SCREEN DNS GUIDE
VERSION 1.0

DATE: JULY 2011

This document is intended for internal use by AvauntGuard personnel only.

AUTHOR: SANDEEP NUCKCHADY

DOCUMENT VERSION 1.4

CONTENTS

INTRODUCTION

The purpose of this guide is to help those interested in understanding the process flow we went through in making screen DNS version 1.0. Targeted users are those with some linux and bsd background and this guide is definitely not the ultimate guide. The reader is encouraged to surf the net for solutions and read the code on svn. This guide does not provide much information on the web-admin and database and readers envied with such knowledge should refer elsewhere.

It's important to point out that there is no single way of actually executing each of the commands presented in this document. So, the reader is welcome to use those commands they are more comfortable with. One last note is that throughout this documentation you would see oft-repeated explanations on the same topic - it's a concern that some of the mistakes might happen again and again.

STRUCTURE

It has been decided to split this guide into six different sections. Section 1 provides an introduction to what constitute the screen DNS version 1.0. The daily tasks that need to be carried out are listed in Section 2. Efforts have been made to provide the different steps needed to (re)start all of the scripts related to the dns server in Section 3. The purpose of Section 4 is to compare the different implementations of looking up the domain names. Section 5 lists the different discussions we had in the making of screen DNS version 1.0. We close this document or report by providing the debatable requirements for the next version of screen DNS.

1. SCREEN DNS OVERVIEW

Three main processes make up the main DNS server. The first one, `MainDNSServer`, is the main dns engine processing requests and responses. Each domain from each request is compared with entries in the cdb file created by the process `Monitor`. This second process, `Monitor` will connect to the MySQL database and select the domain names and unified responses which will be the first two columns of all the cdb files (`whitelist1/2.cdb`, `malware1/2.cdb`, `child1/2.cdb`) with the exception of `IP1/2.cdb` (no unified response; directly category as second column). Depending on the port selected, the requests might also be suffixed with rbls. The output will determine whether the domain names need to be blocked and, if so, a modified response with the blocked page IP of `202.123.3.118` is returned to the client. Any relevant information pertinent to the blocked domain will be sent to the database in the table `blockeddomainstats` via `Re`, the third process waiting for incoming data through a pipe. Machines with IP `192.168.231.50` and `192.168.231.51` have these binaries not necessarily running permanently. All these relevant processes have been shoehorned into one single diagram, Figure 1. The reader is referred to **Appendix E** to get more information about the different IPs and hostnames.

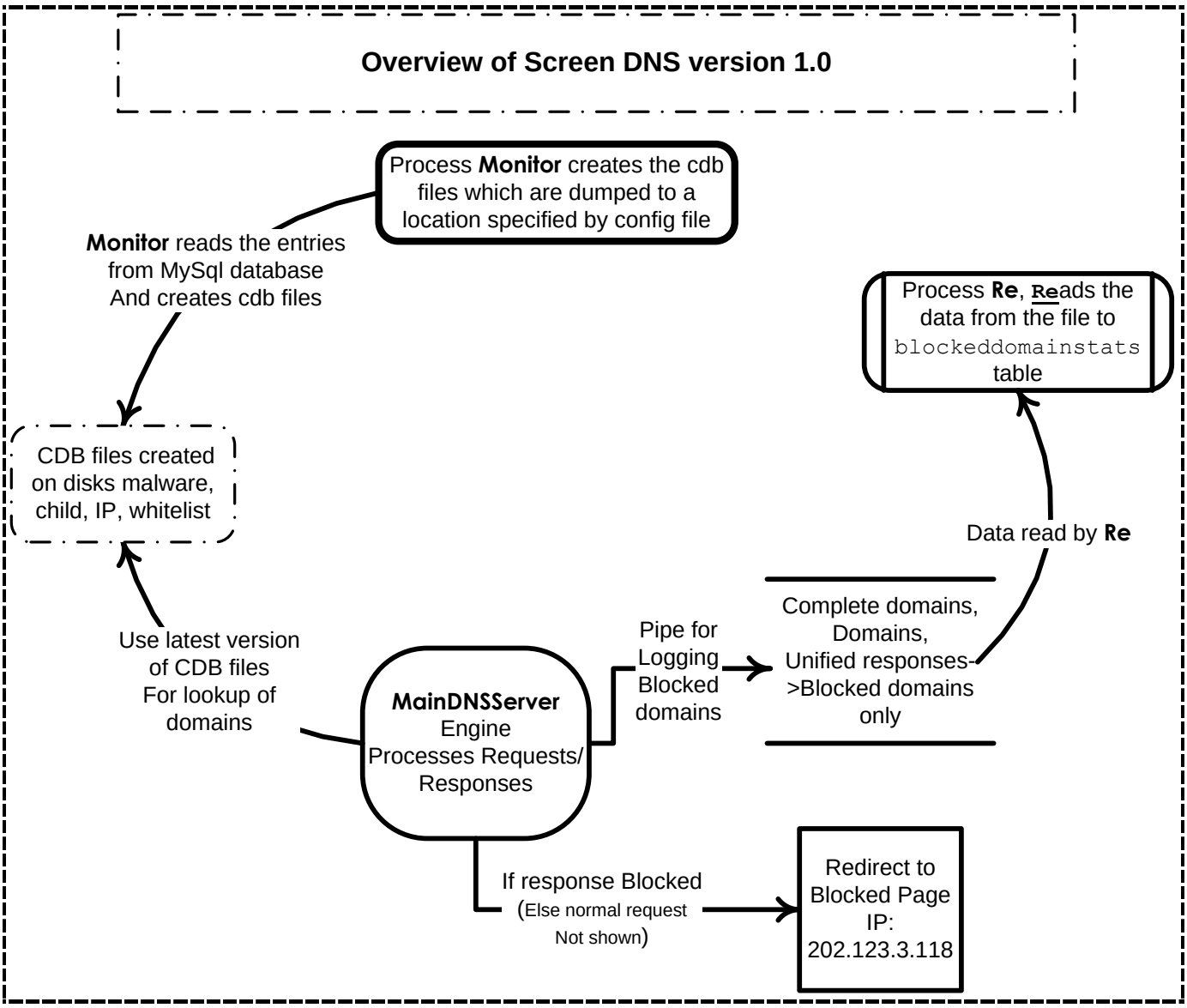

Figure 1 An overview of the different processes of the DNS server

The main features of **ScreenDNSv1.0** are:

1. Listening on three different ports as follows:
   Port 53 -> parses RBL + Malware/Phishing + Child Abuse
   Port 20053 -> Malware/Phishing + Child Abuse
   Port 30053 -> Child Abuse lookup only
2. Each rbl lookup has its own thread + main request goes in a separate thread
3. Each incoming request is processed in a separate thread
4. Look up of domains have been done in three ways: directly to database, using memory and using cdb databases, the latest being the latter. The others can be retrieved from svn
5. CName and IP lookup upon receipts of some DNS responses back
6. Database look up of sub-domains and domains

7. The fully qualified domain names (e.g. `www.foe.co.uk.multi.surbl.org`) are sent for verification by the RBL list; not sub-domains
8. Three config files for the following: logging to the database (Process `Re`) , creating the cdb files (process `Monitor`) and processing the requests/responses (process `MainDNSserver`)
9. Logging to `blockeddomainstats` table uses a pipe mechanism. The pipe will concatenate the whole domain string together with the separator "(" . According to *rfc 1035* the "(" symbol is an illegal character in any domain name
10. All the reasons from both databases and rbls are logged irrespective of the time each one takes
11. Automated script (`/usr/local/etc/rc.d/secdns_startup.sh`) should restart the `MainDNSserver` and the process `Re`
12. No need to stop the `MainDNSserver` process when updating the databases. Use the process `Monitor` to update the cdb files based on the changes in the MySQL database
13. Logging to the database of the following with a specific format: complete domain and domain and unified response/category
14. Logging framework in place but not being used as of this version

2. DAILY TASKS

The daily tasks consist of four different steps, all four could possibly be done by Nagios and the fourth. The list is as follows:

2.1 VERIFY THE THREE MAINDNSSERVER PROCESSES

Nagios could monitor this.
Verify that three instances of `MainDNSserver` are running on `192.168.231.50` and `192.168.231.51`. If any one of them has stopped refer to **Steps to Start "`MainDNSserver`" and "`Re`" scripts** in the **Restarting Scripts** section below to restart them.

2.2 VERIFY WHETHER RE IS RUNNING IN THE MACHINES WITH IPS 192.168.231.50 AND 192.168.231.51

Nagios could monitor this.
If the process `Re` has stopped running refer to **Steps to Start "`Re`" script** section below.

2.3 VERIFY THAT MODDNSDUMPV1.0.PL IS RUNNING ON THE MACHINE WITH IP 192.168.231.17 AND 192.168.231.34

Nagios could monitor this.
If the process `moddnsdumpv1.0.pl` is not active refer to **Steps to start the `modDNSdump.pl`** section below.

2.4 VERIFY LOGDOMAINS TABLE

Need to verify that "`logdomains||date`" is created every day. The earlier this is done in the morning the better the accuracy of the daily statistics which would be generated on the following day. A valid table name would be something like this `logdomains20110811`. If such a table is missing, go to "**Steps to start the modDNSdumpv1.0.pl"** section below.
Important note:
A missing table does not mean that `modDNSdumpv1.0.pl` is not running. A table is only created if both of these conditions are met: *A new day has been detected by the time on the local machine* && *a request is being sent to* `192.168.231.17` (an example ip). So, just to be sure, if you do not find a table for the current day and `modDNSdumpv1.0.pl` script is active, just send a request using dig to e.g. `192.168.231.17`. For instance, `dig @192.168.231.17 www.foe.co.uk` assuming this domain still exist.

3. STARTING SCRIPTS

The different scripts that might need to be restarted are listed in this section. The user is required to not copy and paste the examples provided here. Examples are just supposed to help.

3.1 STEPS TO START "MAINDNSSERVER" AND "RE" SCRIPTS

As of time of this writing the main servers processing the DNS requests are `192.168.231.50` and `192.168.231.51`.

Choose one of the three Options

Option 1

3.1.1 Restart the machine

The binary script, `MainDNSServer`, restarts automatically on the server `192.168.231.50` and `192.168.231.51` (each a VM machine with physical memory 2GB, OS – FreeBSD).

3.1.1.1 Once the machine boots up verify there are **three** instances of `MainDNSserver` and one instance of `Re` running (use `top` or `ps` or anything else you prefer). More than one instance of `Re` means that you have restarted it several times either using the `startup.sh` scripts or restarting it directly.

3.1.1.2 Verify that both DNS1 and DNS2 are processing requests, use `dig @<ipaddress> <domain name> -p <port number> e.g. dig @192.168.231.50 bing.com –p 20053`. The ip is of the machine that you have just restarted. In our case it is `192.168.231.50`. Do the test on each of the different ports i.e. `53`, `20053` and `30053`. If you do not do this, if Nagios is properly configured the red flags should signal you that something is wrong but this might not be done instantly.

3.1.1.3 The requests should be sent to each of the 3 different ports i.e.

```
dig @<machine rebooted> <request> -p 53
dig @<machine rebooted> <request> -p 20053
dig @<machine rebooted> <request> -p 30053
```

3.1.1.4 Do the same for the second DNS server i.e. `192.168.231.51` in our case.

3.1.1.5 Verify whether a domain that's found in the database is being blocked. E.g. `dig @192.168.231.51 topupdates.ru –p <portnumber>` should return a blocked IP assuming `topupdates.ru` is found in the database

Option 2

3.1.2 Manual Restarting

3.1.2.1 Start a new `screen`

3.1.2.2 Verify if the following processes are running: **Three** instances of **`MainDNSserver`** and **`Re`**

3.1.2.3 If at least one of the 3 `MainDNSserver` processes is missing e.g. 2 instances of `MainDNSserver` are only present, kill all of them + terminate the process `Re`. If only `Re` is missing go to **Steps to Start and "`Re`" scripts**

3.1.2.4 Go to the directory: `/usr/local/sbin/DNS`

3.1.2.5 Start the script `StartDNS.sh` as **root**. E.g. `sudo sh ./StartDNS.sh`

3.1.2.6 Go back to step 3.1.2.2. If failed go to method 3.1.1 (**Restart the machine**) above.

3.1.2.7 Verify that both DNS1 and DNS2 (if you have performed these steps on both) are working, use `dig @<ipaddress> <domain name> -p <port number> e.g. dig @192.168.231.50 yahoo.co.uk`

3.1.2.8 Verify whether a domain that's found in the database is being blocked. E.g. `dig @192.168.231.51 topupdates.ru` should return a blocked IP assuming `topupdates.ru` is found in the database

Option 3

3.1.3. The usual way…

3.1.3.1 Start a new `screen`

3.1.3.2 Verify if the following processes are running: **Three** instances of **`MainDNSserver`** and **`Re`**

3.1.3.3 If at least one of the 3 **`MainDNSserver`** processes is missing e.g. 2 instances of **`MainDNSserver`** are only present, then kill all of them + terminate the process `Re`. If only `Re` is missing, go to Section **Steps to Start and "`Re`" scripts** below

3.1.3.4 Go to **`/usr/local/etc/rc.d`**

3.1.3.5 Execute the bash script **`secdns_start.sh`** E.g. **`secdns_start.sh start`** probably as root

3.1.3.6 As you would figure out after some time you do not need to use `screen`

Option 4

3.1.4 Do not use `screen` in **Option 2** or **Option 3** above but run the commands as a background process by putting ampersand at the end

- *The above will cause one application of **`MainDNSserver`** to spawn two more processes which will listen on three different ports separately: 53, 20053 and 30053.*
- *If you need to start the two separate processes **`MainDNSserver`** and **`Re`** individually just read the 2 lines in **`StartDNS.sh`**. The important point is that **`Re`** should always be started first.*
- *If **`Re`** stops working then nothing will be logged to the table and even when you do restart it you will not be able to retrieve the blocked domains sent to the pipe in that interval of time that **`MainDNSserver`** was running.*
- *If a blocked domain is detected the ip returned should be 202.123.3.118*
- *The rbl list is currently hardcoded as follows: "`.rhsbl.ahbl.org`", "`.dnsbl.sorbs.net`", "`.multi.surbl.org`"*

3.2 STEPS TO START "Re" SCRIPT

This is an overemphasis of the previous section which already has explained how to restart the `Re` script.

3.2.1 Start `screen`.

3.2.2 The format would be something like this: `<path to Re> <path of PipeConf.yaml>`. Look into the `StartDNS.sh` script for more information (`/usr/local/sbin/DNS`).
`E.g. /usr/local/sbin/DNS/Re /usr/local/etc/PipeConf.yaml`

3.2.3 Detach from the `screen`. Do not exit.

3.3 STEPS FOR UPDATING THE MAINDNSSERVER WITH THE CHANGES DONE ON THE MYSQL DATABASE 192.168.231.56

3.3.1 Verify that three instances of **`MainDNSserver`** and **`Re`** are present. If absent you will need to refer to the following section: **Steps to Start "`MainDNSserver`" and "`Re`" scripts.**

3.3.2 **SKIP STEPS 3.3.3-3.3.6** if you already have two instances of the same file (e.g. `malware1.cdb` and malware2.cdb). Here is the reason with a real life example:

- One file, `malware1.cdb`, is present. You decide to carry on with ***step 3.3.3 below.***
- Two files, `malware1.cdb` and `malware2.cdb`, are now present in the same folder.
- After some time, you are told to recreate `malware1.cdb` or `malware2.cdb` <u>because new updates have been done to the MySql database.</u>
- You go to the same folder as above and see `malware1.cdb` and `malware2.cdb`. You decide to run the `Monitor` program again as explained in step 3.3.3 below. <u>This will not work.</u> `Monitor` will not recreate a new malware file because already 2 instances of the same file are present. You will have to wait when only one file is present. You are not allowed to manually delete the `malware1.cdb` or `malware2.cdb` files because they might still be used by the `MainDNSserver`. So, you will have to wait for one of them to be deleted.
- If it so happens you accidentally performed the above, you should remember that none of the `malware?.cdb` files will have the latest changes on the MySql database. This implies you will have to run Step 3 later on.

3.3.3 Run the **`Monitor`** binary file specifying the config file. This is done as follows:

3.3.3.1 Always verify that the database information found in the config file:

`/usr/local/etc/configDNS/MonConf.yaml` is up-to-date.

What you will need to check are the following: IP, database name, password, user accounts and the path where the CDB files are located. Verify that **`/usr/local/etc/configDNS/MonConf.yaml`** especially the path to the cdb (a parameter in the `MonConf.yaml` config file) is where the **`MainDNSserver`** is opening the latest cdb files.

3.3.3.2 Type: **/usr/local/sbin/DNS/Monitor /usr/local/etc/configDNS/MonConf.yaml**

3.3.4 Use the **CheckMonitor** command to verify that the database has been created properly and that domains that should be there are really there. E.g.:

<path to CheckMonitor> <cdb-database path> <domain name>

E.g `/usr/local/sbin/DNS/CheckMonitor whitelist1.cdb <non-blocked domain>`
E.g `/usr/local/sbin/DNS/CheckMonitor malware2.cdb <blocked domain>`
E.g `/usr/local/sbin/DNS/CheckMonitor IP1.cdb <blocked_ips>`
E.g `/usr/local/sbin/DNS/CheckMonitor child2.cdb <blocked domain>`

3.3.7 Always Verify that three `MainDNSserver` processes and `Re` are running.

If you get the following output then you know the database has been successfully created **:**

```
Domain <domainName> has been found in <cdb file> with unified Response: <unified response>
```

Depending on the amount of unified responses found in the database you will get the corresponding number of output lines. E.g. 2 unified responses for the same domain will result in 2 of the following line:

```
Domain <domainName> has been found in <cdb file> with unified Response: <unified response>
```

```
Domain <domainName> has been found in <cdb file> with unified Response: <unified response>
```

3.4 STEPS TO START THE MODDNSDUMPV1.0.PL:

The **modDNSdumpv1.0.pl** is found in the machines with ip 192.168.231.17 and 192.168.231.34. Refer to Appendix E for information about the IPs.

Choose any one from the two below:

Option 1:

3.4.1 Use screen

3.4.1.1 Create a new `screen`

3.4.1.2 Verify that a process with the name **modDnsdumpv1.0.pl** is running. If so, skip the steps below unless you are convinced that it's not working as it should. If so, terminate it. A missing table does not necessarily mean that **modDNSdumpv1.0.pl** is not working properly.

3.4.1.3 Verify that **modDnsdumpv1.0.pl** is found within: `/usr/local/sbin`

3.4.1.4 Type the following: `modDnsdumpv1.0.pl –i <interface> "%src %dst %id %question %ans"`

E.g.

```
modDnsdumpv1.0.pl -i em0 "%src %dst %id %question %ans"
```

You most probably need to be a root user.

Option 2:

3.4.2 **Restart the machine.**

3.4.2.1 Verify that a process with the name **`modDnsdumpv1.0.pl`** is running. If so, skip the steps below unless you are convinced that it's not working as it should. If so, terminate it. A missing table does not necessarily mean that `modDNSdumpv1.0.pl` is not working properly.

3.4.2.2 Restart the **relevant machines (IP: `192.168.231.17` and `192.168.231.34`)**

4. SCREEN DNS CONTROVERSY

One of the main discussions we had was about having the main DNS engine being able to determine efficiently if a domain is found in the MySQL database. Table 1 shows the comparison of the different database lookup methods, all of which were implemented, the latest being creation of cdb files.

We are now creating the cdb databases consisting of 2 columns (domains and unified responses / categories) based on each of the different tables (malware, child, etc). Before when we were starting the **`MainDNSserver`** process we had to wait until loading to memory was complete. This time we have separated the two codes to improve the stability of the **`MainDNSserver`** code. The process **`Monitor`** is the one that will create the list of databases with extension `.cdb`.

Table 1: Comparison of the different methods to look up domains

Query DB directly	Query DB From memory	Query From CDB
too many connections might be an issue	No IO reads and writes **Better**	IO read and writes. **Worse**
No Need to load to memory	Loading to memory at orlean varies: few seconds to few minutes **Similar**	Seems to be similar to loading to memory **Similar**
No need to wait to load to memory	Starting of service -> has to wait for loading to memory **Worse**	If database file already present can start service directly without having to wait **Better**
Should be fast if the query is cached by the mysql server	Reading lookup from memory about 20 - 60 microseconds **Similar**	Currently using mmap and speed is comparatively similar to looking directly to memory. **Similar** However, should change that to looking to disk. Initial test showed that there is no performance issue.
No Impact if restart machines	If machines: `192.168.231.17`, `192.168.231.34` and `192.168.231.56` are restarted at Orlean: probably one of them e.g. `192.168.231.17` will start without loading database because cannot connect to `192.168.231.56` (already happened in the past) **Worse**	Latest files from database already present on disk so no problem and main program will start using these files **Better**
n/a	Memory usage higher	More free memory

Worse	Better
Number of requests: `n`	`>> n` due to less use of memory
Worse	**Better**
Merged creation and reading of lookups might cause more code instability.	Segregation of creating lookup from the main code: Maybe increased stability. If creation of the db file crashed the main code should not crash
Worse	**Better**
CPU idle 90% of time	Idle time should be decreased more use of CPU due to less memory usage
Better	**Worse**
Automated insertion and deletion from memory will require a lock on that part of memory. Lots of changes to database -> increase locking time -> increase sending response to client	Expect to need to lock file but there might be more hacks around it and more flexible to achieve
Unknown	**Unknown**

Table 2 is a summary of **Table 1**.

Table 2: Comparison Matrix. A summary of **Table 1. +** means advantage and **–** disadvantage n/a neutral or unknown

	Query DB Directly	Query DB From Memory	Query CBD files
Creation of Connections	-	+	+
I/O read writes	+	+	-
Loading to memory	+	-	-
Waiting time of restarting Service	+	-	+
Reading Queries	n/a	n/a	n/a
Impact of restarting relevant machines	+	-	+
Memory Usage	n/a	-	+/-
Number of requests	n/a	-	+
If DB is not available when `MainDNSserver` is running	-	+	+
Stability	-	-	+
CPU Usage	n/a	+	-
Automated updating of database	+	n/a	n/a

4.1 AUTOMATE LOOK UP OF DOMAIN NAMES

Different suggestions have been made. Currently, it's possible and straightforward to use the current implementation of cdb files to make the `MainDNSserver` code automatically look up the latest changes in the cdb files. The only thing that's needed is a table listing the latest changes done to the MySQL databases and some changes to the Monitor process.

Alternative proposed solutions are:

- Automatic making changes to the memory object if we are doing look up in memory
- Use `inotify` to monitor changes. A very good solution but if only a single change needs to be monitored then just creating a file and verifying its existence (making sure that duplicates not possible via a lock) when these changes occur might be more efficient. A link to http://linux.die.net/man/7/inotify but this is for linux machines and the developer is required to verify if this can be done on bsd machine.

5. DISCUSSIONS UPDATE SCREENDNS V1.0

This section lists the questions and on-going discussions we had in the past.

5.1 TABLE LOGDOMAINS*** IS NOT CREATED DAILY

If you lose connection to the mysql database (`192.168.231.56`) at Orlean, `modDNSdumpv1.0.pl` will throw an exception (print to console if you are running in `screen`) and not continue trying to connect causing it to stop working and no more tables will be created. It has been decided to remove that exception which means that we are "forcing" the program to continue running even though connection failed. This is not good and we should buffer the incoming requests/responses and populate the `logdomains` table whenever connection is re-established. You will need to restart `modDNSdumpv1.0.pl` or rewrite part of the perl scripts to prevent this from happening. A suggestion was made to log to memory when we lose connection to the database and then when we get it back we dump the logs from memory to the database.

5.2 WHY ARE WE RUNNING OUT OF FREE MEMORY?

Nagios should detect if `192.168.231.50` and `192.168.231.51` are running out of memory. If the '*free*' memory is very low that does not mean the program leaks; you will need to add the amount of inactive memory. If the '*a*ctive' memory from '`top`' command keeps increasing, it's possible that some memory leak is occurring. If this happens the immediate solution would be to restart the server if too much swapping is being performed by the OS. Remember that if you have lots of request the free memory is bound to be low due to the amount of threads being created. If that's not the case, then maybe, the threads are not cleaning up properly.

5.3 A PAGE THAT SHOULD BE BLOCKED IS NOT BLOCKED?

You will need to first verify that the main servers with no cache enabled - `192.168.231.50` and `192.168.231.51` are indeed returning a blocked Page IP - `202.123.3.118` or `192.168.231.16`. Most probably the cache on `192.168.231.17` or/and `192.168.231.34` has/have the non-blocked page.

Reasons why the cache version got the non-blocked page:

1. Most probably `192.168.231.50` and `192.168.231.51` took more than 2 seconds to return a reply so NS1 or NS2 took over, thus, poisoning the cache on `192.168.231.17` and `192.168.231.34` with the non-blocked page.
2. If a domain name is found to be blocked by the "RBL list", the blocked page will only be displayed on the second attempt when the cache received the blocked page and not the first time.
3. CDB files are absent.
4. The cache of the browser or somewhere else is returning you the non-blocked page. Try removing the browser cache and do `ipconfig /flushdns` in the command line and then retry. If this failed, run the command prompt as administrator in windows and try cleaning the dns cache.

5. Make sure that the status field is NOERROR when you dig `192.168.231.17` or `192.168.231.34` The dig command for windows can be found: http://members.shaw.ca/nicholas.fong/dig/
6. Another guess is that it might be that the website to be blocked has been determined by the rbl and not by our local database. A blocked response is sent before the normal query is sent. The page you receive is the blocked page. However, since in this version the normal response is also sent, this might overwrite the cache of the blocked response causing the site to be not blocked on the next attempt if the timeout on the cache is still valid.
7. Just use *wireshark* or *tcpdump* to verify the path of responses and queries. Perhaps change your DNS to `192.168.231.17` and `192.168.231.34` or even `192.168.231.50` and `192.168.231.51`

5.4 MAINDNSSERVER RUNNING AND PORTS ARE LISTENING BUT CANNOT RESOLVE ANY PAGE USING 192.168.231.50 OR 192.168.231.51?

Maybe you are receiving mismatched IDs i.e. the transaction IDs of the response and the queries do not match. Perhaps, the stack or heap limit has been reached and cannot handle the incoming responses giving you an ID of 0.

5.5 A PAGE THAT SHOULD NOT BE BLOCKED IS BLOCKED?

First directly send the 'unblocked' query to `192.168.231.50` or `192.168.231.51`. Is it blocked? If not then it's a cache problem. But why is the cache being poisoned. It must have received the response from the `192.168.231.50` or `192.168.231.51` which means that this domain had been blocked at some point. Sometimes it depends on whether the main page is being redirected to another website, then the main query will be the new redirected domain which means that this domain if in the database will be blocked.

5.6 MAINDNSSERVER STOPS LISTENING ON A PORT.

This bug could be caused due to many reasons: Too many threads running at the same time, insufficient memory upon crashing causes the process to crash thus releasing back some of the memory, heap or stack overflow, unintentional destruction of threads, etc. It could also be caused if you have run the `Monitor` program and the cdb files have not been created properly.

To solve the above either run the program in gdb and read the core file or restart the process and in the meantime perform the latter.

5.7 SITE IS NOT FOUND IN THE DATABASE BUT DOMAIN NAME IS BEING BLOCKED.

1. First verify that you are looking in the right database Orlean or Palma.
2. It might be that one of the sub-domains is being blocked and not necessarily the domain name.
3. It might be that the CName is being blocked and not the domain name.
4. It might be that the block is from the RBL. Not all the RBLs are loaded. Those that are: "`rhsbl.ahbl.org`", "`.dnsbl.sorbs.net`", "`.multi.surbl.org`"

5.8 DO I NEED TO RESTART THE MAINDNSSERVER TO RELOAD THE MEMORY?

No. We do not use memory in the current implementation. We were but not anymore. If you want to revert to using memory then just do svn merge and use the previous code. It's a simple algorithm: just use a C++ implementation of 2 columns and dump the MySQL database (domains and unified response) into that.

The no-memory solution has not been implemented completely but a workable implementation is there. Basically we are using *mmap* which we should not. For now, that would do. There is no need to restart the service to parse the domain names and IPs. What you need to do is to create the cdb files i.e. `malware1.cdb / malware2.cdb, whitelist1.cdb / whitelist2.cdb, child1.cdb / child2.cdb, IP1.cdb / IP2.cdb.` To run the program just call the function `Monitor` from within the DNS directory. Do not delete any file unless you are sure about what you are doing. Deleting a cdb file that should not would suddenly cause the domains that should be blocked be unblocked. Files that need to be deleted will do so automatically and the user should not intervene. To sum up, after some time (usually in 20 minutes since creation of the new cdb files) you should see a single instance of each of the cdb files i.e. `malware1.cdb` or `malware2.cdb` but not both. Note that this will only happen if `MainDNSserver` is receiving requests.

What might happen? A repeat of the previous explanation.

One file `malware1.cdb` is present. You run `<path to Monitor> MonConf.yaml`

Two files `malware1.cdb` and `malware2.cdb` are now present in the same folder.

After some time, you are told to recreate `malware1.cdb` or `malware2.cdb` because new updates have been done to the MySql database.

You go to the same folder as above and see `malware1.cdb` and `malware2.cdb`. You decide to run *Monitor* again. This is a futile attempt to update the cdb database. `Monitor` will not recreate the new malware file because already 2 instances of the same file are present. You will have to wait when only one file is present.

5.9 DO WE HAVE CONFIG FILES?

Yes. They are found at : `/usr/local/etc/configDNS`

Table 3 shows the list of config files and their corresponding binary files.

Table 3: The 3 config files of the 3 main processes

Binary Files	Config Files	Contents of Config file
`MainDNSserver`	`config.yaml`	Listening Ports, IP of the local machine, Nameserver IP and port number, Database information to login to the database, Blocked Page IP,

		Time to live + Path to read the cdb files
Monitor	MonConf.yaml	Database information to login to the database only excluding table names etc + path to dump the cdb files
Re	PipeConf.yaml	Database information to login to the database only excluding table names etc

All of config parameters in config.yaml are self explanatory and should not cause any problem except perhaps the TTL field. This needs to be converted to 4 separate bytes i.e. 1 hr = 3600sec = 0.0.14.16 which is a simple conversion from integer to binary.

5.10 BLOCKED PAGE REASON NOT BEING SHOWN WHEN CLICKING ON MORE INFORMATION:

Possible reasons are:

1. Not logging the domain name at all.
2. Domain name is not being logged as an exact replicate as what has been typed in the address bar to the blockdomainstats table. E.g. CNames of the domain names are logged but not the actual domain name.
3. No unified responses for IPs so no display of further information.
4. It might be that logging of the blocked domains to the database is not sufficient fast or logging failed.
5. Looking for the unified response in the wrong table on the website side.

5.11 THE SOURCE IP IN BLOCKEDDOMAINSTATS TABLE IS "DISABLED". WHY?

Source IP will be the IP of 192.168.231.17 and 192.168.231.34 which are the name servers and not the IP of the clients which that field was meant to be initially. Since that's not the case now that field had been disabled.

5.12 HOW DO I COMPILE THE CODE

You will need to do svn checkout of NXTGEN_DNS.

Go to the following directory: /NXTGEN_DNS/trunk/DNSServer

Go to the build directory and type cmake .. and make (might need to clean).

The important files listed in the DNSServer folder are given in Table 4.

Table 4: List of files within DNSServer

Files	Descriptions
build	Use to compile the code by doing cmake .. and make
Config	Directory that contains template for the config files
Include	Directory that contains the header files
DNSNameserver.cpp	The main socket and the threads are being created. Main query to the public IP is being done here
MainDNSserver.cpp	The main code that will create three processes and parse the config file
readP.cpp	To read the logging info. from the pipe
ServerLogging.cpp	A logging framework to easily debug the program
monitor.cpp	Code to create the cdb files
DNSstructure.cpp	Main code to do lookup of the CName and IPs from the DNS response + contain the processing of the RBL queries
Tests	Folder used for testing the different functions + moddnsdump.pl script
Config	MonConf.yaml, PipeConf.yaml, config.yaml templates
Restartscript	secdns_startup.sh which should usually be found in /usr/local/etc/rc.d
CMakeLists.txt	Needed to make the DNS code

5.13 LIST THE DEVELOPMENT MACHINES THAT CAN BE USED TO CODE?

They are as follows:

1. 192.168.230.199 -> High Performance FreeBSD machine 40 GB, 2 core, 4 GB, 64 bit
2. 192.168.230.216 -> Medium Performance Linux Machine, 20 GB, 1 core, 1GB
3. Create your own local Machine -> Low Performance. Tests have been done with config.: 5GB, 1 core, 300 MB

Most tests had been done on a low performance machine before using the first two above.

Installed Lib: svn, cmake, libyaml, boostlibrary 1.46.1, mysql client, tinycdb

If you ever need to install boost library the steps will be sort of like this. These are just guidelines. Read the manual.

Install boost library 1.46.1, extract,

```
./bootstrap.sh --with-libraries=serialization,filesystem,thread
```

./bjam install as root

5.14 LIST THE PRODUCTION MACHINES ?

1. 192.168.231.50 -> 2 core and 2 GB Memory FreeBSD machine

2. 192.168.231.51 -> 2 core and 2 GB Memory FreeBSD machine

Both of the machines above have been configured to compile the code similar to the development machines.

5.15 LIST THE TESTING MACHINE ?

192.168.230.198 -> 2 core 2GB FreeBSD 64 bit machine

5.16 WHY IS IT THAT THE UNIFIED RESPONSE CONTAINS A NON-IP ADDRESS?

As of time of this writing the only time that this occurs is when we log blocked IPs. E.g. PSH, backdoor, are some of the examples. The reason of this is because the query is not linking the category table with the IP table + all the categories found in the blocked_ips are not present. This is not important in this version because the redirected site 202.123.3.118 will perform a lookup of the ip address(es) and display the blocked reason. The blocked ips are logged for statistics purposes.

5.17 ARE WE USING THE MALWAREPRODUCTION AND CAIPRODUCTION TABLES?

We are no more using the malwareproduction and the caiproduction tables.

The time to query the whole malware table is now around 30 seconds and the others are insignificant. It's now 3-5 times slower than using the malwareprodution tables directly.

5.18 WHY ARE WE LOGGING BLOCKED IPS DIRECTLY?

MainDNSserver will search for IPs from the DNS responses from the IP.cdb database. The ips are then logged but it seems that it make more sense to log the domain names rather than the IPs (for statistics). In the current version it's the IPs that's being logged with their categories. The blocked website will look from the latest version of the blocked_ips table.

5.19 HOW TO STOP THE MAINDNSSERVER?

<path to secdns_start.sh> stop most probably as root.

5.20 LIST THE STANDARD LOCATIONS OF THE IMPORTANT FILES

Table 5, Table 6, and Table 7 lists the locations of the Binary Files, Config Files and the Scripts respectively.

Table 5: Binary Files' Locations

Standard Binary Files	Usual Location
CheckMonitor	/usr/local/sbin/DNS
MainDNSserver	/usr/local/sbin/DNS
Monitor	/usr/local/sbin/DNS
Re	/usr/local/sbin/DNS
StartDNS.sh	/usr/local/sbin/DNS
modDnsdumpv1.0.pl	/usr/local/sbin/ Only in 192.168.231.17 and 192.168.231.34

Table 6: Config Files' Locations

Standard Config Files	Usual Location
IP1.cdb	/usr/local/etc/configDNS
child1.cdb	/usr/local/etc/configDNS
malware1.cdb	/usr/local/etc/configDNS
whitelist1.cdb	/usr/local/etc/configDNS
MonConf.yaml	/usr/local/etc/configDNS
PipeConf.yaml	/usr/local/etc/configDNS
config.yaml	/usr/local/etc/configDNS

Table 7: Scripts' Locations

Standard Scripts	Usual Location
StartDNS.sh	/usr/local/sbin/DNS
secdns_startup.sh	/usr/local/etc/rc.d

5.21 WHAT IS MODDNSDUMP.PL?

The main code is found at:

http://dns.measurement-factory.com/tools/dnsdump/

The purpose of modDNSdumpv1.0.pl is to sniff the traffic and log all the necessary information to a logdomains||date table. This makes it possible to link the domain name and the IP address of the client.

5.22 HOW TO CONFIGURE A MACHINE TO RUN MODDNSDUMPV1.0.PL?

The machines that can currently run modDNSdumpv1.0.pl are 192.168.230.199, 192.168.231.17 and 192.168.231.34.

You might need to install some of these and maybe more depending on the machine:

Net-DNS-0.66,Net-Packet-3.27, Net-Pcap-0.05, Socket6, Net-Libdnet, Class-Gomor-1.02, libdumbnet-dev on linux or /usr/ports/net/p5-Net-Libdnet on BSD, Net-IPv4Addr-0.10, Net-IPv6Addr-0.10, Bit-Vector-7.1, Carp-Clan-6.04, Storable-2.25, Mysql client.

It's up to the one installing the above to decide whether the above list of perl modules is really the latest versions are available and whether they are necessary.

5.23 BIND ADDRESS ALREADY IN USE MESSAGE

This message i.e. "bind address already in use" might happen if you are trying to re-run MainDNSServer. <path to config file> This causes the process that is already listening on a given port 53, 20053, 30053 to interfere with the new process. You will need to terminate MainDNSServer and also Re if they are the ones listening to the ports.

5.24 DO WE HAVE A LOGGING FRAMEWORK FOR DEBUG

The logging framework has been written using the log4cxx library. I have not tested it though and the relevant class is called `ServerLogging.cpp`. Instead of redirecting the output to `cout` we should use some sort of logging framework either the one that has already been written or something else you prefer.

5.25 WHAT IF THE RBLS LOOKUPS ARE NON-RESPONSIVE

We only wait for 2 seconds to get a reply from the rbl lookup. This mean that the threads will be active for at most 2 seconds and then the thread should clean itself and return to the calling function.

5.26 IMPACT OF SOME OR ALL CDB FILES ABSENT

The present code should not care about the presence of the CDB files. If they are not present `MainDNSserver` would still run.

5.27 HOW TO RELATE THE LOGDOMAINS AND BLOCKEDDOMAINSTATS TABLES

To get the customer IP (found in the field `ans_ip` of the `logdomains` table) you will need to not ignore the IPs listed in Appendix E. What's left would most probably be the customer IP. To uniquely identify a complete request/response of a given customer and a domain you are required to merge similar transaction IDs in a given time interval (field `id` of the `logdomains` table). Remember that transaction ids would repeat after some time. Perhaps, only statistics for domains where the `rcode` field in `logdomains` is NOERROR should be done. The main `logdomains` table should contain the necessary information to create most of the statistics. Information about the rbls are only provided in the `blockeddomainstats` table.

The latest versions of `MainDNSServer` on the different machines are:

IP Address	Binary File	md5	svn version
192.168.231.51	MainDNSServer	c0e533ab474c2d26994899a120a5d797	507
192.168.231.50	MainDNSServer	6fdd97fddad20e4c9e019ec0134cc063	507

6. NEXT VERSION OF SCREENDNS

During the development of screenDNS v1.0 we had some on-going discussions and in this section we would list most of what could be in the next version.

Stability:

1. New Design to cope with new requirements
2. A threading mechanism to control the amount of threads spawned at a given time
3. Method **`ListCNIP`** in **`DNSstructure.cpp`** will have to be rewritten by ourselves. Code taken from http://www.binarytides.com/blog/dns-query-code-in-c-with-winsock-and-linux-sockets/
   The author made a quick one to look only `CName` and `IPs` and this function is called **`QuickTestCNIP.`** It's recommended that it's done similar to the above URL. **`QuickTestCNIP`** cannot read IPv6.

Robustness:

4. DNS should be able to query both DNS Nameservers: ns1 and ns2
   Currently, either `192.168.231.20` or `192.168.231.23` are being queried as identified in the `config.yaml` file

Flexibility

5. Support for extended DNS
6. RBL list should be taken from the database. This has not been implemented because some of the rbls in the table would be allocated to malware in the later stage and the simple way was to just hardcode the necessary rbls in the code – something that will have to be changed later on.
7. If the same domain is found in child and malware then only the one in the malware will be logged (port 53 and port 20053). If this domain is not found the lookup will be done in the child database.
8. We need to wait for some amount of time to make sure that RBL will either returned us a response IP. Currently, just forward the main response from the public world and wait for rbl to send back a response to the cache so that next time this page will be blocked. Two possible situations, the second one being problematic:
   - A request is sent and is being processed by the list of rbls + is sent as normal request to the public IP. The normal response arrives first and the rbl query is supposed to be blocked and arrives later on. Since the first one was received, the latter will be cache. This is fine.
   - A request is sent and is being processed by the list of rbls + is sent as normal request to the public IP. The rbl response ip arrives first and the normal response arrives last. We are redirected to the blocking page but since the normal response is still being processed this might be cached and then the next query to that page will not be blocked. This I haven't seen yet and is just a guess.
9. Decide on whether you will use memory to load, Cassandra or other methodologies to search for malicious domains.

10. Log the domains found in the whitelist.
11. Stop hardcoding the RBLs + add rbls for malware and child.
12. A question has been raised about the necessity to create three different processes for each port
13. Use of nsupdate instead of database. Perhaps something like this:
    server ns1.orlean.
    zone info.screendns.com
    update add jwi.info.screendns.com 86400 IN A 127.20.20
14. Main program should be able to automatically lookup new domain changes to the database
15. Logging using threads, queue, pipe, etc. A possibility that the MySQL database becomes unavailable-> store in memory and then when they are available dump to database again
16. If one process listening on one port dies we should only be able to restart that one process and not have to kill all the other processes listening on the other ports. Is this in conflict with 12 above?
17. Logging Framework should be in place so that bugs can be resolved rapidly
18. When we look up **`blocked_ips`** we also need to look for the subnet
19. SOA for the blocked responses.
20. If the cdb files are absent the **`MainDNSserver`** engine we do not care. A suggestion was made to flag this whenever those files are absent.
21. To automatically update the constant database (cdb) another table with the latest entries should be made. The cdb files cannot be updated once they have been created.
22. Need to log blocked domains found in both malware and child databases. If a domain is found in both malware and child only the one in the malware will be done. **Appendix G** shows how this might be solved
23. Compute the checksum of the crafted response.

APPENDIX A

Requirement Specification of `MainDNSserver` Version 1.0.

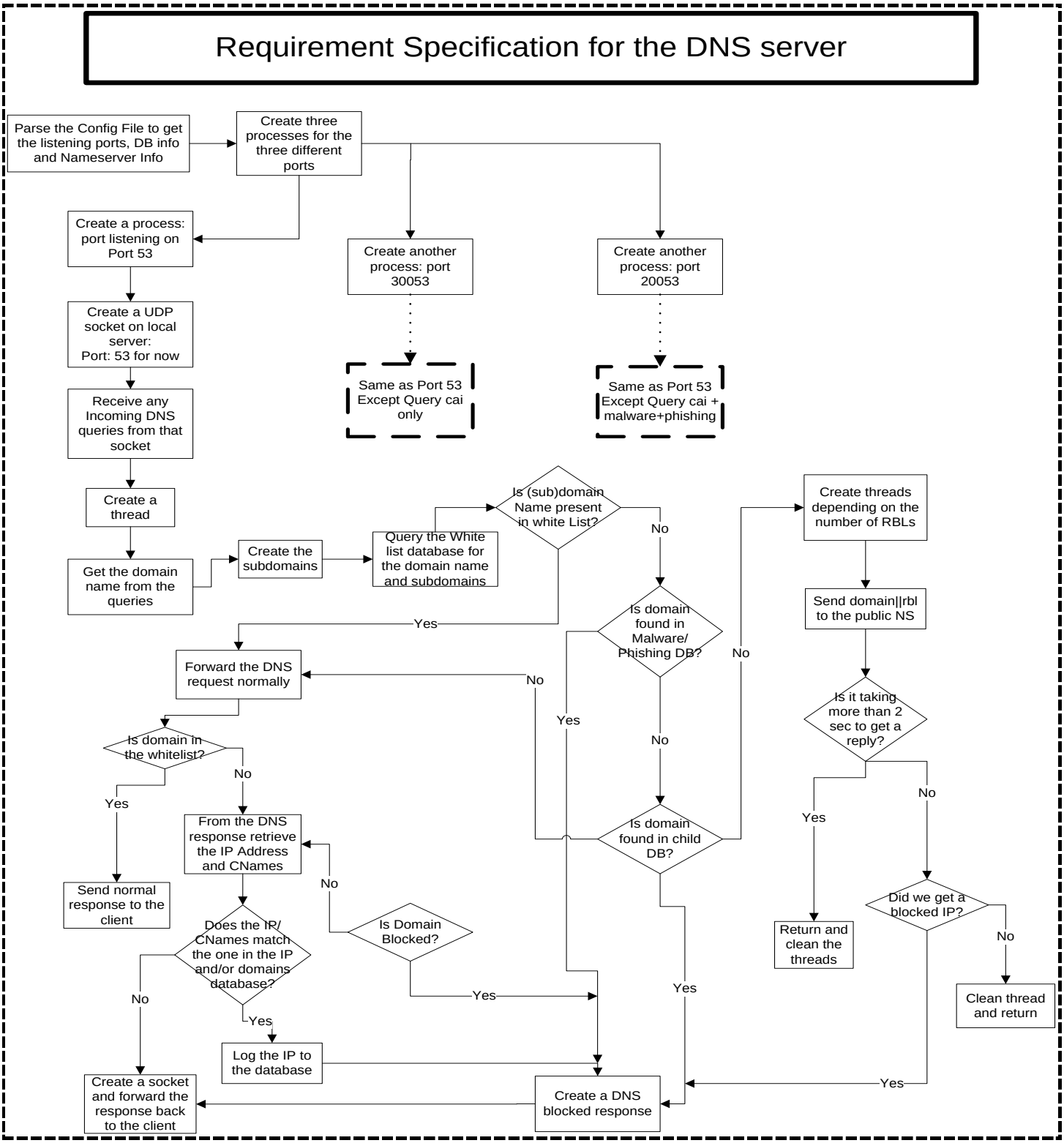

Figure 2: The overall specification of the `MainDNSServer`

APPENDIX B

Writing to the pipe is being done within `MainDNSserver` process. See Figure 3.

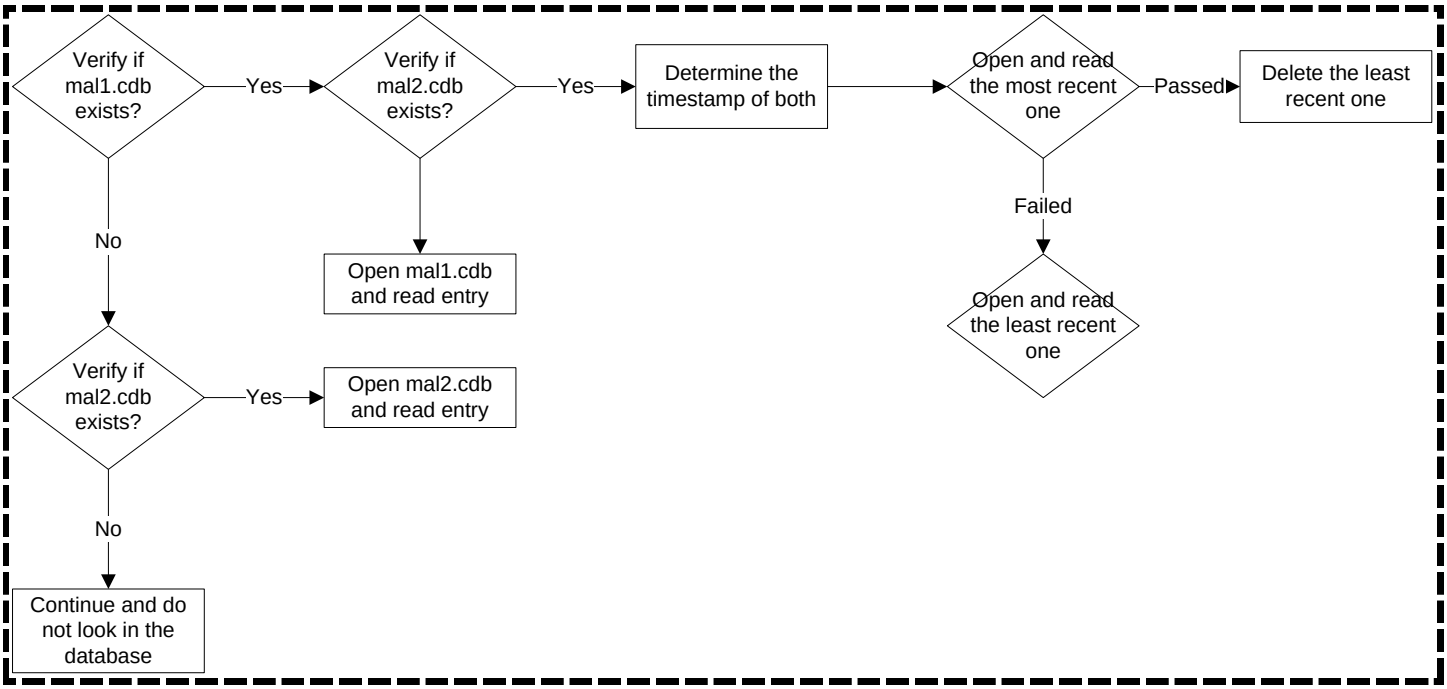

Figure 3: CDB files being read from within the `MainDNSServer`

APPENDIX C

The basic of the `Monitor` process is shown in Figure 4.

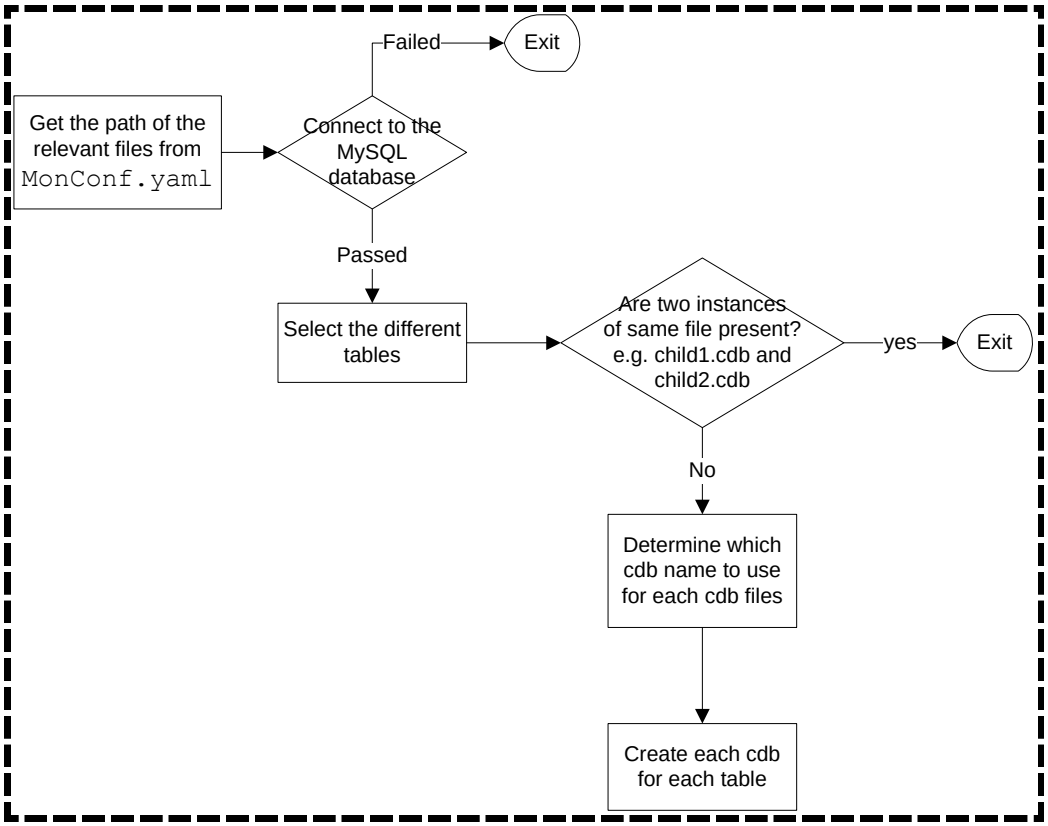

Figure 4: Monitor Process Specification

APPENDIX D

The pipe consists of two parts. `MainDNSServer` will write to a pipe and `Re` will read the data from the other end using the file `BlockedDomain`. This is shown in Figure 5.

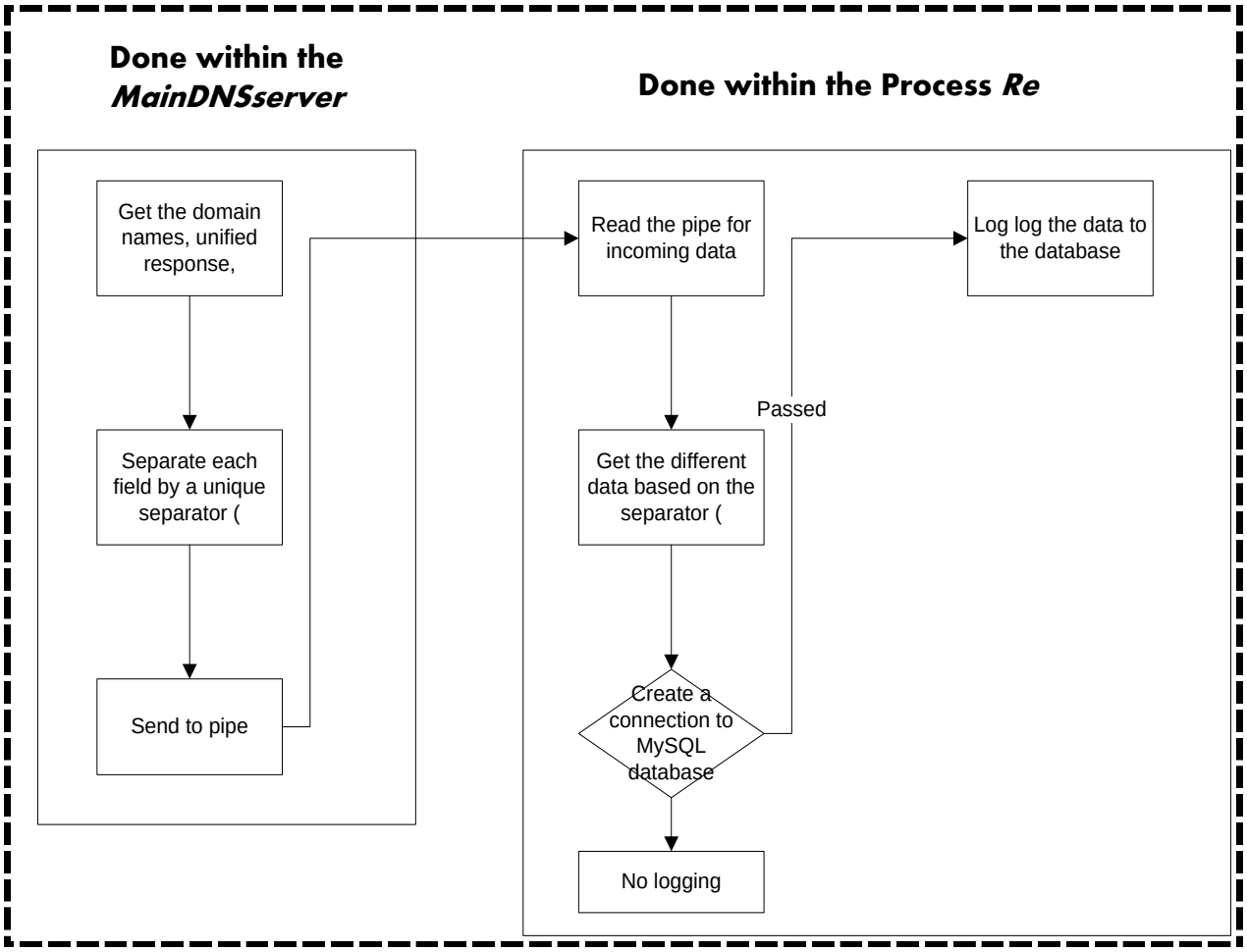

Figure 5: The Re Pipe communicating with the `MainDNSserver` process

APPENDIX E

The flow of a DNS request from the Client is shown in Figure 6.

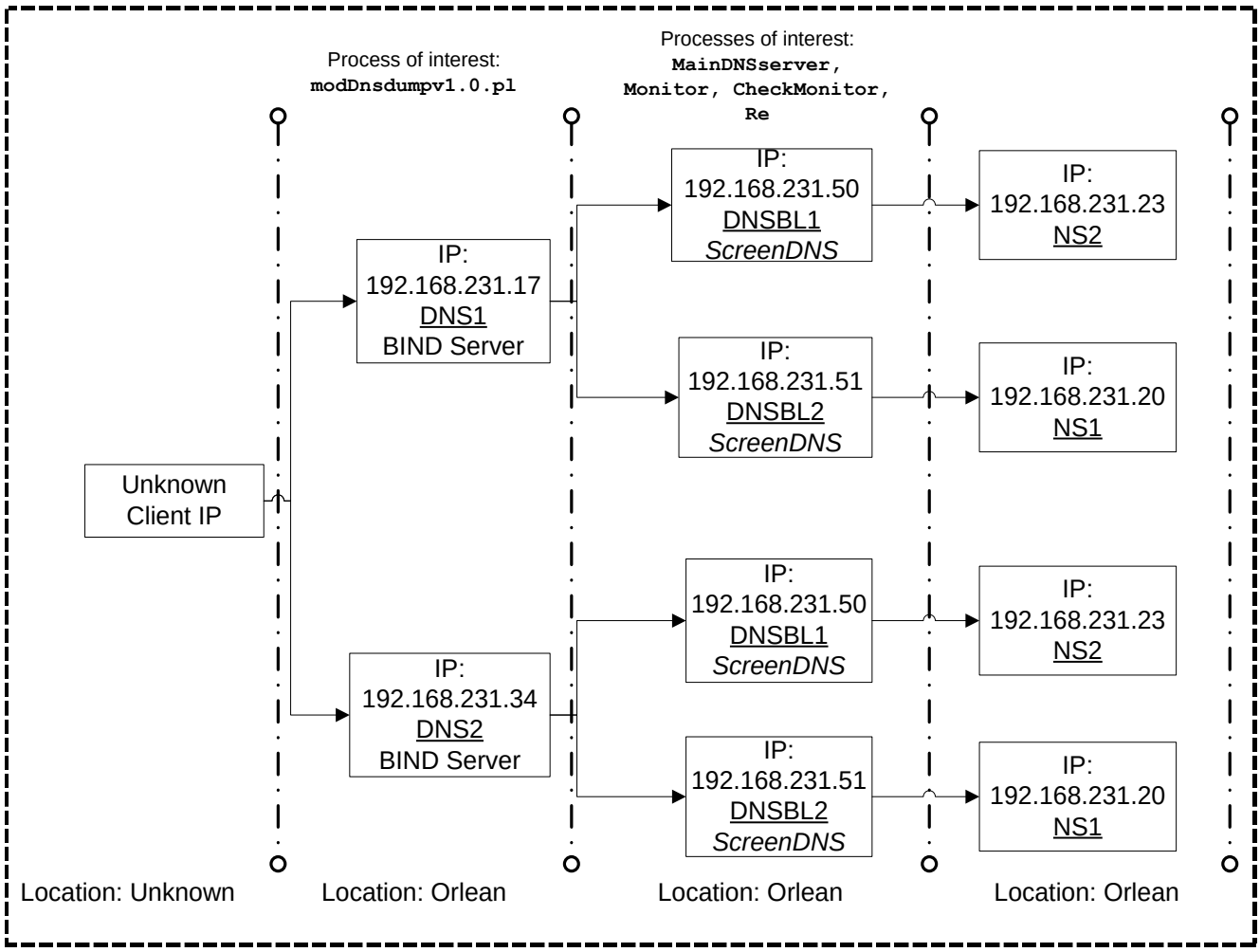

Figure 6: From the Client to DNS1&2 to DNSBL1&2 to NS1&2

APPENDIX F

Task Description of new requirement: Add the ip of the blocked domains to **`blockeddomainstats`**.

Below is an attempt at explaining how to implement it though you are free to choose your own methods.

You will first need to know the different fields within `blockeddomainstats` table:

```
clientip | domain | blockeddomain | createdata | response | zone_name
```

Currently, writing the logs will require a change to the database. So, two possibilities either add a new column or change the name of the `clientip`. If the latter is done, you would need to change the parameter "`Disabled`" in the write log function and insert the new ip from the response of the blocked domain.

Tips:

So far, in the `DNSNameserver.cpp` class when a domain is blocked we only do the rbl lookups and forget about the normal query. In the new requirement you will need to send the normal request and read the response of the blocked domain for the ip and log it. Ideally, logging would have to be done after the response is sent. You could do this by enabling the thread to do the normal request even if the db function reports that the domain is blocked. Then, you will have to call the method to send the response just after calling the normal query (do not wait for it).

Another way is to parse the blocked response, read the ip and then change the mysql query in `readP.cpp` to satisfy something like this: where `domains = <blockeddomains>` update the blocked ip field.

APPENDIX G

If a domain is found in both malware and child databases, only the unified response within the malware db will be logged. The task consists of logging also the unified response from the child database.

The way to do this is as follows:

In the `QueryDatabases` function of `DNSNameserver.cpp`:

```
Switch(status)

If malware found; log; return id;

If child found; log; return id;
```

If we do not detect malware we fall over to looking the child database. Else we return instantly. So, if a domain is found in both the malware and child databases no lookup will be done in the child db. But if it's found in either the malware or the child db we log it.

Solution or choose your own:

```
Switch(status)

If malware found; log; store id;

If child found; log; store id;

return id;
```