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

Contents

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.

2 Installation

2.1 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 4. The Makefile in the named subdirectory also must be altered to include the libckr.a file as shown.

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

1. Create a new subdirectory at the same level as named in the bind tree, to hold the Checker code. Change the default working directory to the new directory and unload the checker.tar file there.
2. Build the checker library by typing "`make -f cklib.mk`" at the shell prompt.
3. Change to the named directory and alter `ns_req.c, ns_resp.c, and Makefile` as shown in Section 4, or copy the files supplied in checker.tar.
4. Build the new named daemon by typing "`make`" at the shell prompt and install the executables in the normal manner.

2.2 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 name server is started.

Note that the default log file is written to `/usr/tmp/checker.log`, which is configured by a defined constant in `checker.h`. Normally the configuration file carries a request to change the log file to whatever directory has been selected, leaving the default one to have only a single entry for each time the name server is restarted.

2.3 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.

2.4 The dig program

Currently Checker uses `dig` to translate IP addresses into domain names. There is a configuration item used to tell Checker where the executable for `dig` resides.

3 Operation

3.1 Control commands

Two special query names are trapped and used to trigger activity within the checker subsystem. When password checking is turned off the control command has four tokens separated by dots, in the same style as a domain name. When password checking is turned on a fifth token is added at the end to contain the plaintext password which is encrypted and compared against the password string configured in by Checker's operator.

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 report 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 name server when it is finished, allowing a new report to proceed after that point. If the report process crashes, or takes 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 (or the `dig program`). It is important to make sure the recursion desired bit is 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.

3.2 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 (optional), report process starts and stops, and names selected

1	checker enabled	enables database and error checks
2	debug flag	checker input packets to logging file
3	memory logging	write memory use to logging file
4	password string	encrypted password string
5	password check	enables password on commands
6	domain checking	drop packets from local IP address domain
7	port53 checking	drop packets not from port 53
8	maintenance interval	seconds between cleanup scans
9	time queue	enables deletion of old records
10	deletion time	cache time for good queries
11	negative cache	cache time for bad queries
12	logging file	file name for log file
13	output file	file name for report file
14	dig path	path name for dig name xlate program
15	translate address	enables IP address xlate in reports
16	error threshold	lower bound error count to report
17	error mask	error modes to be checked for
18	authority check	look at authority bit on NXDOMAIN
19	pec checking	enables proactive error checking
20	pec block	seconds before allowing recheck of same name
21	pec length	seconds to intedict replies to selected name
22	pec interval	seconds between selecting new name

Figure 1: Runtime Configuration Options

for PEC interdiction. Debug trace output also is written to the log file if the debug configuration switch is turned on.

3.3 Run-Time 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. Please see Figure 1 for a summary list of all of the options that can appear in `checker.cfg`.

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. Also controlled by this flag is a periodic log file entry that summarizes the time queue deletion activity.

4. password string *option*
   option = `%s string`, sets variable `base_password`.
   The password string is an encrypted version of whatever character string the name server operator wishes to use as a password. Password strings can be created for this item using the standalone `makepass` program supplied with Checker which is merely a packaged version of the standard UNIX crypt() function. There is a default password string filled in if this item is missing from the configuration file. The default password is `checker`, with the encrypted version of `checker` loaded from a defined constant at compile time.

5. password check *option*
   option = `"off" or "on"`, sets variable `password_check` to 0 or 1 respectively.
   The password checking flag controls whether or not Checker takes the last token off of a control command (after the fourth dot) and encrypts it as a password. If the encrypted value matches the contents of the password string then the control command is processed.

6. 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.

7. 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.

8. 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.

9. 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.

10. deletion time *option*
    option = `%d integer value`, sets variable `qr_deletion_time`.
    The deletion time value sets the amount of time a valid query name can be inactive before it is purged from checker's database. This value has no meaning if the time queue is disabled. Typically this is set to a magnitude that is similar to the average TTL on a name server record.

11. negative cache time *option*
    option = `%d integer value`, sets variable `negative_cache`.

The negative cache retention time is similar to the deletion time, it sets the length of time that an invalid query name (as indicated by an NXDOMAIN response code in the latest response to that query name) can be inactive before being deleted from the database. Typically the negative cache time is set very low (100 to 400 seconds).

12. 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.

13. 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.

14. dig path *option*
    option = `%s string`, sets variable `dig_path`.
    The dig path name is used to find the `dig` program when the translate address flag is set during report generation. If the dig path name is set to the special string **"internal"** then resolver code is called directly within Checker's report generation routines. This is experimental, and not recommended. Both options exist to allow development of more efficient code.

15. 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.

16. 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.

17. 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 detailed comments on each error mode in the Error Detection Section.

    - 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)

18. 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.

19. 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.

20. 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.

21. 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.

22. 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.

3.4 Reports

Currently there is only one report option; initiated by the "checker.control.command.2." special query name. Five reports are written into the output file when this command is processed. Three of the reports are single pages, the other two can be two pages depending on the number of clients being handled by Checker's host. The five reports are described below with an example image of each report page. The example pages have had $\boxed{\text{n}}$ marks overlaid on them as reference marks for the discussion paragraphs under each report heading.

3.4.1 Checker Processing Statistics Report

This report gives a summary of checker's operation since it was first invoked. Please reference Figure 2, and the seven indicated locations for the following descriptions.

1. The top section of the report summarizes the packet traffic passed through Checker. The **packets seen** entry counts the number of query and response packets that the host name server has fed into Checker. The **discarded** entry is the number of packets that Checker's input filter decided not to process. Note that the **discarded** field usually represents packets not processed by Checker (but still

```
CHECKER PROCESSING STATISTICS at Tue Jul 20 08:20:02 1993

62714 packets seen; 2538 discarded (  4.05 percent)                   1
30088 queries ( 47.98 percent), 323.32 per hour
30088 responses ( 47.98 percent)
Query table (20997 slots), 11.52  pct. filled,  0.748 pct. collisions
IP address table (20997 slots), 7.02   pct. filled,  0.271 pct. collisions

nocheck_authority: 1              ck_debug_flag: 0                    2
translate_address: 1            disable_tq_flag: 0
pec_checking:      0       local_domain_checking: 1
memory_logging:    1             port_53_checking: 0

1164360 bytes in dynamic database (by calculation)                    3
31.36 percent of ip addresses have error records

2579  QR_HTABLE records,  61896 bytes,   5.32 percent of database     4
5030  QR_DATA records,   684080 bytes,  58.75 percent of database
1534  IP_DATA records,   184080 bytes,  15.81 percent of database
481   ERR_DATA records,   63492 bytes,   5.45 percent of database
4778  IPD_DATA records,  114672 bytes,   9.85 percent of database
      name strings,       56140 bytes,   4.82 percent of database

30 secs expected for requery, 5 secs for processing                   5
recursion lb window is 1000 secs, servfail window is 1000 secs
6 entries trigger a recursion error (10 slots max)
6 entries trigger a bad server response error (10 max)

output report filename is /usr/checker/checker.stats                  6
activity log file is /usr/checker/checker.log
path to DIG name translation program is /usr/etc
14400 secs for negative cache, 24 slots in delete queue
unreferenced query records last for 14400 seconds
600 secs per maintenance interval, 4.00   hours per cycle
error check mask is 0x000007F7, with a 50 threshold
3600 secs between PEC starts, 3600 secs is PEC test time
names are blocked after pec checks for 3600 seconds

                                                                      7
Usage Rate Chart - Query counts per bin,   0.50 hours per bin
  384 |           X  X                                             | 384
  360 |           X  X            X    X                           | 360
  337 |           X  XXX          X    X                           | 337
  314 |           X XXXX         XX    X                       XX  | 314
  291 |       X   XXXXXX  X     XXX    X                     X XX  | 291
  267 |       X XXXXXXXX XX X   XXX    X                 X  XX XX  | 267
  244 |       X XXXXXXXXXXX X   XXX    X  X X            X  XX XX  | 244
  221 |       X XXXXXXXXXXX X X XXX    X  X X      X     X  XX XXX | 221
  198 |       XXXXXXXXXXXXXXXXXXXXX    XX X XXX    X     X XXX XXX | 198
  174 |       XXXXXXXXXXXXXXXXXXXXX X XXX XXXXX   XX  X  X XXXXXXX | 174
  151 |       XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX XXXX  X  X XXXXXXX | 151
  128 |     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX XXX XXXXXXXXX | 128
  105 |     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX XXXXXXXXX | 105
   81 | XXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX | 81
   58 | XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX | 58
      +------------------------------------------------------------+
       ^Jul 19 02:26       <--- 29.8 hours --->       Jul 20 08:12^
```

Figure 2: Example Processing Statistics Report Page

handled by the name server) due to the `port53 checking` or `domain checking` flags having been set. The **queries** and **responses** lines break out the number of query packets seen versus the number of responses. This is usually the same. The per hour figure for queries is based on total up time for the name server. For debugging and performance testing reasons the efficiency of the hash tables is also shown in this section.

2. The second part of the report shows the current state of the binary configuration flags set in the `/etc/checker.cfg` file. The variables, and their values, are described in Section 3.3. These flags have wide ranging effect on Checker's operation and it is important to know their state before studying the rest of the report.

3. Following the flags summary are two lines that summarize the amount of storage being used and the percentage of clients that have shown some type of error behavior. 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 name server's operations.

4. The next few lines break out the database in terms of the different type of data records that make up the database. QR_HTABLE records form the head of linked lists of QR_DATA records. The head of the list holds a query name, and the data records hold timing information for the clients that have queried for that particular name. IP_DATA records carry global information about a client's timing patterns, and its accumulated error detection counts. ERR_DATA records are attached to IP_DATA records, but only allocated if a client triggers a detector. IPD_DATA records are used to correlate queries from multiple clients within a single IP addressing domain (on the theory that there should be a caching name server for an address domain that shows many queries for the same name from different clients). It is typical for about 60 percent of the database size to be made up of query data records.

5. The next group of output lines describes the parameters that control detection of timing related errors. The **expected requery time** value is merely the standard delay time that a name server should wait before requerying. This is used to detect clients that are not waiting long enough before reissuing a query. The **processing time** figure is used to estimate the amount of time lag that might be generated with network transfers and code execution on a client machine. The **recursion lb window** and **servfail window** values define "leaky bucket" stacks that are used to detect recursion and bad handling of server failures. The next two lines show the number of entries in the stack that are required to trigger an error detection. (A leaky bucket stack eliminates entries that are older than the window length. The length of the stack then represents a sliding average of activity). These parameters are defined constants in the `checker.h` include file.

6. The next few text lines of the report give static information that summarizes some of the processing parameters set either through compile time defines or in the configuration file. The meaning of these fields is explained in Section 3.3.

7. At the bottom of the Processing Statistics Report is a simple chart showing query activity over the most recent few hours. The width of the chart is set to 58 columns, with each column holding the total number of queries from three maintenance intervals. The top and bottom of the vertical scale is automatically set from the maximum and minimum counts found over the total time span for the chart. Note that the vertical scale is NOT queries per hour, but rather queries per three maintenance intervals.

3.4.2 Unique Queries Report

This report page, shown in Figure 3, lists the top 50 clients in terms of the number of different query records they currently reference in the database. This is a direct measure of the diversity of queries being made by a particular client since a new query record is created for each unique query name/type by a particular client IP address. The list is sorted by count, and that count, plus the percent of total database size is printed.

1. The top part of the report page contains general statistics on how much of Checker's activity is represented by the entries on the report. The `percent of total queries` is the percentage of the number of QR_DATA records in the database. This is not a percentage of the number of different query names seen, but better represents the total amount of the database that the clients in the report use. The `percentage of addresses` value shows what percent of the total number of clients seen by Checker are represented in the report (as represented by IP_DATA records).
2. The body of the report is formed by a list of client's addresses, their domain name (if `translate address` is on), and their contribution to Checker's database. The length of the list is a compile time constant set in `checker.h`. The **count** column is the number of query names that this client has in the database. The **pct** column is the percentage of all QR_HTABLE records, which measures how many of the total unique names the particular client referenced. Note that the client's domain name may be blank even if `translate address` is on if the address cannot be translate or if the host machine busy enough to cause a timeout on requests to `dig` or the resolver.

3.4.3 Name Server Usage Report

This report, shown in Figure 4, lists the top 50 clients in terms of their percentage of the total name server activity during the period in which the client was issuing queries. Percentages are calculated by dividing the clients query count by the number of queries processed by the name server in the time period between the first query seen from the client and the most recent one. If this time span is shorter than three maintenance intervals then it is stretched equally on both ends to be three times the current maintenance interval. Because the percent usage is calculated across a client's active period it often happens that clients with small query counts appear all through the list if their query pattern is bursty.

1. The top section of the report contains only what percent of all clients seen by Checker are contained in the report. Two fixed text lines also give an explanation of the columns of the body of the report.
2. The main part of the report contains four columns: The two right hand columns are client IP addresses and their corresponding domain name (if the `translate address` option is turned on). The two left hand columns show two usage measures to try to highlight the client's contribution to the total load on the server. The leftmost column is the percentage of total name server activity that was devoted to the client's requests **during the client's active period**, as explained above. This is the field that the report is sorted on, and is an attempt to remove longevity from the picture. If the report was sorted simply on the clients' query counts then the report would settle down within a few days to only show those clients that are steady users of this particular name server. For reference the second column is the total number of queries by the client. A client that appears on this list with a large query count is therefore both a steady user and significant ongoing load. A low query count indicates a burst of activity within a short time period, perhaps during a lull in queries from other clients.

```
UNIQUE QUERIES by SOURCE ADDRESS at Tue Jul 20 08:20:02 1993

(ipaddr's on list supplied  38.07 percent of total queries)        1
(ipaddr's on list are   3.26 percent of addresses stored)
(pct is client's percent of total query names stored)
(count is client's total of unique queries in db)

 pct    count     ipaddr             ipaddr DNS name
----------------------------------------------------------------
8.032   404      192.132.30.2     well.sf.ca.us.                   2
1.889   95       143.191.3.1      apache.telebit.com.
1.869   94       192.190.111.22   pandora.sf.ca.us.
1.750   88       192.231.221.1    telerama.pgh.pa.us.
1.750   88       129.29.130.1     doim1.usma.army.mil.
1.412   71       132.151.1.1      CNRI.Reston.VA.US.
1.292   65       26.26.0.247      PENTAGON-AI.ARMY.MIL.
1.193   60       129.137.32.101   ucbeh.san.uc.edu.
1.113   56       192.33.4.10      ns.psi.net.
1.034   52       140.174.73.66    idiom.berkeley.ca.us.
0.934   47       148.59.1.1       heifetz.msen.com.
0.915   46       35.1.1.42        merit.edu.
0.855   43       157.91.128.1     washington.ihets.edu.
0.815   41       192.82.208.3     sgiblab.SGI.COM.
0.716   36       36.56.0.151      Argus.Stanford.EDU.
0.696   35       198.3.124.254    gw.santec.boston.ma.us.
0.696   35       192.132.30.4     nkosi.well.sf.ca.us.
0.676   34       192.220.128.14
0.557   28       128.138.243.151  bruno.cs.colorado.edu.
0.497   25       132.239.254.201  ucsd.edu.
0.477   24       129.79.1.9       iugate.ucs.indiana.edu.
0.477   24       156.98.1.1       nic.state.mn.us.
0.457   23       128.196.128.233  Maggie.Telcom.Arizona.EDU.
0.457   23       192.138.180.129  glenvw.ncook.k12.il.us.
0.398   20       139.130.4.4      uneeda.aarnet.EDU.AU.
0.358   18       130.43.2.2       apple.com.
0.358   18       128.101.101.101  ns.unet.umn.edu.
0.358   18       129.24.8.1       ariel.unm.edu.
0.338   17       131.215.139.100  tybalt.caltech.edu.
0.318   16       192.52.195.10    norad.arc.nasa.gov.
0.318   16       156.98.80.2      sunny.health.state.mn.us.
0.298   15       137.192.240.5    NIC.MR.Net.
0.298   15       128.32.136.12    nettlesome.Berkeley.EDU.
0.298   15       128.52.32.80     life.ai.mit.edu.
0.298   15       128.194.15.2     HELIOS.TAMU.EDU.
0.298   15       16.1.0.2         gatekeeper.dec.com.
0.278   14       192.153.156.22   noc.cerf.net.
0.278   14       132.151.1.35     ietf.cnri.reston.va.us.
0.278   14       192.9.9.1        Sun.COM.
0.278   14       128.125.253.136  usc.edu.
0.258   13       128.163.128.19   s.ms.uky.edu.
0.258   13       128.135.24.100   clout.chi.il.us.
0.258   13       159.127.53.10    kosmos.quantum.epsilon.com.
0.258   13       128.95.120.1     dns1.cac.washington.edu.
0.239   12       149.174.128.1    iha.compuserve.com.
0.239   12       128.138.213.21   csn.org.
0.239   12       131.91.128.3     acc.fau.edu.
0.239   12       128.83.135.25    ccwf.cc.utexas.edu.
0.239   12       139.185.1.5      gatekeeper.oracle.com.
0.239   12       130.233.40.33    batman.cs.hut.fi.
```

Figure 3: Example Queries by Source Address Report Page

```
NAME SERVER USAGE by SOURCE ADDRESS at Tue Jul 20 08:20:50 1993

(list is   3.26 percent of ip addresses stored)                      1
(pct is ipaddr's percent of total over its active period)
(count is ipaddr's total number of queries)

 pct    count     ipaddr                    name
-----------------------------------------------------------------
16.251  137       192.187.138.1    rhythm.com.                        2
14.033  4188      192.132.30.2     well.sf.ca.us.
6.799   2024      128.111.72.60    ferkel.ucsb.edu.
4.225   3         133.4.2.2        wnoc-tyo.wide.ad.jp.
2.941   5         134.109.132.51   saturn.hrz.tu-chemnitz.de.
2.857   5         128.42.17.10     masc1.rice.edu.
2.837   4         150.214.4.35     erica.cica.es.
2.674   5         132.170.108.2    eola.cs.ucf.edu.
2.660   5         130.182.121.1    eagle.calstatela.edu.
2.432   728       129.29.130.1     doim1.usma.army.mil.
2.410   2         192.42.239.1     winnie.fit.edu.
2.392   5         198.89.37.104    morgan.ca.tekelec.com.
2.256   6         192.22.0.9
2.175   452       148.59.1.1       heifetz.msen.com.
2.160   643       192.231.221.1    telerama.pgh.pa.us.
2.147   643       36.56.0.151      Argus.Stanford.EDU.
2.012   604       192.136.137.6    RITCHIE-GIIS.ARMY.MIL.
1.924   573       26.26.0.247      PENTAGON-AI.ARMY.MIL.
1.912   317       129.142.17.240   aau.dk.
1.878   392       132.239.254.201  ucsd.edu.
1.806   538       162.148.13.13    eskimo.com.
1.802   2         192.111.123.25   wave.aoml.erl.gov.
1.765   3         129.69.30.143    osi.BelWue.DE.
1.754   2         192.84.62.1      dec4pt.puug.pt.
1.695   2         192.48.96.9      ftp.UU.NET.
1.663   497       192.190.111.22   pandora.sf.ca.us.
1.604   3         128.8.82.2       vpaap.umd.edu.
1.433   5         132.252.150.16   hertha.exp-math.uni-essen.de.
1.420   10        164.119.100.1
1.418   2         192.216.40.10    ictv.com.
1.418   2         130.195.11.3     kauri.vuw.ac.nz.
1.410   232       131.212.32.6     ub.d.umn.edu.
1.373   409       157.91.128.1     washington.ihets.edu.
1.370   2         141.76.1.9       irz101.inf.tu-dresden.de.
1.366   5         192.12.184.3     pagan.agps.lanl.gov.
1.360   400       152.2.21.1       hermes.oit.unc.edu.
1.316   387       143.191.3.1      apache.telebit.com.
1.310   3         131.78.1.1       dsac.dla.mil.
1.299   388       192.65.218.27    drgate.dra.com.
1.296   15        132.178.16.3     euskadi.idbsu.edu.
1.266   3         142.3.200.53     MERCURY.CS.UREGINA.CA.
1.212   2         142.152.2.2
1.194   4         192.216.177.1    IITBARR.IIT.COM.
1.187   4         192.5.14.6       cv.vortex.com.
1.174   349       36.73.0.4        medmail.Stanford.EDU.
1.154   335       192.152.9.3      sugen1.sugen.sf.ca.us.
1.153   134       136.201.1.3      itdsrv1.ul.ie.
1.136   2         130.130.64.1
1.120   235       192.63.100.18    sys1.unigate1.unisys.com.
1.087   3         130.101.5.4      dax.cc.uakron.edu.
```

Figure 4: Example Name Server Usage Report Page

3.4.4 Query/Response Report

This report page, shown in Figure 5, lists the most frequently seen query names, and the clients that requested their translation. The report is sorted by query count, with the rate per hour, percent of total activity, and number of clients also shown.

1. The top section of the report contains two general statistics that summarize how much of the total name server activity is represented by those names listed in the report. The `percent of total queries` figure is calculated by dividing the number of times the names on the report were referenced by the total number of query packets seen (as reported at the top of the Processing Statistics Report). The `percent of query names` value is calculated by dividing the number of names on the list by the number of names (QR_HTABLE records) stored in the database.

2. The body of the report consists of a series of mini-reports, one per query name. The first line of the mini-report contains the query rate per hour for that name, the number of different clients referencing that name, the raw count of queries for that name, and the percent of the name server's load this name represented during the period that it was being referenced. As in the Name Server Usage report, percent is calculated over the time period between the first and most recent queries unless that period is less than three maintenance intervals long. If the interval is less than three maintenance intervals wide then it is stretched equally on both ends until it is. Also shown on this line is the most recent response code in the reply to a query for this name, the query type, and finally the query name itself.

3. The second part of the query name mini-report is a list of the clients requesting that name to be translated. This section of the report is only meaningful if `translate address` is turned on. The client names have their contribution toward the total query count next to their name. Only a few lines are devoted to each query name, therefore the client machine names are sorted by query count before printing. If any names are dropped due to lack of space then the last of the lines has dots running to the edge of the page.

3.4.5 Error Table Report

This part of the report, shown in Figure 6, is the heart of the Checker Project. It details the types of errors detected within each IP address' query packet stream. Client's are sorted by the rate (per hour) of their error with the largest cumulative count. This attempts to order the clients by the amount of "damage" the unnecessary queries are causing. Also shown are the length of time between the first occurance and most recent occurance of each error, and the first time it was detected. The contents of this report are thresholded by the configuration parameter `error threshold`. This means that no clients are considered for inclusion unless they have at least one failure mode with a count greater than the threshold. It is common for the report to be sparse if requested during the first few hours of name server operation.

1. The first line of each client's "mini-report" contains the client's IP address, its domain name (if `translate address` is turned on), and whether or not queries have originated from a name server (originating on Port 53), or a resolver library call.

2. The second and third lines of the "mini-report" contain a summary of the query activity by that client; including the number of packets (both query and response) and the rate per hour of those packets. Each of query and response also are tagged with counts showing how many of those packets tripped an error detector. The percent of packets with errors is highly dependant on the type of errors the client is making since some errors are detected using query packets, and others are found using response packets.

```
QUERY/RESPONSE TABLE at Tue Jul 20 08:21:53 1993

(list is  45.68 percent of total queries processed)      1
(list is   0.50 percent of query names in database)
(qcnt is total query count, N is number of clients)
(rate is queries per hour for query name)
(pct is percent of all queries during active period)

-rate-- -N- -qcnt- --pct-- -rcode-- -type-- ---------query-name-------------
----------------------------------------------------------------------------
 12.84  1    1187   3.99  nxdomain  MX     momad.li.ny.us
        ferkel.ucsb.edu.(1187),

 13.17  51   1182   4.16  OK        A      village.boston.ma.us
        rhythm.com.(135), decvax.zk3.dec.com.(127), plains.NoDak.edu.(104),
        ccwf.cc.utexas.edu.(88), ercole.polito.it.(62),
        bach.seattleu.edu.(60), iugate.ucs.indiana.edu.(58), ..................

 11.21  97   1030   3.50  OK        CNAME  ietf.cnri.reston.va.us              2
        hermes.oit.unc.edu.(341), ucsd.edu.(77), Argus.Stanford.EDU.(62),
        dns-west.nersc.gov.(37), usc.edu.(32), (29), apple.com.(23),        3
        SPOOLER1.BBN.COM.(21), daiduk.kaist.ac.kr.(19), nic.near.net.(18), .....

  9.52  57    878   2.96  OK        A      gnosys.svle.ma.us
        iugate.ucs.indiana.edu.(142), itdsrv1.ul.ie.(134),
        ariel.unm.edu.(74), spot.Colorado.EDU.(60), fslg8.fsl.noaa.gov.(52),
        uneeda.aarnet.EDU.AU.(38), vaxmom.scri.fsu.edu.(37), ..................

  9.11  56    842   2.82  OK        MX     asylum.sf.ca.us
        Argus.Stanford.EDU.(471), gatekeeper.dec.com.(70), brown.edu.(39),
        charon.amdahl.com.(31), usc.edu.(24), MINTAKA.LCS.MIT.EDU.(22),
        lion.pa.dec.com.(21), quern.epilogue.com.(19), MC.LCS.MIT.EDU.(16), ....

  9.10  1     836   2.83  nxdomain  A      momad.li.ny.us
        ferkel.ucsb.edu.(836),

  8.15  3     757   2.52  nxdomain  CNAME  mcnair-ns1.us
        RITCHIE-GIIS.ARMY.MIL.(592), doim1.usma.army.mil.(145),
        doim3.usma.army.mil.(20),

  8.01  2     740   2.48  nxdomain  A      nkosi.well.sf.ca.us.ca.us
        well.sf.ca.us.(735), nkosi.well.sf.ca.us.(5),

  7.81  2     721   2.42  nxdomain  A      well.sf.ca.us.sf.ca.us
        well.sf.ca.us.(717), nkosi.well.sf.ca.us.(4),

  7.78  2     718   2.41  nxdomain  A      nkosi.well.sf.ca.us.sf.ca.us
        well.sf.ca.us.(708), nkosi.well.sf.ca.us.(10),

  7.60  2     703   2.36  nxdomain  A      well.sf.ca.us.ca.us
        well.sf.ca.us.(696), nkosi.well.sf.ca.us.(7),

 10.41  3     641   3.90  nxdomain  A      ca2002.mn.us
        ub.d.umn.edu.(232), nic.state.mn.us.(223), ns.unet.umn.edu.(186),
```

Figure 5: Example Queries/Response Report Page

```
--------------------------------------------------------------------
ERROR TABLE, threshold=50, Tue Jul 20 08:22:44 1993
--------------------------------------------------------------------

192.187.138.1 (rhythm.com.) (server)  [1]
137 queries,    55.02/hour,  97.81 percent with errors  [2]
137 responses,  55.02/hour,  89.78 percent with errors
      Too Many Referrals   123   (93.20 /hour,1.32   hours,Jul 20 00:01)  [3]
      Ignored No Record    134   (97.79 /hour,1.37   hours,Jul 19 23:58)

192.132.30.2 (well.sf.ca.us.) (server)
4188 queries,    45.30/hour,  79.20 percent with errors
4188 responses,  45.30/hour,   0.00 percent with errors
      Ignored No Record    2     (0.02  /hour,86.77  hours,Jul 16 14:25)
      Ignored No Domain    3315  (35.86 /hour,92.45  hours,Jul 16 11:33)

128.111.72.60 (ferkel.ucsb.edu.) (server)
2024 queries,    21.90/hour,  99.70 percent with errors
2024 responses,  21.90/hour,   0.00 percent with errors
      Ignored No Domain    2018  (21.85 /hour,92.37  hours,Jul 16 11:54)

192.136.137.6 (RITCHIE-GIIS.ARMY.MIL.) (server)
604 queries,     6.51/hour,  98.18 percent with errors
604 responses,   6.51/hour,   0.00 percent with errors
      Ignored No Domain    593   (6.40  /hour,92.68  hours,Jul 16 11:37)

36.56.0.151 (Argus.Stanford.EDU.) (server)
643 queries,     6.94/hour,  90.05 percent with errors
643 responses,   6.94/hour,  29.24 percent with errors
      Too Many Referrals   188   (2.08  /hour,90.57  hours,Jul 16 13:04)
      Ignored No Record    560   (6.10  /hour,91.74  hours,Jul 16 12:17)
      Ignored No Domain    19    (0.21  /hour,91.11  hours,Jul 16 11:36)

129.29.130.1 (doim1.usma.army.mil.) (server)
728 queries,     7.86/hour,  75.14 percent with errors
728 responses,   7.86/hour,   0.00 percent with errors
      Ignored No Domain    547   (6.02  /hour,90.93  hours,Jul 16 13:18)

132.239.254.201 (ucsd.edu.) (server)
392 queries,     6.16/hour,  86.22 percent with errors
392 responses,   6.16/hour,  52.04 percent with errors
      Ignored Referrals    1     (            ,0.00   hours,Jul 18 18:02)
      Too Many Referrals   204   (3.48  /hour,58.65  hours,Jul 17 17:06)
      Ignored No Record    334   (5.67  /hour,58.86  hours,Jul 17 16:54)
      Requery Inside TTL   4     (0.17  /hour,23.93  hours,Jul 18 15:13)

162.148.13.13 (eskimo.com.) (server)
538 queries,     5.82/hour,  98.14 percent with errors
538 responses,   5.82/hour,   0.00 percent with errors
      Ignored No Record    5     (3600.00/hour,0.00   hours,Jul 18 06:02)
      Ignored No Domain    523   (5.66  /hour,92.39  hours,Jul 16 11:48)
```

Figure 6: Example Error Report Page

3. The third part of the "mini-report" is a list of the individual error modes seen in the client's packet traffic. Each failure type is shown with its short name, the total number of occurances, which then shown as a per hour rate. For each failure mode the first and latest occurance is recorded, with the first occurance and the time span shown in the report entry. As mentioned above, the "mini-reports" that make up the Error Table Report are sorted by looking at the per hour rate of errors on each client's worst (highest count) failure mode. This can overemphasize a client that had one burst of errors, but this tendency is diminished by making sure the per hour rate is calculated over at least three maintenance intervals, and stretching the time if necessary. Occasionally, as shown in Figure 6, a relatively low count client can appear at the top of the report because it had a high rate. Clients that have high counts and high per hour error rates are the real target of the Checker Project.

3.4.6 Error Mode Explanations

The different errors are listed below, along with their mask position. A full explanation, and the actual tests, are done in `checks.c`. Many comments are included in the code and users are urged to review it.

1. Client Resolver Incorrect Response to Server Failure, (SERVER_FAILURE_A, 0x020)

   This query message check looks for resolver code that does not correctly redirect its queries to a different name server when the first one times out on a reply. By not redirecting its queries the client is continuing to generate traffic that has little expectation of being answered. The "leaky bucket" approach is used to make sure that legitimate retries are not declared as an error.

   SERVER_FAILURE_A is declared when:

   - No response has been seen to the most recent identical query from the client
   - The client is not a name server (not port 53)
   - There have been too many queries from the client without an intervening response (N entries within servfail window amount of time).

2. Client Name Server Incorrect Response to Server Failure, (SERVER_FAILURE_B, 0x040)

   This query message check is the same as SERVER_FAILURE_A, except that the client must be another name server. The distinction between resolver and name server is made because the code implementations are quite different.

   SERVER_FAILURE_B is declared when:

   - No response has been seen to the most recent query of any kind from the client
   - The client is another name server (port 53)
   - There have been too many queries from the client without an intervening response (N entries within servfail window amount of time).

3. Fast Retransmission, (FAST_RX, 0x001)

   This query message detector trips when a client reissues a query more quickly than the recommended timeout period. This can be wasteful because the reply to the first one may actually be on its way back, but the client did not wait long enough to receive it. The fact that the name server will process the unnecessary second query and send a reply to it only compounds the problem.

   FAST_RX is declared when:

   - No server failure errors were declared

- The preceding identical query generated no response from the name server.
- The most recent query happened within NETWORK TIMEOUT of the preceding identical query.

4. Client ignored referral and tried again, (RECURSION_A, 0x002)

   This query message check detects code or situations where the local name server's referral (saying that it doesn't know the answer, but the referral computer might) has been ignored. This is detected by watching for identical queries coming from the same location within a short amount of time.

   RECURSION_A is declared when:

   - The preceding identical query did get a response
   - The response was error free
   - The response had no answer records
   - The response had some referral records
   - The preceding identical query from the same location happened less than NETWORK_TIMEOUT plus DELTA seconds ago.

5. Client ignored an authoritative "no such record", (ZERO_ANSWER, 0x010)

   This query message detector looks for lower level resolvers or name servers that have ignored this name server's reply that the query's domain is good, but the machine name doesn't exist.

   ZERO_ANSWER is declared when:

   - The preceding identical query did get a response.
   - The response was error free.
   - The response had no answer, referral, or data records.
   - The newest query occured within the cache TTL time of the preceding query.

6. Client requeried inside of cache TTL value (CACHE_LEAK, 0x080)

   This query message check looks for clients who's cache is not holding returned answers long enough. A "cache leak" could indicate incorrectly implemented code, or the fact that the client was restarted recently.

   CACHE_LEAK is declared when:

   - The previous identical query did receive a response.
   - The response was error free.
   - The response did have answer or referral records.
   - The newest query has occured within the TTL of the answer to the preceding identical query.

7. Client requeried too often on the same query (RECURSION_B,0x004)

   This response message check uses a "leaky bucket" stack of time stamps to check the number of times the client has called up for the exact same query within a settable length of time. If the client requests too often it may mean that a recursion loop exists.

   RECURSION_B is declared when:

   - The response message shows no errors in the response code.

- The number of responses to this client has exceeded a certain threshold within the allowable time period.

8. Host name server named itself in referral (RECURSION_C,0x008)

   This response message check looks into the response message to make sure that that Checker's host name server did not place its own address into a referral record.

   RECURSION_C is declared when:

   - The response message shows no errors in the response code.
   - The response has no answer records.
   - The response has Checker's host's address in a referral record.

9. Client ignored a "no such domain" response and requeried (NAME_ERROR,0x100)

   This query message check looks for clients that do not accept a "no such domain" response and issue a requery.

   NAME_ERROR is declared when:

   - The client has previously received a reply to this query.
   - The previous response error code was "no such domain".
   - The time between the current query and the previous response is less than a settable "negative cache" timeout.
   - (optional - runtime configuration) The previous response was authoritative.

10. Client's address domain doesn't use central cache (NO_CENTRAL_CACHE,0x200)

    This query message check looks at clients that are part of a common IP address domain and attempts to discover those that do not have a lower level name server to handle that address domain. This check is primarily of interest to those that run root level name servers and are looking for campus sized networks that should have authority for their domain delegated to a more local level to avoid traffic across backbone links.

    NO_CENTRAL_CACHE is declared when:

    - At least one reply to the same query has been returned to an IP addressing domain.
    - This query is being done within the time to live of the previous reply.
    - There are have been more than one clients calling from this IP addressing domain.

11. Client sent a badly formatted query (FORMAT_ERROR,0x400)

    This response message check looks for responses being returned to clients that carry a FORMAT ERROR response code, meaning that the hosting name server could not understand the client's query message.

    FORMAT_ERROR is declared when:

    - The response message has a response code of FORMERR.

3.5 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.

3.5.1 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. These time lengths (`deletion time` and `negative cache`) are 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 rebooted before the database can get very large). Note that `deletion time` is usually set to a high value and `negative cache` set to a small value to make sure that bad query names do not clog the database. It is interesting to note the large effect the `negative cache` value has on the size of the database, illustrating again how much of DNS's traffic is made up of bad queries.

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, can be 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.

3.5.2 Timing

Detailed studies of name server response time with and without Checker are still being done. Analysis of the code suggests that the only time that there is significant processing is when a new query name is seen from a given client IP address. This requires a new record to be entered into the database, which involves allocating some space and threading the record's address into the various linked lists. On most systems this work is done very quickly and doesn't burden the name server.

Another, relatively rarer, occurance is the database scan done once each maintenance interval to purge records that have not been reference within their cache timeout (`deletion time` for valid queries, and `negative cache` for ones with errors in their response code). The length of time between maintenance scans is set in the configuration file. To reduce the load at any given scan the database is divided into N sections (currently 24, set by a compile time parameter) and only 1 of the sections is scanned at a time. Therefore it takes Nx(`maintenance_interval`) to review the whole database. The timing consequences of the database scan are related to the number of query names held in the database. The delay comes on the first packet passed to Checker after the interval expires.

3.5.3 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 /usr/tmp. 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. Memory usage is only written if `memory logging` is turned on, and is written in two ways on every maintenance interval scan through the database. The first is a raw number returned from the `mallinfo()` function call that represents total dynamic memory allocated by the named daemon. The second is a entry the summarizes the time deletion queue slot that was examined in the maintenance scan.
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.

3.5.4 Security

Security concerns are addressed by the use of the `password check` and `password string` variables described in Section 3.3. It is up to the name server operator to decide whether or not to enable password checking, and to enter a non-default password. As delivered the default password is "`checker`".

The consequences of not using passwords 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).

3.5.5 PEC Checking

In its more sophisticated operation mode Checker can be used to randomly "interdict" replies to randomly selected query names. This option requires a slightly more complex change to named's code, not shown in Section 4, to allow the return value from the `checker()` call to determine whether or not a reply packet is issued from named. 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.

This feature is still in the testing stages, with effort being focused on how to interpret the results of a PEC interdiction period. It is thought that a baseline of activity for each client must be established before changes due to PEC interdiction can be meaningful (it does no good to interdict replies that are going to a client that is already misbehaving).

4 BIND changes, without PEC

Note that all of the changes here are described in terms of diff listings between the code with checker() and unpatched code from the original release file obtained from gatekeeper.dec.com or ftp.uu.net. The changed files are supplied with the checker release, but if it is necessary to apply the changes by hand to patched code (i.e. Lewis patches for 4.8.3) then it may be necessary to correlate the code in the changed areas with the patched files to be edited by hand. In that case it is advised that an editor be used to search for "CHECKER" in the supplied files and the enclosed code be copied to the patched files.

4.1 BIND 4.8.3

4.1.1 ns_req.c

Ns_req.c requires changes in three 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:

```
590,592dxxx
< #ifdef CHECKER
<         checker( 1, msg, cp-msg, from, qsp );
< #endif
<
```

4.1.2 ns_resp.c

Ns_resp.c requires two 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:

  ```
  1191,1193dxxxx
  < #ifdef CHECKER
  <         checker( 1, msg, msglen, &qp->q_from, qp->q_stream ) == 1 ) {
  < #endif
  ```

4.1.3 Makefile

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

```
9c9
< DEFINES= -DDEBUG -DSTATS -DCHECKER
---
> DEFINES= -DDEBUG -DSTATS
15c15
< LIBS= ../checker/libckr.a ${RES}
---
> LIBS= ${RES}
```

4.2 BIND 4.9.2 release 930706

This code has not been installed or run, but is very similar to the 4.8.3 version and no operational differences are anticipated.

4.2.1 ns_req.c

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

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

  ```
  116,119d115
  < #ifdef CHECKER
  < extern u_long  checker();
  ```

```
< #endif /* CHECKER */
<
```

- Capture of incoming query packet:

```
180,183d175
< #ifdef CHECKER
<         checker( 0, msg, msglen, from, qsp );
< #endif /* CHECKER */
<
```

- Capture of outgoing response packet:

```
791,795d782
< #ifdef CHECKER
<         checker( 1, msg, cp-msg, from, qsp );
< #endif /* CHECKER */
<
```

4.2.2 ns_resp.c

Ns_resp.c requires two 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:

```
96,98d95
< #ifdef CHECKER
< extern  u_long checker();
< #endif
<
```

- Capture of outgoing response packet:

```
1386,1388d1382
< #ifdef CHECKER
<         checker( 1, msg, msglen, &qp->q_from, qp->q_stream );
< #endif /* CHECKER */
```

4.2.3 Makefile

The Makefile in the named subdirectory must be modified to define CHECKER and to include the libckr.a among the libraries to be linked into named. As of the current Checker release the 4.9 Makefile was not correct. It will be updated here when Bind 4.9 is released for full use on the Internet.

```
TBD
```

5 Future Work

Ongoing work in The Checker Project includes several things mentioned in this document, and enhancements to make `checker()` more accessible or easier to manage. The following paragraphs summarize some of the anticipated work:

- `Error Modes:` Many of the error modes that can be reported seem to be reported in combination with the same complementary set of other error modes. This suggests that some of the detectors have overlapping range, or that some of the error modes are caused by a single software problem. More research is needed to watch the statistics of the error detectors to see if these impressions can be quantified and turned to a useful purpose. A further problem in this area is determining how to set some of the thresholds that trigger an error, particularly the recursion and timing related problems. This includes time deltas as well as trigger depths in the leaky bucket stacks.
- `Timing Studies:` As mentioned earlier, timing studies of Checker's impact on name server operation is still being done. Because there are several processing modes it is somewhat difficult to characterize Checker's time profile. This will continue to be monitored as time permits.
- `Bind 4.9 Testing:` As the BIND 4.9 releases stabilize, Checker's operation with the new releases needs to be verified. It will be interesting to watch the behavior of the Internet in terms of error rates as negative caching (a highly recommended 4.9 option) becomes more common.
- `PEC Checking:` This is an exciting concept that has been tested in USC's laboratory, but with limited success in interpreting the output. Much more work needs to be done to separate the baseline activity of a client from the extra behavior caused by the PEC interdiction.
- `Memory Efficiency:` Until the negative caching time deletion parameter was added it was difficult to control the amount of memory that Checker used for its database. There is still not a well understood relationship between caching times and the amount of memory used. This needs to be investigated and tables or algorithms developed for predicting and controlling the amount of memory that Checker uses.
- `Database Checkpointing:` When the name server is restarted all of its dynamic memory is lost and must be rebuilt over a period of time. Because much of Checker's output is based on historical and statistical data this causes Checker's reports to not be meaningful for several hours to a day. It would be nice if there was a feature where Checker's database could be written to disk and later reloaded on command. This would also permit offline analysis without the constraints of a runtime name server.
- `Report Requests:` Currently the only control request to generate reports triggers all of the reports to be generated and written into the output file. This is overkill and it would be better if there were several options that allowed the user to select the desired report type.
- `Name String Storage:` When a report is requested a child is spawned to handle the request. It is the child that scans the database, sorts the results, and reverse translate the client IP addresses into domain names for the report. There is currently no provision for the inverse translations to be cached for the next report, forcing the program to retranslate things it should already know. The intention is to develop a return data channel from the reporting child to checker() where the translations can be passed for caching in the database.
- `Dig Removal:` The dig program is invoked each time an IP address is translated back into its domain name. Dig's output is written to a disk file and picked up by checker() which parses the contents to get the name. This is seriously inefficient, and drastically slows the report generation. Dig is used in this manner because direct calls to the resolver by a function within the name server itself has caused operational problems, only the least of which is not getting the reverse translation done. The intention is to fix the resolver problem or directly call name server database routines.