Cracking Basic Hashes

Basic Syntax

```
$ john [options] [path to the file]
```

[path to the file] - It is the file containing the hash we're trying to crack, if it's in the same directory we won't need to name the path, just the file.

Automatic Cracking

John has built-in features that detect what type of hash it's being given.

```
$ john --wordlist=[path to wordlist] [path to file]
```

Example Usage:

```
$ john --wordlist=/usr/share/wordlists/rockyou.txt hash_to_crack.txt
```

Identifying Hashes

• John is not always reliable in recognizing hashes automatically.
• We can use online hash-identifiers or python tools like *hash-identifier*.
• After recognizing the hash type, we can then set john to use a specific format.

Format-Specific Cracking

After identifying the hash, we can tell john to use the specific format while cracking the provided hash using the following syntax:

```
$ john --format=[format] --wordlist=[path to wordlist] [path to file]
```

--format = This is the flag to tell John that we're giving it a hash of specific format.
[format] = This is the format that the hash is in.

Example Usage:

```
$ john --format=raw-md5 --wordlist=/usr/share/wordlists/rockyou.txt
hash_to_crack.txt
```

Note: We can list all the format types using -

```
$ john --list=formats
```

Cracking Windows Authentication Hashes

• Authentication hashes are the hashed versions of passwords that are stored by operating systems.
• It's sometimes possible to crack them using the bruteforce method.
• To get our hands on these hashes, we must often already be a privileged user.

NTHash / NTLM

• NThash is the hash format used by modern Windows Operating systems to store user and service passwords in.
• Commonly referred to as “NTLM” which preferences the previous version of Windows format for hashing passwords known as “LM”.
• We can acquire NTHash/NTLM hashes by dumping the SAM database on a Windows machine, by using a tool like Mimikatz or from Active Directory Database.
• We may not have the crack the hash to continue privilege escalation.
• We can conduct a “pass the hash” attack instead.
• But for weak password policy hash cracking is a viable option.

Cracking /etc/shadow Hashes

• On Linux password hashes are stored in /etc/shadow file.
• It also stores information about date of last password change and password expiration information.
• Contains one entry per line for each user or user account of the system.
• Usually this file is only accessible by the root user.
• So, to get our hands on the hashes we need to have sufficient privileges.

Unshadowing

• In order to crack /etc/shadow passwords, we need to combine it with /etc/passwd file in order for John to understand the data.
• As John is very particular about the formats it needs the data in.
• To do this, we use a tool- ***unshadow***.
• The basic syntax is:

```
$ unshadow [path to passwd] [path to shadow]
```

unshadow- invokes the unshadow tool.
[path to passwd] - file that contains the /etc/passwd file
[path to shadow] - file that contains the /etc/shadow file

Example Usage:

```
$ unshadow local_passwd local_shadow > unshadowed.txt
```

Note on the files

When using unshadow, you can either use the entire /etc/passwd and /etc/shadow file- if you have them available, or you can use the relevant line from each, for example:

FILE 1 - local_passwd

Contains the /etc/passwd line for the root user:

root:x:0:0::/root:/bin/bash

FILE 2 - local_shadow

Contains the /etc/shadow line for the root user:

root:$6$2nwjN454g.dv4HN/$m9Z/r2xVfweYVkrr.v5Ft8Ws3/YYksfNwq96UL1FX0OJjY1L6l.DS3KEVsZ9rOVLB/ldTeEL/OIhJZ4GMFMGA0:18576::::::

Cracking

• Now, we will be feeding the output from *unshadow* (unshadowed.txt) directly to John.
• We should not need to specify the mode here as we have made the input specifically for John.
• In some cases, we would need to specify the format as

```
--format=sha512crypt
```

Example command-

```
$ john --wordlist=/usr/share/wordlists/rockyou.txt --format=sha512crypt
unshadowed.txt
```

Single Crack Mode

• Till Now, we have been using John's wordlist mode to crack hashes.
• John has another mode, called Single Crack Mode.
• In this mode, John uses the information provided in the username, to try and work possible passwords by word mangling.

Word Mangling

Suppose we take the username: *Markus*
Example of Word Mangling:

- Markus1, Markus2, Markus3 (etc.)
- MArkus, MARkus, MARKus (etc.)
- Markus!, Markus$, Markus* (etc.)

• In this case John is building it's own dictionary based on the information that is fed and uses a set of rules called “mangling rules”.

Gecos

• John's implementation of the word mangling also features compatibility with the Gecos fields of the UNIX operating systems.
• In the entries of both /etc/passwd and /etc/shadow if we look closely, we see that each field is separated by colon ‘:’. Each of these fields that these records are split into are called Gecos fields.
• John can take information stored in those records, such as full name, home directory name to add in to the wordlist it generates when cracking /etc/shadow hashes with single crack mode.

Using Single Crack Mode

For example, if we wanted to crack the password of the user name “Mike”, using single crack mode, we'd use:

```
$ john --single --format=[format] [path to file]
```

--single- This flag tells John that we want to use single crack mode

Example Usage:

```
$ john --single --format=raw-sha256 hash.txt
```

NOTE:

If we're cracking hashes in single crack mode, we need to change the file format that we're feeding john for it to understand with what data to create a wordlist.
We do this by prepending the hash with the username that the hash belongs to.
So, according to the above example- wo would change the file

From: 1efee03cdcb96d90ad48ccc7b8666033

To: mike:1efee03cdcb96d90ad48ccc7b8666033

Custom Rules

• We can define our own set of rules, which John will use to dynamically create passwords that could be replicated with a certain mangling pattern.
• This is especially useful when we know more information about the structure of the password.

How to Create Custom Rules

Custom rules are defined in the *john.conf* file, usually located in /etc/john/john.conf.
Syntax of these common rules:

The first line-

[List.Rules:THMRules] - Used the define the name of our rule. This is what we will use to call our custom rule as a John argument.

We then use a regex style pattern to define where in word will be modified. The basic and most commonly used modifiers are:

- **Az** - Takes the word and appends it with the characters we define.
- **A0** - Takes the word and prepends it with the characters we define.
- **c** - Capitalizes the character positionally.

- These can be used in combination to define where and what in word we want to modify.
- We, then need to define what characters should be appended, prepended or otherwise included.
- We do this by adding character sets in square brackets [ ] in order they should be used.
- These directly follow the modifier patterns inside of double quotes “ ”.
- Common examples:
  - [0-9] - Will include numbers from 0-9.
  - [0] - Will include only the number 0.
  - [A-z] - Will include both uppercase and lowercase.
  - [A-Z] - Will include only uppercase.
  - [a-z] - Will include only lowercase.
  - [a] - Will include only the letter a.
  - [!#$%@] - Will include the symbols !#$%@.

For example - Assume the word “polopassword” is in our wordlist and we need to match the example password - “Polopassword1!”.

Then, we can create an entry that looks like:

```
[List.Rules:PoloPassword]
cAz"[0-9] [!#$%@]"
```

Using Custom Rules

We could then call the above custom rule as a John argument using the “--rule” flag with the name of the rule as:

```
$ john --wordlist=[path to wordlist] --rule=PoloPassword [path to file]
```

Cracking Password Protected Zip Files

- We can use John to crack the password of password protected zip files.

• We will use a separate part of the John suite of tools to convert the zip file into a format that John will understand.

Zip2John

• This tool is used to convert the zip file into a hash format that John is able to understand.
• The basic syntax is like:

```
$ zip2john [options] [zip file] > [output file]
```

[options] - Allows us to pass specific checksum options to zip2john. This shouldn't often be necessary.
[zip file] - The path to the zip file.

Example Usage:

```
$ zip2john zipfile.zip > zip_hash.txt
```

Cracking

Then we can take the output file from zip2john and feed it directly to John as:

```
$ john --wordlist=/usr/share/wordlists/rockyou.txt zip_hash.txt
```

Cracking Password Protected RAR Archives

Rar2John

• Almost identical to zip2john tool.
• We will use the *rar2john* tool to convert the rar file into a hash format that John is able to understand.
• The basic syntax is:

```
$ rar2john [rar file] > [output file]
```

rar2john - Invokes the rar2john tool
[rar file] - Path to the rar file that we wish to get the hash of

Example Usage:

```
$ rar2john rarfile.rar > rar_hash.txt
```

Cracking

Then we can take the output of the rar2john and feed it to John as:

```
$ john --wordlist=/usr/share/wordlists/rockyou.txt rar_hash.txt
```

Cracking SSH Keys with John

• We can use John to crack the SSH private key password of id_rsa files.
• Unless configured otherwise, we autheticate our SSH login using a password.
• However, we can configure key-based authentication, which lets us use our private key, id_rsa, as an authentication key to login to a remote machine over SSH.
• However, doing so will often require a password.
• We will use John to crack the password to allow authentication over SSH using the key.

SSH2John

• It converts the id_rsa private key that we use to login with SSH into a hash format that John can work with.
• This can be done as:

```
$ ssh2john [id_rsa private key file] > [output file]
```

ssh2john - Invokes the ssh2john tool
[id_rsa private key file] - The path to the id_rsa file we wish to get the hash of.

Example Usage:

```
$ ssh2john id_rsa > id_rsa_hash.txt
```

Cracking

Now we can feed the output directly to John as:

```
$ john --wordlist=/usr/share/wordlists/rockyou.txt id_rsa_hash.txt
```