# The Checker Project
# Installation Guide and Operator's Manual

Stephen B. Miller, Peter B. Danzig
Computer Science Department
University of Southern California
Los Angeles, California 90089-0781
smiller@caldera.usc.edu, danzig@usc.edu

## 1 Introduction

The Checker Project is an ongoing effort directed by the USC Computer Science Department that is intended to quantify that portion of the Internet bandwidth used by unnecessary port 53 name server traffic. Earlier research by Danzig, Obraczka, and Kumar in "An Analysis of Wide- Area Name Server Traffic" (Spring of 1992, USC TR 92-504) has shown that the port 53 traffic may be as much as 20 times what is really necessary. The analysis that led to that conclusion was done offline using trace files. The Checker Project is an attempt to write their methods into a package that can perform realtime monitoring of name server traffic and give fast feedback on misbehaving systems.

The ultimate goal of this work is to reduce the number of name server packets traversing the Internet. This will be acheived by highlighting inefficient practices and improperly installed subsystems. It is hoped that by simply showing in quantitative terms the effects of an improper installation the administrators of that location will feel obliged to modify their setup. No enforcement tactics are forseen, nor are they believed to be necessary.

Note that for all descriptions in this report the term "query name" includes the query type parameter from the query packet. In record keys within checker's database the query type is used as a modifier on the query name to keep the different kinds of name translation requests separate.

### 1.1 Installation

#### 1.1.1 BIND code alteration

Checker is added to two of the BIND named daemon files. Two places are altered in ns_req.c and one place in ns_resp. For both BIND 4.8.3 and BIND 4.9 the changes are relatively simple, with the relevant code changes shown in Section 3. The Makefile in the named subdirectory also must be altered to include the cklib.a file as shown.

To build the new version of named please perform the following steps:

1. Create a new subdirectory, perhaps at the same level as named in the bind tree, to hold the checker code. Change the default working directory (cd command) to the new directory and unload the checker.tar file there. If you are using BIND 4.8.3 copy (or diff) the `ns_req.c and ns_resp.c` to the named directory.

2. Build the checker library by typing " `make -f cklib.mk`" at the shell prompt.

3. Change default (cd) to the named directory and alter `ns_req.c, ns_resp.c, and Makefile` as shown in Section 3.

4. Build the new named daemon by typing " `make`" at the shell prompt and install the executables in the normal manner.

#### 1.1.2 Log and report file directory

The checker subsystem produces two files during normal operation: a log file and a report file. The path to the files is specified in the checker.cfg configuration file. It is suggested that a separate directory be created under `/usr, /var, or /adm`. This directory must be present before the altered nameserver is started.

Please create the directory specified in the configuration file using mkdir, or alter the configuration file to point to an existing directory.

Note that the default log file is written to `/etc`. Normally the configuration file carries a request to change the log file to whatever directory has been selected. This means that a checker.log file will exist in `/etc`, usually having only a single entry for each time that the checker code is first invoked after restarting the name server.

#### 1.1.3 Configuration file

The checker subsystem reads a configuration file when the first query is processed or by direction of the user. The configuration file is assumed to reside in `/etc` and have the name checker.cfg. Configuration options are explained in their own section below. Please place a copy of the configuration file in `/etc` before attempting to start the new nameserver.

# 2 Operation

## 2.1 Control commands

Two special query names are trapped and used to trigger activity within the checker subsystem:

1. " checker.control.command.1." is used to request that the checker code re-read the configuration file. This is used to change checker's characteristics without taking the name server down.
2. "checker.control.command.2." is used to trigger the report generator. Reports are done by a child process that is created when this command is detected. The child process appends the report to any existing file, or creates a new file if it doesn't exist. Only one report may be active at a time, with a time out factor of 30 minutes. The child process acknowledges back to the nameserver when it is finished, allowing a new report to proceed after that point. If the report process crashes, or take more than 30 minutes then any new report request after the 30 minute time period will cause a kill command to be issued against the report child, and the new report will then be started.

Control commands are usually sent using the nslookup tool. It is important to first set nslookup's recursion desired bit OFF using the " `set norecursion`" command to nslookup. The default server should also be set to the address of whatever name server is hosting Checker. After that the Checker control command is issued by typing " `checker.control.command.1.`" or " `checker.control.command.2.`" at the nslookup prompt. Note that the trailing dot prevents the default domain from being appended to the control command.

## 2.2 Log file entries

The checker subsystem produces a log file as the nameserver runs. The log echos the configuration as it is read, memory usage on a periodic basis, report process starts and stops, and names selected for PEC interdiction. As a default the log file is created in `/etc` as checker.log. Usually the configuration file has a new log file path as its first entry, reducing the `/etc/checker.log` to having only a single entry for each time the checker code is first invoked after restarting the name server.

## 2.3 Consequences of Using Checker

The Checker addition to the BIND nameserver software has two modes: As a sophisticated cache and reporting tool, and as an active diagnostic device for discovering bad nameserver implementations. This section of the document attempts to list all of the resource related consequences of including checker in a nameserver installation. The discussion is broken into four areas: Memory, Disk, PEC Checking, and Security.

### 2.3.1 Memory

In the first, simpler, mode a database is built from the incoming queries received by the nameserver. Any matching replies add information to the records created by the initial query. Therefore the first consequence of using checker is to increase the amount of memory used by the nameserver as a whole to cover the data retained for checker's reporting functions.

The amount of memory used by checker is controlled by the length of time a record is allowed to remain in the cache without being accessed. This time length is in the configuration file, along with some parameters that control how often the database is scanned for overage records. Operation of the timed deletion feature can be completely disabled, and is usually turned off on low utilization nameservers (they are usually rebooted before the database can get very large).

The actual amount of memory consumed by the nameserver and by checker is reported in two ways. The total dynamic memory allocated (using calloc, malloc, etc) by the nameserver, including the non-checker operations, is periodically reported in the log file. The calculated total used just by checker is located in the statistics page of the checker report. The log file entry can be turned off by a configuration file flag.

### 2.3.2 File space

In addition to the source files (and objects) used to build checker, four files are used during normal operation.

1. The first is the configuration file held in the /etc directory. This is a small text file that has negligible effect on disk resources.

2. The second file is the default log file created in /etc. This file is opened and appended to whenever checker is first started. If the configuration file does not include an entry redirecting the log file then it is also the ongoing operations log.

3. The third file is the redirected log file. Because the checker log and report files may accumulate over time (or there may be several renamed versions of them) it is normal to redirect them to their own directory. The log file contains entries echoing the configuration data whenever the configuration file is read. It also contains notations whenever PEC checking on a query name is started or stopped, whenever a report is started, and for memory usage.

4. The fourth file is the report file. This is appended to under three conditions: a) a report is requested by an operator, b) records with error detections are purged from the database, and c) PEC checking selects a query name to be interdicted (see below). This file has the potential to grow very large if not deleted periodically. A new file will be created if an existing one cannot be found.

Note that checker is not initialized until the first query is made to the nameserver. It is normal to not see the log or output files for a few seconds after the nameserver is first started.

### 2.3.3 PEC Checking

In its more sophisticated operation mode Checker can be used to randomly "interdict" replies to randomly selected query names. The purpose of this is to check the retry and timeout logic of nameservers or resolvers making queries for the selected name. Because the names are selected randomly it is hoped that all client nameservers will have their logic checked over a period of time, without interfering with normal operations.

The frequency with which names are selected, how long they are interdicted, and the length of post check grace period (when the name can't be reselected) are all set by fields in the configuration file. There also exists a flag for completely bypassing the PEC Checking code.

The effect of enabling PEC will be to occasionally see timeouts by client nameservers. If the clients are properly implemented they will select an alternate nameserver and go on about their business. Some of the improperly constructed resolvers or nameservers will continue to retry the PEC blocked name and be recorded in the checker report file as having an excessive number of retries.

### 2.3.4 Security

Security concerns revolve mainly around unauthorized use and malicious use of resources. For Checker there really isn't much in the way of unauthorized use problems, it is a research project and the reports are really only useful to those participating. For malicious use there is one situation where an outside user could cause a problem. In later revisions this will be changed, but the current software does not verify the source of the control commands. The consequences of this are that a malicious user could send a stream of configuration read requests or report requests and tie up the nameserver. The effect of this is greatly reduced by Checker's refusal to start a new report while another is in progress, and by the fact that the configuration file itself is under the system manager's control. In both cases the real danger is of the log or report files growing too large. (A non-responsive name server is taken care of by DNS itself).

## 2.4 Configuration options

Many of the features in the checker code are controlled through entries in the configuration file. The following paragraphs explain each possible entry and list the default value. All entries consist of two text tokens followed by the parameter value.

1. checker enabled *option*
   *option* = `"off" or "on"`, sets variable `checker_enabled` to 0 or 1 respectively.
   The checker enabled flag is an overall control on checker's operation. It is checked after looking at the incoming query name. This permits a disabled checker to still respond to control commands that request the configuration file to be read or reports to be generated. If this flag is disabled no database operations will occur, and no failure mode checking will be done.

2. debug flag *option*
   *option* = `%d integer value`, sets variable `ck_debug_flag`.
   The debug flag is used to turn on trace data for each query name seen. The output is written to the log file. This generates a lot of data, and slows the name server operation. By default it is turned off.

3. memory logging *option*
   *option* = `"off" or "on"`, sets variable `memory_logging` to 0 or 1 respectively.
   The memory logging flag determines whether or not checker periodically logs the amount of dynamic memory the name server is using. This includes that memory used by named for its own cache and can give a much higher figure than the calculated value that checker puts in the statistics report.

4. domain checking *option*
   *option* = `"off" or "on"`, sets variable `local_domain_checking` to 0 or 1 respectively.
   The domain checking flag controls whether or not checker looks to see if the packet is associated with a node within the same network (as defined by the IP address class) as the name server that is hosting checker. If domain checking is enabled then packets from the local network are excluded from checker's database.

5. port53 checking *option*
   *option* = `"off" or "on"`, sets variable `port_53_checking` to 0 or 1 respectively.
   The port 53 checking flag controls whether or not checker looks to see if the packet is from another name server, or from a simple resolver. If the packet originated on a port other than 53 then it must be from a resolver. If port 53 checking is enabled then packets that are not from other name servers are excluded from checker's database.

6. maintenance interval *option*
   *option* = `%d integer value`, sets variable `maintenance_interval`.
   The maintenance interval determines how often checker examines the time deletion queue, reports memory usage, and looks for finished report processes. This is expected to be 3600 seconds under normal production use, but has been set to 600 seconds for testing.

7. time queue *option*
   *option* = `"off" or "on"`, sets variable `disable_tq_flag` to 1 or 0 respectively.
   The time deletion queue is the method by which checker purges its cache of old entries. If this is disabled the cache has the potential to increase without bounds. In practice a low use server will usually be rebooted before the cache gets too large. This should be enabled on a high traffic name server to avoid using too much memory. The deletion time value described below is used to decide what names get purged when the time queue examination is done.

8. deletion time *option*
   *option* = `%d integer value`, sets variable `qr_deletion_time`.
   The deletion time value sets the amount of time a query name can be inactive before it is purged from checker's database. This value has no meaning if the time queue is disabled.

9. logging file *option*
   *option* = `%s string`, sets variable `logging_file_name`.
   The logging file name is used to request checker to close the current logging file and reopen one using the name in the parameter. The close/reopen is done immediately after reading this line from the configuration file, therefore it is usually the first line in the configuration file.

10. output file *option*
    *option* = `%s string`, sets variable `output_file_name`.
    The output file name is used to set the name of the report file. Note that reports are always appended to any existing file of the same name.

11. server name *option*
    *option* = `%s string`, sets variable `srvname`.
    This string is an override on the gethostname() function call that is normally used to discover the local host's name. This can be used if the node hosting the name server is "multihomed", with different node names for each of its network connections.

12. translate address *option*
    *option* = `"off" or "on"`, sets variable `translate_address` to 0 or 1 respectively.
    The translate address determines whether or not the child process doing the report generation uses in-addr.arpa queries to translate IP addresses into host names on the report.

13. error threshold *option*
    *option* = `%d integer value`, sets variable `error_threshold`.
    The error threshold is a value that is used to determine whether or not an accumulated error count is worthy of reporting. Its value should be in relation to the amount of traffic a name server sees.

14. error mask *option*
    *option* = `%X hexadecimal integer value`, sets variable `echeck_mask`.
    The error mask defines what failure modes are watched for. Each mode is assigned a bit position in the mask, the following table lists the bit assignments and the mnemonic name of the failure mode. Please see the comments in the checks.c file for detailed explanations of each of the failure modes.

    - 0x001 did not wait for response (FAST_RX)
    - 0x002 ignored referral and tried again (RECURSION_A)
    - 0x004 too many referrals, recursions (RECURSION_B)
    - 0x008 host named itself in referral (RECURSION_C)
    - 0x010 ignored "no such record" response (ZERO_ANSWER)
    - 0x020 did not handle server failure (resolver, SERVFAIL_A)
    - 0x040 did not handle server failure (name server, SERVFAIL_B)

- 0x080 requery detected inside of TTL value (CACHE_LEAK)
- 0x100 ignored "no such domain" response (NAME_ERROR)
- 0x200 domain doesn't use central cache (NO_CENTRAL_CACHE)
- 0x400 sent bad query (FORMAT_ERROR)

15. authority check *option*
    *option* = `"off" or "on"`, sets variable `nocheck_authority` to 0 or 1 respectively.
    The authority checking flag determines whether or not checker examines the authority bit on name error (NXDOMAIN) failure mode checks. This should normally be off.

16. pec checking *option*
    *option* = `"off" or "on"`, sets variable `pec_checking` to 0 or 1 respectively.
    The pec checking flag controls whether or not the proactive error checking (PEC) features of the checker code are enabled. If PEC is enabled then query names will be selected randomly for special treatment. Special treatment includes: purging of error counts for IP addresses that have queried for that name, interdiction of replies, and a special report of failures detected during the period that replies were being blocked. The pec_interval, pec_length, and pec_block variables determine how often a new query name is select, how long replies will be interdicted, and for how long after a test is a name is blocked from being reselected.

17. pec block *option*
    *option* = `%d integer value`, sets variable `pec_blocked_time`.
    The pec block variable determines how much time must pass before a query name can be reselected for PEC testing by the random number generator. This is currently set to 3600 seconds.

18. pec length *option*
    *option* = `%d integer value`, sets variable `pec_test_length`.
    The pec length variable determines how long responses to a selected query name will be interdicted. This is currently set to 3600 seconds, but may be set to a much shorter time interval as the amount of name server traffic is increased.

19. pec interval *option*
    *option* = `%d integer value`, sets variable `pec_interval`.
    The pec interval determines how often checker selects a new query name for PEC checking. This is currently set to 3600 seconds.

## 2.5 Reports

Currently there is only one report option: "report on everything". This results in a multipage report with the following sections, one to a page:

1. Statistics report This report gives a summary of checker's operation since it was first invoked. Each entity cached in the database is totaled, along with the number of bytes that type of record occupies. Also shown are some statistics on the hash table to help validate the hashing algorithm. The last few lines of the report give static information that sumarizes some of the processing parameters set either through compile time defines or in the configuration file. Note that the total number of bytes in the malloc'd database will not match the number in the log file because this value is calculated from the number of records held in checker's database, and the log file value is returned by a C runtime library call. The difference is the amount of space used by the BIND code.

2. IP address report This part of the report is a simple list of IP addresses (and names if the translate name option is enabled), ordered by the frequency with which they made name translation requests during the recorded processing period. This report is used as the first clue that one requestor may be exceeding the usual or expected number of requests.

3. Query/Response report This part of the report lists the most frequently seen query names, and the IP address that requested their translation. The same name may appear twice if two different machines requested its translation. This report is used in conjunction with the IP address report to detect machines that are making more than their fair share of requests.

4. Error report This part of the report is the heart of the Checker Project. It details the types of errors detected within each IP address' query packet stream. Entries in this report are dependant on the error threshold configuration item. A key part of each entry is the query/response counts under the IP address field. An address that has a low number of unique queries in relation to the total number of queries is most likely misbehaving.

# 3 BIND 4.8.3 changes

## 3.1 ns_req.c

Ns_req.c requires changes in four areas as described below by diff listing fragments:

- Declaration of function type near the top of the file:

```
53,56d52
< #ifdef CHECKER
< extern u_long  checker();
< #endif
<
```

- Capture of incoming query packet:

```
113,116d108
< #ifdef CHECKER
<         checker( 0, msg, msglen, from, qsp );
< #endif
<
```

- Capture of outgoing response packet, and response interdiction for PEC checking:

```
590,595d576
< #ifdef CHECKER
<         if( checker( 1, msg, cp-msg, from, qsp ) == 1 ) {
<             if( qsp != QSTREAM_NULL ) sq_done(qsp); }
<         else {
< #endif CHECKER
<
```

- End block for response interdiction:

```
612,614d592
< #ifdef CHECKER
<         }
< #endif
```

## 3.2 ns_resp.c

Ns_resp.c requires three changes similar to ns_req.c. In the ns_resp.c case there is no need to capture an incoming query packet.

- Declaration of function type near the top of the file:

```
45,48d44
< #ifdef CHECKER
< extern  u_long checker();
< #endif
<
```

- Capture of outgoing response packet, and response interdiction for PEC checking:

```
1191,1195d1097
< #ifdef CHECKER
<         if( checker( 1, msg, msglen, &qp->q_from, qp->q_stream ) == 1 ) {
<             if( qp->q_stream != QSTREAM_NULL ) sq_done(qp->q_stream); }
<         else {
< #endif
```

- End block for response interdiction:

```
1212,1214d1113
< #ifdef CHECKER
<         }
< #endif
```

### 3.3 Makefile

The Makefile in the named subdirectory must be modified to define CHECKER and to include the cklib.a among the libraries to be linked into named.

```
9,10c9
< DEFINES= -DDEBUG -DSTATS -DCHECKER
< #DEFINES= -DDEBUG -DSTATS
---
> DEFINES= -DDEBUG -DSTATS
13c12
< CFLAGS= -O -I../include -DBSD=43 ${DEFINES}
---
> CFLAGS= -O ${DEFINES}
15,17c14,15
< RES=    ../res/libresolv.a
< LIBS= ${RES} ../../checker/libckr.a
< #LIBS= ${RES}
---
> RES=
> LIBS= ${RES}
```