad reputation. The word summons up thoughts of malicious computer users finding new ways to harass people, defraud corporations, steal information and maybe even destroy the economy or start a war by infiltrating military computer systems. While there’s no denying that there are hackers out there with bad intentions, they make up only a small percentage of the hacker community.

This is the picture of a hacker. Fortunately, it's the one created by the media. Considering the definition above, this course should be entitled "Anti Facebook Hacking Course", as it has nothing to do with the media nonsense. Let me explain you why.

What is this Facebook Hacking Course really about?

Facebook Hacking Course basically contains series of videos which will tell you exactly how hackers hack facebook accounts, what methods they use and how you can avoid falling for these kinds of attacks. The course contains Video Modules parts which will tell you about Facebook hacking and security and with each part there is a lab where you can practice what you learned.

In the Facebook Hacking Course, all your questions and confusions will be answered.

The author of this course is **Rafay Baloch**, an expert into the field of hacking. His Facebook Hacking Course contains:

1. **The exact techniques which hackers use to hack facebook accounts.**
2. **Security tips to protect your facebook account from getting hacked.**
3. **Protecting your Privacy**

And the best part of this course is that he has made it in such a manner that even a 5th grade kid can understand it and learn how to hack facebook account.

With my experience of over 5 years in the field of ethical hacking and security, all I can tell you is that this course contains everything a beginner needs to learn how to hack facebook, and to educate the Internet users to be aware of the common scams and frauds and stay away from them. This course does not demand any prior knowledge about Hacking.

Bonus

By buying this Facebook Hacking Course you will get the following bonus:

1. Secret Anonymizing Techniques

This section will contain 2 bonus videos which will tell you the exact methods used by hackers to hide their identity while doing malicious things online.

2. Direct Email Access and Support

If you get stuck or don't understand anything presented in the course, there is 24/7 support to help you no longer how much time it takes. However this offer is for limited time only.

Click Here to Download Facebook Hacking Course

Congratulations!

You’ve made it through the whole book! With this book, you have been introduced to the basic categories in the subject of password cracking. By now you should be craving for more knowledge! So, what now?

Keep Learning!

That’s right! Keep learning! One of the biggest mistakes I notice with new and intermediate ethical hackers is that they want to know everything at once. They go out and jump from topic to topic. Time passes by and they still don’t know enough about anything. I know, I went through this phase as well. Trust me.

One of the best ways to learn is to purchase books on your topic, subscribe to related blogs, and join ethical hacking communities.

Suggestions

I would love to hear your honest opinion about this book. What did you think of it? What did you like? What didn’t you like? What would you like to see in future versions? What are you interested in? Please visit the following URL to participate in this quick informative survey:

Click here for survey.

That’s all folks! I hope that this course has been a great learning experience for you. If you have any questions please feel free to e-mail me at:

contact@HackThePc.com