John The Ripper
Day 26/365
John Basic Syntax

`john [options] [file path]`

- `john`: Invokes the John the Ripper program
- `[options]`: Specifies the options you want to use
- `[file path]`: The file containing the hash you're trying to crack; if it's in the same directory, you won't need to name a path, just the file.

Automatic Cracking

John has built-in features to detect what type of hash it's being given and to select appropriate rules and formats to crack it for you; this isn't always the best idea as it can be unreliable, but if you can't identify what hash type you're working with and want to try cracking it, it can be a good option! To do this, we use the following syntax:

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

- `--wordlist=`: Specifies using wordlist mode, reading from the file that you supply in the provided path
- `[path to wordlist]`: The path to the wordlist you're using, as described in the previous task

Example Usage:

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

Format-Specific Cracking

Once you have identified the hash that you're dealing with, you can tell John to use it 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 you're giving it a hash of a specific format and to use the following format to crack it
- `[format]`: The format that the hash is in

Example Usage:

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

Custom Rules

The first line:

`[List.Rules:THMRules]` is used to define the name of your rule; this is what you will use to call your custom rule a John argument.

We then use a regex style pattern match to define where the word will be modified; again, we will only cover the primary and most common modifiers here:

- `Az`: Takes the word and appends it with the characters you define
- `A0`: Takes the word and prepends it with the characters you define
- `c`: Capitalises the character positionally

These can be used in combination to define where and what in the word you want to modify.

Lastly, we must define what characters should be appended, prepended or otherwise included. We do this by adding character sets in square brackets `[ ]` where they should be used. These follow the modifier patterns inside double quotes `" "`. Here are some common examples:

- `[0-9]`: Will include numbers 0-9
- `[0]`: Will include only the number 0
- `[A-z]`: Will include both upper and lowercase
- `[A-Z]`: Will include only uppercase letters
- `[a-z]`: Will include only lowercase letters

Please note that:

- `[a]`: Will include only `a`
- `[!£$%@]`: Will include the symbols `!`, `£`, `$`, `%`, and `@`

Putting this all together, to generate a wordlist from the rules that would match the example password `Polopassword1!` (assuming the word `polopassword` was in our wordlist), we would create a rule entry that looks like this:

```
[List.Rules:PoloPassword]
```

```
cAz"[0-9] [!£$%@]"
```

Utilises the following:

- `c`: Capitalises the first letter
- `Az`: Appends to the end of the word
- `[0-9]`: A number in the range 0-9
- `[!£$%@]`: The password is followed by one of these symbols

Using Custom Rules

We could then call this custom rule a John argument using the `--rule=PoloPassword` flag.

As a full command: `john --wordlist=[path to wordlist] --rule=PoloPassword [path to file]`

Zip2John

Similar to unshadow
Basic syntax :

- `zip2john [options] [zip file] > [output file]`
- `[options]`: Allows you to pass specific checksum options to `zip2john`; this shouldn't often be necessary
- `[zip file]`: The path to the Zip file you wish to get the hash of
- `>`: This redirects the output from this command to another file
- `[output file]`: This is the file that will store the output

Example Usage

`zip2john zipfile.zip > zip_hash.txt`

also Rar2John, almost identical

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

- `rar2john`: Invokes the `rar2john` tool
- `[rar file]`: The path to the RAR file you wish to get the hash of
- `>`: This redirects the output of this command to another file
- `[output file]`: This is the file that will store the output from the command

Example Usage

`/opt/john/rar2john rarfile.rar > rar_hash.txt`

SSH2John

`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 you wish to get the hash of
- `>`: This is the output director. We're using it to redirect the output from this command to another file.
- `[output file]`: This is the file that will store the output from

Example Usage

`/opt/john/ssh2john.py id_rsa > id_rsa_hash.txt`

(On kali `python /usr/share/john/ssh2john.py`)