john Package Description

John the Ripper is designed to be both feature-rich and fast. It combines several cracking modes in one program and is fully configurable for your particular needs (you can even define a custom cracking mode using the built-in compiler supporting a subset of C). Also, John is available for several different platforms which enables you to use the same cracker everywhere (you can even continue a cracking session which you started on another platform).

Out of the box, John supports (and autodetects) the following Unix crypt(3) hash types: traditional DES-based, “bigcrypt”, BSDI extended DES-based, FreeBSD MD5-based (also used on Linux and in Cisco IOS), and OpenBSD Blowfish-based (now also used on some Linux distributions and supported by recent versions of Solaris). Also supported out of the box are Kerberos/AFS and Windows LM (DES-based) hashes, as well as DES-based tripcodes.

When running on Linux distributions with glibc 2.7+, John 1.7.6+ additionally supports (and autodetects) SHA-crypt hashes (which are actually used by recent versions of Fedora and Ubuntu), with optional OpenMP parallelization (requires GCC 4.2+, needs to be explicitly enabled at compile-time by uncommenting the proper OMPFLAGS line near the beginning of the Makefile).

Similarly, when running on recent versions of Solaris, John 1.7.6+ supports and autodetects SHA-crypt and SunMD5 hashes, also with optional OpenMP parallelization (requires GCC 4.2+ or recent Sun Studio, needs to be explicitly enabled at compile-time by uncommenting the proper OMPFLAGS line near the beginning of the Makefile and at runtime by setting the OMP_NUM_THREADS environment variable to the desired number of threads).

John the Ripper Pro adds support for Windows NTLM (MD4-based) and Mac OS X 10.4+ salted SHA-1 hashes.

“Community enhanced” -jumbo versions add support for many more password hash types, including Windows NTLM (MD4-based), Mac OS X 10.4-10.6 salted SHA-1 hashes, Mac OS X 10.7 salted SHA-512 hashes, raw MD5 and SHA-1, arbitrary MD5-based “web application” password hash types, hashes used by SQL database servers (MySQL, MS SQL, Oracle) and by some LDAP servers, several hash types used on OpenVMS, password hashes of the Eggdrop IRC bot, and lots of other hash types, as well as many non-hashes such as OpenSSH private keys, S/Key skeykeys files, Kerberos TGTs, PDF files, ZIP (classic PKZIP and WinZip/AES) and RAR archives.

Unlike older crackers, John normally does not use a crypt(3)-style routine. Instead, it has its own highly optimized modules for different hash types and processor architectures. Some of the algorithms used, such as bitslice DES, couldn’t have been implemented within the crypt(3) API; they require a more powerful interface such as the one used in John. Additionally, there are assembly language routines for several processor architectures, most importantly for x86-64 and x86 with SSE2.

Source: https://github.com/magnumripper/JohnTheRipper/releases
John the Ripper Homepage | Kali John the Ripper Repo

- Author: Solar Designer

- License: GPLv2

Tools included in the john package

mailer – Emails users who have had their passwords cracked

```
root@kali:~# mailer
Usage: /usr/sbin/mailer PASSWORD-FILE
```

john – John the Ripper password cracker

```
root@kali:~# john
John the Ripper password cracker, version 1.8.0.6-jumbo-1-bleeding [linux-x86-64-avx]
Copyright (c) 1996-2015 by Solar Designer and others
Homepage: http://www.openwall.com/john/

Usage: john [OPTIONS] [PASSWORD-FILES]
--single[=SECTION]        "single crack" mode
--wordlist[=FILE] --stdin wordlist mode, read words from FILE or stdin
                  --pipe  like --stdin, but bulk reads, and allows rules
--loopback[=FILE]         like --wordlist, but fetch words from a .pot file
--dupe-suppression        suppress all dupes in wordlist (and force preload)
--prince[=FILE]           PRINCE mode, read words from FILE
--encoding=NAME           input encoding (eg. UTF-8, ISO-8859-1). See also
                          doc/ENCODING and --list=hidden-options.
--rules[=SECTION]         enable word mangling rules for wordlist modes
--incremental[=MODE]      "incremental" mode [using section MODE]
--mask=MASK               mask mode using MASK
--markov[=OPTIONS]        "Markov" mode (see doc/MARKOV)
--external=MODE           external mode or word filter
--stdout[=LENGTH]         just output candidate passwords [cut at LENGTH]
--restore[=NAME]          restore an interrupted session [called NAME]
--session=NAME            give a new session the NAME
--status[=NAME]           print status of a session [called NAME]
--make-charset=FILE       make a charset file. It will be overwritten
--show[=LEFT]             show cracked passwords [if =LEFT, then uncracked]
--test[=TIME]             run tests and benchmarks for TIME seconds each
--users=[-]LOGIN|UID[,..] [do not] load this (these) user(s) only
--groups=[-]GID[,..]      load users [not] of this (these) group(s) only
--shells=[-]SHELL[,..]    load users with[out] this (these) shell(s) only
--salts=[-]COUNT[:MAX]    load salts with[out] COUNT [to MAX] hashes
```

```
--save-memory=LEVEL       enable memory saving, at LEVEL 1..3
--node=MIN[-MAX]/TOTAL    this node's number range out of TOTAL count
--fork=N                  fork N processes
--pot=NAME                pot file to use
--list=WHAT               list capabilities, see --list=help or doc/OPTIONS
--format=NAME             force hash of type NAME. The supported formats can
                          be seen with --list=formats and --list=subformats
```

unafs – Script to warn users about their weak passwords

```
root@kali:~# unafs
Usage: unafs DATABASE-FILE CELL-NAME
```

unshadow – Combines passwd and shadow files

```
root@kali:~# unshadow
Usage: unshadow PASSWORD-FILE SHADOW-FILE
```

unique – Removes duplicates from a wordlist

```
root@kali:~# unique
Usage: unique [-v] [-inp=fname] [-cut=len] [-mem=num] OUTPUT-FILE [-ex_file=FNAME2] [-ex_file_only=FNAME2]

    reads from stdin 'normally', but can be overridden by optional -inp=
    If -ex_file=XX is used, then data from file XX is also used to
    unique the data, but nothing is ever written to XX. Thus, any data in
    XX, will NOT output into OUTPUT-FILE (for making iterative dictionaries)
    -ex_file_only=XX assumes the file is 'unique', and only checks against XX
    -cut=len  Will trim each input lines to 'len' bytes long, prior to running
    the unique algorithm. The 'trimming' is done on any -ex_file[_only] file
    -mem=num.  A number that overrides the UNIQUE_HASH_LOG value from within
    params.h.  The default is 21.  This can be raised, up to 25 (memory usage
    doubles each number).  If you go TOO large, unique will swap and thrash and
    work VERY slow

    -v is for 'verbose' mode, outputs line counts during the run
```

unshadow Usage Example

Combine the provided passwd ***(passwd)*** and shadow ***(shadow)***(shadow) and redirect them to a file ***(> unshadowed.txt)***:

```
root@kali:~# unshadow passwd shadow > unshadowed.txt
```

john Usage Example

Using a wordlist ***(–wordlist=/usr/share/john/password.lst)***, apply mangling rules ***(–rules)*** and attempt to crack the password hashes in the given file ***(unshadowed.txt)***:

```
root@kali:~# john --wordlist=/usr/share/john/password.lst --rules unshadowed.txt
Warning: detected hash type "sha512crypt", but the string is also recognized as "crypt"
Use the "--format=crypt" option to force loading these as that type instead
Loaded 1 password hash (sha512crypt [64/64])
toor             (root)
guesses: 1  time: 0:00:00:07 DONE (Mon May 19 08:13:05 2014)  c/s: 482  trying: 1701d - andrew
Use the "--show" option to display all of the cracked passwords reliably
```

unique Usage Example

Using verbose mode ***(-v)***, read a list of passwords ***(-inp=allwords.txt)*** and save only unique words to a file ***(uniques.txt)***:

```
root@kali:~# unique -v -inp=allwords.txt uniques.txt
Total lines read 6089 Unique lines written 5083
```