NAME

local-unbound-control, local-unbound-control-setup - Local-unbound remote server control utility.

SYNOPSIS

local-unbound-control [**-hq**] [**-c** *cfgfile*] [**-s** *server*] *command*

DESCRIPTION

Local-unbound-control performs remote administration on the *local-unbound*(8) DNS server. It reads the configuration file, contacts the Local-unbound server over SSL sends the command and displays the result.

The available options are:

-h Show the version and commandline option help.

-c *cfgfile*
The config file to read with settings. If not given the default config file @ub_conf_file@ is used.

-s *server[@port]*
IPv4 or IPv6 address of the server to contact. If not given, the address is read from the config file.

-q quiet, if the option is given it does not print anything if it works ok.

COMMANDS

There are several commands that the server understands.

start
Start the server. Simply execs *local-unbound*(8). The Local-unbound executable is searched for in the **PATH** set in the environment. It is started with the config file specified using *-c* or the default config file.

stop
Stop the server. The server daemon exits.

reload
Reload the server. This flushes the cache and reads the config file fresh.

verbosity *number*
Change verbosity value for logging. Same values as **verbosity** keyword in *unbound.conf*(5). This new setting lasts until the server is issued a reload (taken from config file again), or the next

verbosity control command.

log_reopen
Reopen the logfile, close and open it. Useful for logrotation to make the daemon release the file it is logging to. If you are using syslog it will attempt to close and open the syslog (which may not work if chrooted).

stats
Print statistics. Resets the internal counters to zero, this can be controlled using the **statistics-cumulative** config statement. Statistics are printed with one [name]: [value] per line.

stats_noreset
Peek at statistics. Prints them like the **stats** command does, but does not reset the internal counters to zero.

status
Display server status. Exit code 3 if not running (the connection to the port is refused), 1 on error, 0 if running.

local_zone *name type*
Add new local zone with name and type. Like **local-zone** config statement. If the zone already exists, the type is changed to the given argument.

local_zone_remove *name*
Remove the local zone with the given name. Removes all local data inside it. If the zone does not exist, the command succeeds.

local_data *RR data...*
Add new local data, the given resource record. Like **local-data** config statement, except for when no covering zone exists. In that case this remote control command creates a transparent zone with the same name as this record.

local_data_remove *name*
Remove all RR data from local name. If the name already has no items, nothing happens. Often results in NXDOMAIN for the name (in a static zone), but if the name has become an empty nonterminal (there is still data in domain names below the removed name), NOERROR nodata answers are the result for that name.

local_zones
Add local zones read from stdin of local-unbound-control. Input is read per line, with name space

type on a line. For bulk additions.

local_zones_remove
Remove local zones read from stdin of local-unbound-control. Input is one name per line. For bulk removals.

local_datas
Add local data RRs read from stdin of local-unbound-control. Input is one RR per line. For bulk additions.

local_datas_remove
Remove local data RRs read from stdin of local-unbound-control. Input is one name per line. For bulk removals.

dump_cache
The contents of the cache is printed in a text format to stdout. You can redirect it to a file to store the cache in a file.

load_cache
The contents of the cache is loaded from stdin. Uses the same format as dump_cache uses. Loading the cache with old, or wrong data can result in old or wrong data returned to clients. Loading data into the cache in this way is supported in order to aid with debugging.

lookup *name*
Print to stdout the name servers that would be used to look up the name specified.

flush *name*
Remove the name from the cache. Removes the types A, AAAA, NS, SOA, CNAME, DNAME, MX, PTR, SRV and NAPTR. Because that is fast to do. Other record types can be removed using **flush_type** or **flush_zone**.

flush_type *name type*
Remove the name, type information from the cache.

flush_zone *name*
Remove all information at or below the name from the cache. The rrsets and key entries are removed so that new lookups will be performed. This needs to walk and inspect the entire cache, and is a slow operation. The entries are set to expired in the implementation of this command (so, with serve-expired enabled, it'll serve that information but schedule a prefetch for new information).

flush_bogus
Remove all bogus data from the cache.

flush_negative
Remove all negative data from the cache. This is nxdomain answers, nodata answers and servfail answers. Also removes bad key entries (which could be due to failed lookups) from the dnssec key cache, and iterator last-resort lookup failures from the rrset cache.

flush_stats
Reset statistics to zero.

flush_requestlist
Drop the queries that are worked on. Stops working on the queries that the server is working on now. The cache is unaffected. No reply is sent for those queries, probably making those users request again later. Useful to make the server restart working on queries with new settings, such as a higher verbosity level.

dump_requestlist
Show what is worked on. Prints all queries that the server is currently working on. Prints the time that users have been waiting. For internal requests, no time is printed. And then prints out the module status. This prints the queries from the first thread, and not queries that are being serviced from other threads.

flush_infra *all|IP*
If all then entire infra cache is emptied. If a specific IP address, the entry for that address is removed from the cache. It contains EDNS, ping and lameness data.

dump_infra
Show the contents of the infra cache.

set_option *opt: val*
Set the option to the given value without a reload. The cache is therefore not flushed. The option must end with a ':' and whitespace must be between the option and the value. Some values may not have an effect if set this way, the new values are not written to the config file, not all options are supported. This is different from the set_option call in libunbound, where all values work because Local-unbound has not been initialized.

The values that work are: statistics-interval, statistics-cumulative, do-not-query-localhost, harden-short-bufsize, harden-large-queries, harden-glue, harden-dnssec-stripped, harden-below-nxdomain, harden-referral-path, prefetch, prefetch-key, log-queries, hide-identity,

hide-version, identity, version, val-log-level, val-log-squelch, ignore-cd-flag, add-holddown, del-holddown, keep-missing, tcp-upstream, ssl-upstream, max-udp-size, ratelimit, ip-ratelimit, cache-max-ttl, cache-min-ttl, cache-max-negative-ttl.

get_option *opt*

Get the value of the option. Give the option name without a trailing ':'. The value is printed. If the value is "", nothing is printed and the connection closes. On error 'error ...' is printed (it gives a syntax error on unknown option). For some options a list of values, one on each line, is printed. The options are shown from the config file as modified with set_option. For some options an override may have been taken that does not show up with this command, not results from e.g. the verbosity and forward control commands. Not all options work, see list_stubs, list_forwards, list_local_zones and list_local_data for those.

list_stubs

List the stub zones in use. These are printed one by one to the output. This includes the root hints in use.

list_forwards

List the forward zones in use. These are printed zone by zone to the output.

list_insecure

List the zones with domain-insecure.

list_local_zones

List the local zones in use. These are printed one per line with zone type.

list_local_data

List the local data RRs in use. The resource records are printed.

insecure_add *zone*

Add a **domain-insecure** for the given zone, like the statement in unbound.conf. Adds to the running Local-unbound without affecting the cache contents (which may still be bogus, use **flush_zone** to remove it), does not affect the config file.

insecure_remove *zone*

Removes domain-insecure for the given zone.

forward_add [*+i*] *zone addr ...*

Add a new forward zone to running Unbound. With +i option also adds a *domain-insecure* for the zone (so it can resolve insecurely if you have a DNSSEC root trust anchor configured for other

names). The addr can be IP4, IP6 or nameserver names, like *forward-zone* config in unbound.conf.

forward_remove [*+i*] *zone*
Remove a forward zone from running Unbound. The +i also removes a *domain-insecure* for the zone.

stub_add [*+ip*] *zone addr ...*
Add a new stub zone to running Unbound. With +i option also adds a *domain-insecure* for the zone. With +p the stub zone is set to prime, without it it is set to notprime. The addr can be IP4, IP6 or nameserver names, like the *stub-zone* config in unbound.conf.

stub_remove [*+i*] *zone*
Remove a stub zone from running Unbound. The +i also removes a *domain-insecure* for the zone.

forward [*off* | *addr ...*]
Setup forwarding mode. Configures if the server should ask other upstream nameservers, should go to the internet root nameservers itself, or show the current config. You could pass the nameservers after a DHCP update.

Without arguments the current list of addresses used to forward all queries to is printed. On startup this is from the forward-zone "." configuration. Afterwards it shows the status. It prints off when no forwarding is used.

If *off* is passed, forwarding is disabled and the root nameservers are used. This can be used to avoid to avoid buggy or non-DNSSEC supporting nameservers returned from DHCP. But may not work in hotels or hotspots.

If one or more IPv4 or IPv6 addresses are given, those are then used to forward queries to. The addresses must be separated with spaces. With '@port' the port number can be set explicitly (default port is 53 (DNS)).

By default the forwarder information from the config file for the root "." is used. The config file is not changed, so after a reload these changes are gone. Other forward zones from the config file are not affected by this command.

ratelimit_list [*+a*]
List the domains that are ratelimited. Printed one per line with current estimated qps and qps limit from config. With +a it prints all domains, not just the ratelimited domains, with their estimated qps. The ratelimited domains return an error for uncached (new) queries, but cached queries work

as normal.

ip_ratelimit_list [*+a*]
List the ip addresses that are ratelimited. Printed one per line with current estimated qps and qps limit from config. With +a it prints all ips, not just the ratelimited ips, with their estimated qps. The ratelimited ips are dropped before checking the cache.

list_auth_zones
List the auth zones that are configured. Printed one per line with a status, indicating if the zone is expired and current serial number.

auth_zone_reload *zone*
Reload the auth zone from zonefile. The zonefile is read in overwriting the current contents of the zone in memory. This changes the auth zone contents itself, not the cache contents. Such cache contents exists if you set Local-unbound to validate with for-upstream yes and that can be cleared with **flush_zone** *zone*.

auth_zone_transfer *zone*
Transfer the auth zone from master. The auth zone probe sequence is started, where the masters are probed to see if they have an updated zone (with the SOA serial check). And then the zone is transferred for a newer zone version.

rpz_enable *zone*
Enable the RPZ zone if it had previously been disabled.

rpz_disable *zone*
Disable the RPZ zone.

view_list_local_zones *view*
list_local_zones for given view.

view_local_zone *view name type*
local_zone for given view.

view_local_zone_remove *view name*
local_zone_remove for given view.

view_list_local_data *view*
list_local_data for given view.

view_local_data *view RR data...*
local_data for given view.

view_local_data_remove *view name*
local_data_remove for given view.

view_local_datas_remove *view*
Remove a list of *local_data* for given view from stdin. Like local_datas_remove.

view_local_datas *view*
Add a list of *local_data* for given view from stdin. Like local_datas.

EXIT CODE

The local-unbound-control program exits with status code 1 on error, 0 on success.

SET UP

The setup requires a self-signed certificate and private keys for both the server and client. The script *local-unbound-control-setup* generates these in the default run directory, or with -d in another directory. If you change the access control permissions on the key files you can decide who can use local-unbound-control, by default owner and group but not all users. Run the script under the same username as you have configured in unbound.conf or as root, so that the daemon is permitted to read the files, for example with:

sudo -u local-unbound local-unbound-control-setup

If you have not configured a username in unbound.conf, the keys need read permission for the user credentials under which the daemon is started. The script preserves private keys present in the directory. After running the script as root, turn on **control-enable** in *unbound.conf*.

STATISTIC COUNTERS

The *stats* command shows a number of statistic counters.

threadX.num.queries
number of queries received by thread

threadX.num.queries_ip_ratelimited
number of queries rate limited by thread

threadX.num.cachehits
number of queries that were successfully answered using a cache lookup

threadX.num.cachemiss

number of queries that needed recursive processing

threadX.num.dnscrypt.crypted
number of queries that were encrypted and successfully decapsulated by dnscrypt.

threadX.num.dnscrypt.cert
number of queries that were requesting dnscrypt certificates.

threadX.num.dnscrypt.cleartext
number of queries received on dnscrypt port that were cleartext and not a request for certificates.

threadX.num.dnscrypt.malformed
number of request that were neither cleartext, not valid dnscrypt messages.

threadX.num.prefetch
number of cache prefetches performed. This number is included in cachehits, as the original query had the unprefetched answer from cache, and resulted in recursive processing, taking a slot in the requestlist. Not part of the recursivereplies (or the histogram thereof) or cachemiss, as a cache response was sent.

threadX.num.expired
number of replies that served an expired cache entry.

threadX.num.recursivereplies
The number of replies sent to queries that needed recursive processing. Could be smaller than threadX.num.cachemiss if due to timeouts no replies were sent for some queries.

threadX.requestlist.avg
The average number of requests in the internal recursive processing request list on insert of a new incoming recursive processing query.

threadX.requestlist.max
Maximum size attained by the internal recursive processing request list.

threadX.requestlist.overwritten
Number of requests in the request list that were overwritten by newer entries. This happens if there is a flood of queries that recursive processing and the server has a hard time.

threadX.requestlist.exceeded
Queries that were dropped because the request list was full. This happens if a flood of queries need

recursive processing, and the server can not keep up.

threadX.requestlist.current.all
Current size of the request list, includes internally generated queries (such as priming queries and glue lookups).

threadX.requestlist.current.user
Current size of the request list, only the requests from client queries.

threadX.recursion.time.avg
Average time it took to answer queries that needed recursive processing. Note that queries that were answered from the cache are not in this average.

threadX.recursion.time.median
The median of the time it took to answer queries that needed recursive processing. The median means that 50% of the user queries were answered in less than this time. Because of big outliers (usually queries to non responsive servers), the average can be bigger than the median. This median has been calculated by interpolation from a histogram.

threadX.tcpusage
The currently held tcp buffers for incoming connections. A spot value on the time of the request. This helps you spot if the incoming-num-tcp buffers are full.

total.num.queries
summed over threads.

total.num.cachehits
summed over threads.

total.num.cachemiss
summed over threads.

total.num.dnscrypt.crypted
summed over threads.

total.num.dnscrypt.cert
summed over threads.

total.num.dnscrypt.cleartext
summed over threads.

total.num.dnscrypt.malformed
summed over threads.

total.num.prefetch
summed over threads.

total.num.expired
summed over threads.

total.num.recursivereplies
summed over threads.

total.requestlist.avg
averaged over threads.

total.requestlist.max
the maximum of the thread requestlist.max values.

total.requestlist.overwritten
summed over threads.

total.requestlist.exceeded
summed over threads.

total.requestlist.current.all
summed over threads.

total.recursion.time.median
averaged over threads.

total.tcpusage
summed over threads.

time.now
current time in seconds since 1970.

time.up
uptime since server boot in seconds.

time.elapsed

time since last statistics printout, in seconds.

EXTENDED STATISTICS

mem.cache.rrset
Memory in bytes in use by the RRset cache.

mem.cache.message
Memory in bytes in use by the message cache.

mem.cache.dnscrypt_shared_secret
Memory in bytes in use by the dnscrypt shared secrets cache.

mem.cache.dnscrypt_nonce
Memory in bytes in use by the dnscrypt nonce cache.

mem.mod.iterator
Memory in bytes in use by the iterator module.

mem.mod.validator
Memory in bytes in use by the validator module. Includes the key cache and negative cache.

mem.streamwait
Memory in bytes in used by the TCP and TLS stream wait buffers. These are answers waiting to be written back to the clients.

mem.http.query_buffer
Memory in bytes used by the HTTP/2 query buffers. Containing (partial) DNS queries waiting for request stream completion.

mem.http.response_buffer
Memory in bytes used by the HTTP/2 response buffers. Containing DNS responses waiting to be written back to the clients.

histogram.<sec>.<usec>.to.<sec>.<usec>
Shows a histogram, summed over all threads. Every element counts the recursive queries whose reply time fit between the lower and upper bound. Times larger or equal to the lowerbound, and smaller than the upper bound. There are 40 buckets, with bucket sizes doubling.

num.query.type.A
The total number of queries over all threads with query type A. Printed for the other query types

as well, but only for the types for which queries were received, thus =0 entries are omitted for brevity.

num.query.type.other
Number of queries with query types 256-65535.

num.query.class.IN
The total number of queries over all threads with query class IN (internet). Also printed for other classes (such as CH (CHAOS) sometimes used for debugging), or NONE, ANY, used by dynamic update. num.query.class.other is printed for classes 256-65535.

num.query.opcode.QUERY
The total number of queries over all threads with query opcode QUERY. Also printed for other opcodes, UPDATE, ...

num.query.tcp
Number of queries that were made using TCP towards the Local-unbound server.

num.query.tcpout
Number of queries that the Local-unbound server made using TCP outgoing towards other servers.

num.query.tls
Number of queries that were made using TLS towards the Local-unbound server. These are also counted in num.query.tcp, because TLS uses TCP.

num.query.tls.resume
Number of TLS session resumptions, these are queries over TLS towards the Local-unbound server where the client negotiated a TLS session resumption key.

num.query.https
Number of queries that were made using HTTPS towards the Local-unbound server. These are also counted in num.query.tcp and num.query.tls, because HTTPS uses TLS and TCP.

num.query.ipv6
Number of queries that were made using IPv6 towards the Local-unbound server.

num.query.flags.RD
The number of queries that had the RD flag set in the header. Also printed for flags QR, AA, TC, RA, Z, AD, CD. Note that queries with flags QR, AA or TC may have been rejected because of that.

num.query.edns.present
number of queries that had an EDNS OPT record present.

num.query.edns.DO
number of queries that had an EDNS OPT record with the DO (DNSSEC OK) bit set. These queries are also included in the num.query.edns.present number.

num.query.ratelimited
The number of queries that are turned away from being send to nameserver due to ratelimiting.

num.query.dnscrypt.shared_secret.cachemiss
The number of dnscrypt queries that did not find a shared secret in the cache. The can be use to compute the shared secret hitrate.

num.query.dnscrypt.replay
The number of dnscrypt queries that found a nonce hit in the nonce cache and hence are considered a query replay.

num.answer.rcode.NXDOMAIN
The number of answers to queries, from cache or from recursion, that had the return code NXDOMAIN. Also printed for the other return codes.

num.answer.rcode.nodata
The number of answers to queries that had the pseudo return code nodata. This means the actual return code was NOERROR, but additionally, no data was carried in the answer (making what is called a NOERROR/NODATA answer). These queries are also included in the num.answer.rcode.NOERROR number. Common for AAAA lookups when an A record exists, and no AAAA.

num.answer.secure
Number of answers that were secure. The answer validated correctly. The AD bit might have been set in some of these answers, where the client signalled (with DO or AD bit in the query) that they were ready to accept the AD bit in the answer.

num.answer.bogus
Number of answers that were bogus. These answers resulted in SERVFAIL to the client because the answer failed validation.

num.rrset.bogus
The number of rrsets marked bogus by the validator. Increased for every RRset inspection that

fails.

unwanted.queries
Number of queries that were refused or dropped because they failed the access control settings.

unwanted.replies
Replies that were unwanted or unsolicited. Could have been random traffic, delayed duplicates, very late answers, or could be spoofing attempts. Some low level of late answers and delayed duplicates are to be expected with the UDP protocol. Very high values could indicate a threat (spoofing).

msg.cache.count
The number of items (DNS replies) in the message cache.

rrset.cache.count
The number of RRsets in the rrset cache. This includes rrsets used by the messages in the message cache, but also delegation information.

infra.cache.count
The number of items in the infra cache. These are IP addresses with their timing and protocol support information.

key.cache.count
The number of items in the key cache. These are DNSSEC keys, one item per delegation point, and their validation status.

dnscrypt_shared_secret.cache.count
The number of items in the shared secret cache. These are precomputed shared secrets for a given client public key/server secret key pair. Shared secrets are CPU intensive and this cache allows Local-unbound to avoid recomputing the shared secret when multiple dnscrypt queries are sent from the same client.

dnscrypt_nonce.cache.count
The number of items in the client nonce cache. This cache is used to prevent dnscrypt queries replay. The client nonce must be unique for each client public key/server secret key pair. This cache should be able to host QPS * ‘replay window‘ interval keys to prevent replay of a query during ‘replay window‘ seconds.

num.query.authzone.up
The number of queries answered from auth-zone data, upstream queries. These queries would

otherwise have been sent (with fallback enabled) to the internet, but are now answered from the auth zone.

num.query.authzone.down
The number of queries for downstream answered from auth-zone data. These queries are from downstream clients, and have had an answer from the data in the auth zone.

num.query.aggressive.NOERROR
The number of queries answered using cached NSEC records with NODATA RCODE. These queries would otherwise have been sent to the internet, but are now answered using cached data.

num.query.aggressive.NXDOMAIN
The number of queries answered using cached NSEC records with NXDOMAIN RCODE. These queries would otherwise have been sent to the internet, but are now answered using cached data.

num.query.subnet
Number of queries that got an answer that contained EDNS client subnet data.

num.query.subnet_cache
Number of queries answered from the edns client subnet cache. These are counted as cachemiss by the main counters, but hit the client subnet specific cache, after getting processed by the edns client subnet module.

num.rpz.action.<rpz_action>
Number of queries answered using configured RPZ policy, per RPZ action type. Possible actions are: nxdomain, nodata, passthru, drop, tcp-only, local-data, disabled, and cname-override.

FILES

@ub_conf_file@
Local-unbound configuration file.

@UNBOUND_RUN_DIR@
directory with private keys (unbound_server.key and unbound_control.key) and self-signed certificates (unbound_server.pem and unbound_control.pem).

SEE ALSO

unbound.conf(5), *local-unbound*(8).