NAME

NBS − Never Before Seen anomaly detector suite

SYNOPSIS

nbs [options]
nbsmk [options]
nbsdump [options]
nbspreen [options]

DESCRIPTION

NBS implements a fast look-up database for arbitrary strings, and outputs strings it has never seen before. This capability is useful for system log analysis, network event detection or monitoring, or other security/system administration tasks.

There are few built-in limits to NBS; the database's limits are those of the underlying BSD-DB B-tree index, and the filesystem. Theoretically databases of millions of records and terabytes of information should function correctly and be fairly quickly searchable. The underlying records are stored in a flat file with hard offsets encoded in the B-trees; the files are thus not machine portable, and are optimized for check-time performance and append. Deleting records is not supported except as part of a batch file-preening operation using *nbspreen* to expire records based on last update time.

A typical cycle of use for the *NBS* suite is to create an *NBS* database with *nbsmk* then to use *nbs* to check status against and update its contents. The *nbsdump* program might be used in a batch process to dump the most (or least-) frequently seen items, for reporting purposes. Lastly, the *nbspreen* utility might be used to periodically "expire" old entries so that they will show up again as "never before seen" if they occur extremely intermittently (e.g.: annually or monthly).

When an *NBS* database is created, the database contains the settings that control how it is to be updated. Thus, it is important to correctly configure the database to suit your purpose. In most cases, the defaults will provide the performance and flexibility you need. When an *NBS* database is created, the utility creates several files, e.g.:
neverseen.cnt − the counts index (allows rapid sorting/retrieval of "bottom/top N" data)
neverseen.idx − the master index (allows rapid neverseen checking and sorted retrieval of data keys)
neverseen.rec − the data records and their update information
neverseen.upd − the update time index (allows rapid sorting/retrieval of "least/most recently" updated data)

In extreme performance situations, it is possible to ask *NBS* to not update some of the indexes, or even the primary record database. It is possible to run *NBS* in a mode so that the only index that is kept is the primary B-tree; though this is seldom necessary. Depending on the indexes that are enabled or disabled, some of the files in a given database may be absent, or may remain small and never grow. Users should only worry about leaving indexes out if NBS appears to be a performance or disk space hog once tested operationally. User experience shows this is seldom an issue.

Typical applications of the *NBS* suite are to track DHCP assignments, inter-host connectivity, URLs seen in web server logs, sender and recipient addresses of Emails, attachment names, IDS alerts seen, etc.

NBSMK Options:

−d database specify the root name for the database to create. The default is **neverseen**. Databases may be fully-pathed.

−K create the database to not maintain the count B-tree (".cnt") file.

−U create the database to not maintain the update B-tree (".upd") file.

−0 (zero) create the database to not maintain the record file. In this case the ".rec" file is created so it can contain set-up information and be used to synchronize access via locking, but will not be updated with records.

−S create the database to store records in *structural analysis mode* which is an experimental technique in which *NBS* takes its input and converts it into an abstract printf-like string that discards the numbers and strings while replacing them with template specifiers such as "%d". Thus, the string **"21:24:35.715495**

216.193.194.13.ftp-data" would get converted to: **"%d:%d:%d.%d %d.%d.%d.%d.%s-%s** and only new templates would be logged when discovered. This mode may be useful for detecting corruptions in highly structured data.

−D toggles some diagnostic/debug output.

NBS Options:

−d database specify the root name for the database to use. The database must already have been created with *nbsmk*.

−i infile specify the file from which to read input. Default is standard input. Inputs are considered to be line-oriented non-binary data.

−o outfile specify the file to which to write output. Default is standard output. Standard/IO buffering is performed on output; while the output is line-oriented, no guarantees are made as to when buffers are flushed. This option truncates the output file.

−a outfile is the same as **-o** except that the output file is appended to, rather than truncated.

−e errfile specify the file to which error messages should be written. The error file is truncated.

−E exit with the number of never before seen lines as the exit status (or -1 in the event of an error). Normal exit status is 0 (success) or -1 (failure).

−s Output summary statistics about the database and percentage if the input lines that were never before seen.

−D toggles some diagnostic/debug output.

NBSDUMP Options:

−d database specify the root name for the database to use. The database must already have been created with *nbsmk*.

−k key specify a single key to look for and dump if it exists. This would generally be combined with the -V option. For example, if you wanted to know how many times "index.html" was accessed you could use **nbsdump −V −k index.html**

−c # specify the number of entries to dump. This is generally used to get top/bottom counts (e.g.: "top ten list") and is often combined with the **-R** flag to control ordering. Depending on the particular value that is requested (default is the key string) the ordering will be either the key string, the count, or the update time.

−K order output by the sorting sequence of the keys in the database (alphabetic or reverse-alphabetic). This flag is actually a no-op, since **-K** is the default.

−C order output by the number of times that the key has been seen (count). This builds a sorted frequency chart for the key.

−U order output by the update times for the record. This builds a "most/least recently updated" list.

−R reverse the sort order. Default sort order is small to large. If you want the top 10 items from the database you would use **nbsdump -c 10 -R**

−T dd:hh:mm:ss apply a time mask to the dump; disregard records older (in seconds) than the specified time. Values can be omitted from the right hand side; i.e: **−T 30** would disregard records older than 30 days. **nbsdump -c 10 -R -T 60:0:0:1** would print the first 10 records, in reverse order, that had been updated within the last sixty days and one second. Bear in mind that this type of query will result in a "search until satisfied" traversal of the index so performance may suffer.

−V turns on more verbose output. With this flag selected the complete contents of the database record are printed out (whatever is available).

−M turns on minimalist output. With this flag selected, *nbsdump* tries to be as succinct as possible (usually just printing a key or count).

−X outputs in pseudo-XML-oid format for those who insist on XML-ish markup.

−Y outputs in year/date output where the date is the leading value.

−x outputs in "technical" format. Technical format is the anti-XML; it's **attribute=value** pairs suitable for extracting with shell scripts.

−s append statistics about the database's contents and state when completed.

−S append statistics about the database **only**. No records are printed.

−D toggles debugging output.

NBSPREEN Options:

−T d:h:m *mandatory* option specifies the cutoff for how old records should be in order to be purged. If fields are omitted, they are parsed left-to-right, I.e: **−T 365** means "discard anything that has not been updated in a year" whereas **−T 0:24** means "discard if older than 24 hours."

−d database specifies the database to use. The database must already exist.

−b specifies that the old versions of the database should be renamed +".BAK" instead of deleted in the update process.

−v turns on verbose output.

−D turns on diagnostic/debug output.

The *nbspreen* utility creates a new database and copies all the records from the original into it, if they are newer than a specified cut-off time. Once the operation is complete, the new database replaces the old. By design, the *NBS* system should not require frequent database rebuilds. I.e.: *nbspreen* is not a performance-enhancing tool; it is a tool for allowing you to decide a new value of "never before." This might come in handy if, say, you want to track DHCP leases but wish to be notified of repeat leases that occur very rarely - say every 3 months or so. In that case, you would rebuild the database with *nbspreen* every day, week, or month, discarding everything older than 3 months with **−T 90**. Be aware that rebuilding the database requires reading and rewriting the entire database as a new file; make sure there is enough disk space and that the I/O load will not be burdensome.

EXAMPLES

NBS can be used for a wide variety of applications but is primarily aimed toward anomaly detection. For a simple example, let's suppose we want to detect new access requests against web pages in our Apache-based web server.

Log messages on this system are in the form:
10.10.10.10 - - [02/Dec/2002:20:39:56 -0500] "GET / HTTP/1.1" 200 2187 so we write a simple perl script to pull out the URLs:

```
#!/usr/bin/perl
while (<>) {
    if(/"GET.*HTTP/) {
        @array = split(/ /);
        print("$array[6]\n");
    }
}
```

This perl script dumps out URLs when run as follows:
retail /var/www/logs/access_log | geturl.pl

We then create an *NBS* database to store the URLs:
nbsmk
and populate it from a cron job as follows:
retail /var/www/logs/access_log | geturl.pl | nbs -o new

Any new records that are introduced into the database will be stored in the file "new" which can be Emailed to an administrator.

If we subsequently wish to have a periodic Emailing of the top-ten URLs in the database, we might use:
nbsdump -c 10 -r | mail -s "top ten URLs" mjr

SEE ALSO

retail

BUGS

There are line-length limitations inherent in nbs' main input routine. Lines longer than BUFSIZ - 1 will be truncated and may get treated as multiple lines of input. While nbs' database should be able to handle large amounts of information, you should be conscious of the size of the records; larger records in the main index will impact performance and disk use.

It would be nice if there was a way for NBS to retain an entire line of input but only perform NBS parsing on a subset of it. Unfortunately doing that would require an internal parser; it seems better to keep parsing external.