Tools: NBS

- NBS = *N*ever *B*efore *S*een anomaly detector
  - Childishly simple
  - Devilishly useful (or not, depending on the battle you choose to fight with it)
- Basic premise:
  - If we've ***never seen it before*** by definition it's an ***anomaly***

Typically, the reason people don't do NBS is because it might (potentially) have to manage a lot of data entries and the processing could get intense. Imagine mapping connectivity between all your 3,000 hosts! There are 9,000,000 possible combinations. The database would get huge!

Fortunately, I've solved that. The next few slides provide a walkthrough of the NBS anomaly detection driver.

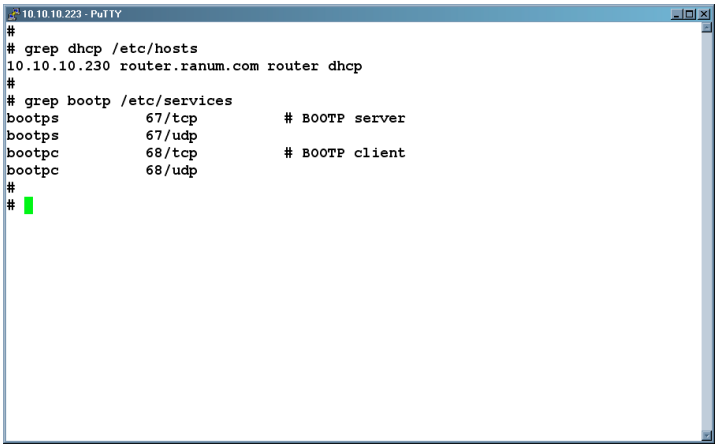
Building NBS

- Requires the BSD db library
  - http://www.sleepycat.com
  - Read the installation directions at http://www.sleepycat.com/docs/
  - Simple “configure” to build
- Requires the NBS utility
  - http://www.ranum.com (follow links to computer security and then to “code”)
  - Unpack and type “make”

To build the NBS driver you’ll need to BSD database library; NBS is basically nothing more than a couple of B-tree indexes, and the BSD database library has a very high-performance and reliable implementation of B-trees.

Once you’ve build BSD-DB (read the directions! It’s not just “configure” “make”) and installed it, then get the NBS sources off of Marcus Ranum’s web page and unpack and compile them. You may need to change the LIBS= parameter in the Makefile and may need to add -I/usr/local/BerkelyDB*blahblah* to the CFLAGS= parameter so that the #include file for <db.h> is found. It shouldn’t be hard to build!

NBS in Action: New DHCPs



This is a walkthrough example of how easy it is to use NBS and the kind of results you can get with it. Setup time for something like this is a few *minutes* and it'll give you results that might make you a local hero someday.

Here what we are doing is set-up stuff. We're going to monitor our network for Never Before Seen MAC/IP combinations being leased by DHCP servers. If any of those 3 values changes, we're going to know about it!

So, with one simple operation we'll:

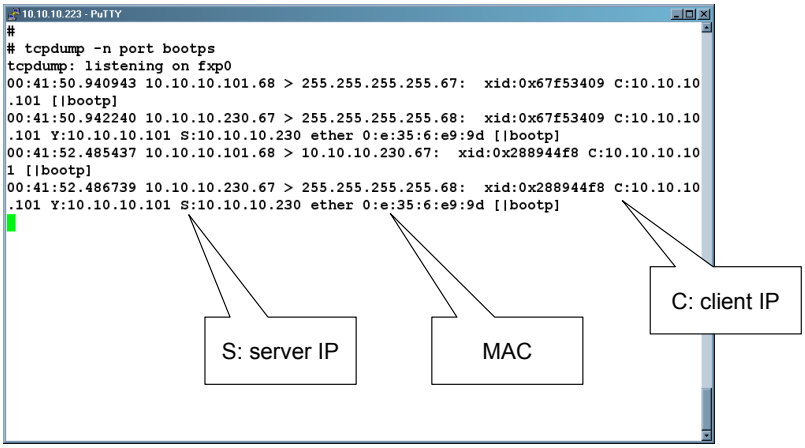
•Learn about new MACs

•Learn about new DHCP servers (that should be rare!)

•Learn about new IP addresses being leased that we don't normally lease

First subtlety: DHCP is kludged onto bootp. So we need to look at bootp traffic. Bootp has a server and a client side. Let's watch the server, because it's less likely to lie. If we were looking for NBSsing malformed records and weird client queries we could look at the client side, but that's a project for another day.

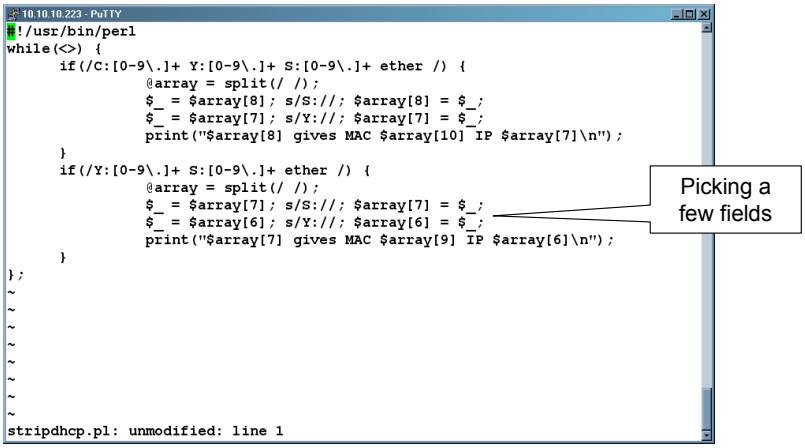
Tcpdump as log data source



NBS works on text strings. *Any* strings. So, we want to turn our bootp service traffic into strings. What better tool to do that than good old Tcpdump?

Here we run tcpdump on bootp traffic and observe some of how it works. The main thing we observe is the layout of the traffic that we're looking for. Aha! There's the client, server, and MAC address. We have everything we need - now let's just strip those fields out with a little script!

Marcus is a Perl Newbie

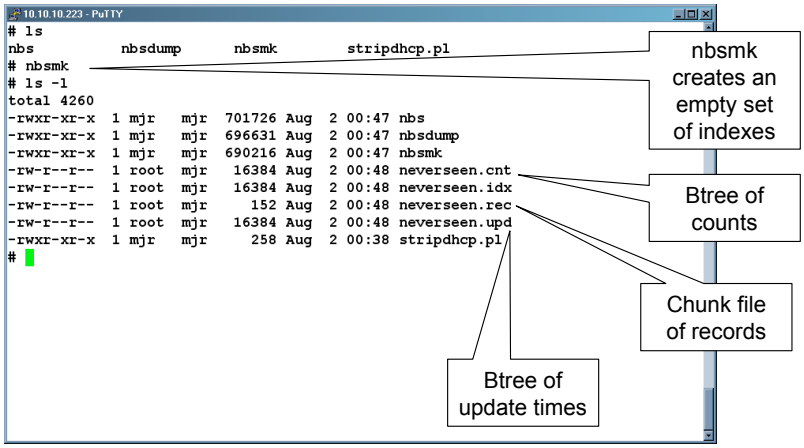


I suck at perl. In fact, you're looking at my very first (and so far: only) perl program. Normally, for something like this I'd use awk but perl seems to be everyone's choice these days… *sigh* Dan Klein has since explained to me how to write this same script much better. But you get the idea.

Anyhow, what this script does is matches on the two different forms in which *the version of tcpdump on my machine* outputs the DHCP lease-giving packet. One downside with using a program as input to another program is that different versions may vary their outputs and break everything.

Just make this work for your machine and don't mess with it.

Initialize NBS Database



The program nbsmk is used to create a new NBS database. The way NBS works, it reads its configuration from the database itself. So once you've created the database you can just call nbs on it and it'll figure everything out.

The default nbsmk parameters create a Btree of counts, a Btree of update times, a Btree of nbs strings, and a record file of update/count information.

There are a bunch of options to nbsmk that can cause it to omit some of the indexes for additional performance. In fact, if you want no information other than just that a record was seen, you can omit updating the data records and save a lot of space and time. Generally, nbs is fast enough that you don't need to worry about performance even for very large datasets.

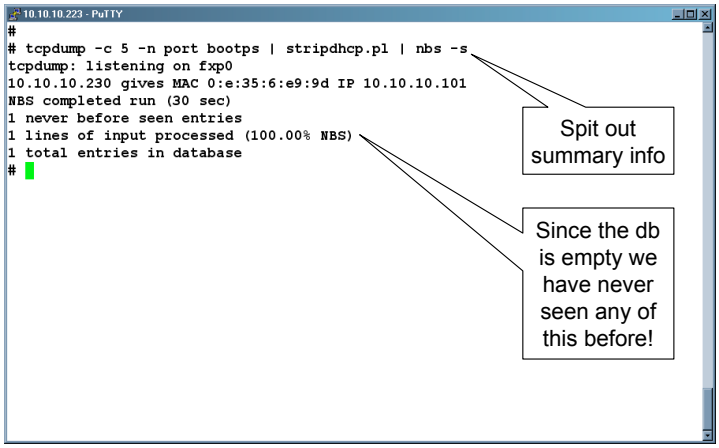
If you call nbsmk with the -? Flag or a bad flag it'll print out its current set of options. The most popular alternative option is the:

```
-d database
```

option. This allows you to set the database name to something other than the default (neverseen). Full pathnames also work, e.g.:

```
nbsmk -d /var/nbs/dhcp
```

Tcpdump | stripdhcp | nbs



Here we run the initial "training" round of nbs. Since it's never seen *anything* before, everything it sees now is an anomaly. So nbs prints it out on the standard output. The:

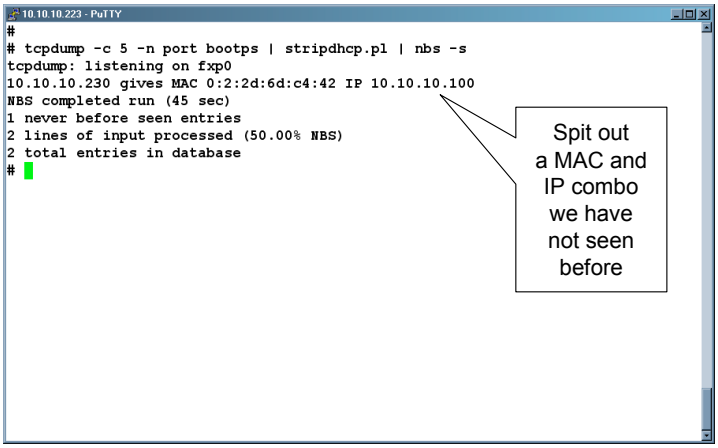
-s

flag tells nbs to output a summary when it is completed its run. The "100% NBS" score indicates that every line of input we sent through had never been seen before. That's not unexpected.

In this particular example, our use of NBS is a bit contrived. Tcpdump is exiting after collecting 5 packets, in order to cause it to actually terminate. In a production setup, you might run this in a loop, in which tcpdump exits after every 100,000 packets, so nbs flushes and the script is able to process the output. An alternative approach is to have a script that starts and restarts tcpdump:

```
OLDPID=`cat /var/log/tcpdump.pid`
mv newdump.out olddump.out
nohup tcpdump -n -w newdump.out &
echo $! > /var/log/tcpdump.pid
kill -HUP $OLDPID
tcpdump -n -r olddump.out | nbs -s > /tmp/nbs.out
if [ -s /tmp/nbs.out ]; then
        cat /tmp/nbs.out | mail -s "NBS report" mjr
fi
rm olddump.out
```

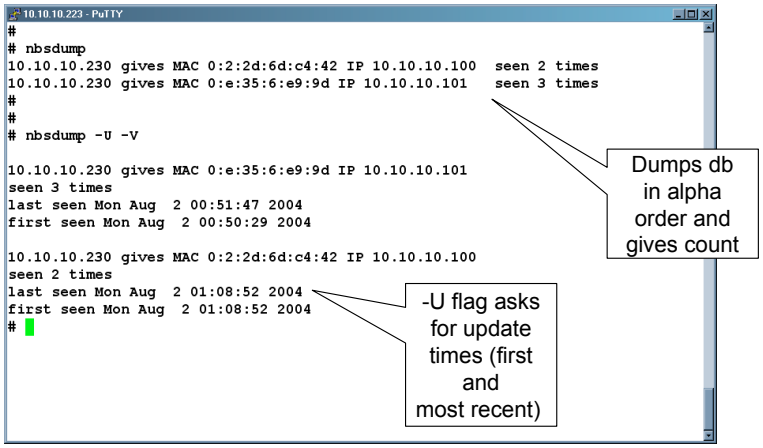
More input...



When we run a second pass with new data (I turned on a new wireless machine in the home network) - surprise! We see a never before seen value! From now on, we'll never get notified about that particular value, unless we age it from the database.

That's enough of a simple example! Let's look at a more complex and possibly more powerful one!

NBSdump



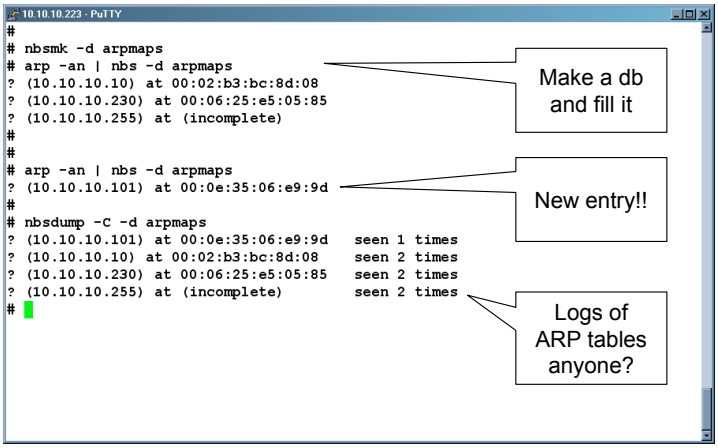
Here we are using nbsdump to output the contents of an NBS database. There are a variety of different flags you can pass to the dumper to control the format of the output as well as the order and count of what is dumped.

One of the neat things about Btrees is that they inherently sort the data that is stored in them. That's *why* nbs uses Btrees - when you dump the records, you get them sorted based on whatever value you asked for. If you ask for counts, you get the results sorted from least frequently seen to most frequently, etc. This can be incredibly useful and - most importantly - incredibly fast. A lot of the time log analysis is not done because of the processing constraints inherent in searching and sorting through lots of records. Nbs keeps the databases the way it does so that things like counts are pre-computed for you.

In the first example, we dump the values, and it displays the number of times they are seen.

In the second example, we asked for the last update time (-U) and full output (-V) which prints the time that the record was first installed in the database as well as the last update time. When we requested the last update times, the output is sorted in terms of most recent to least recent.

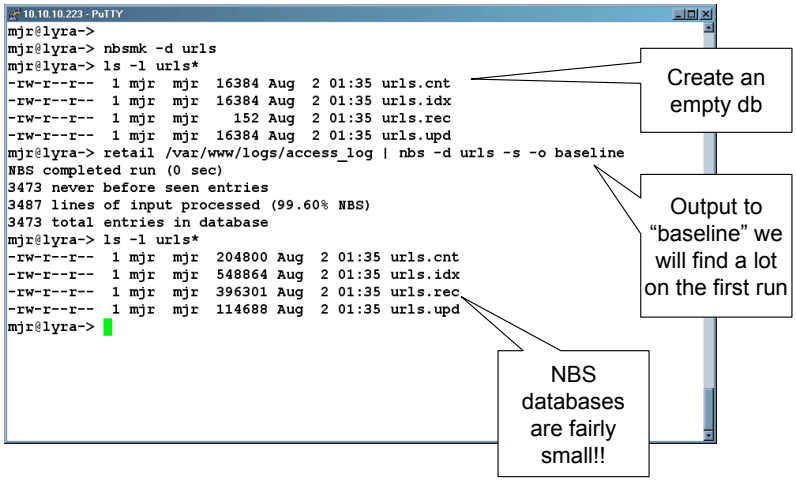
NBSsing ARP maps



In this example, we're tracking ARP/IP mappings on a network. Remember, we won't get notified whenever there's a *change* we'll only get notified when something happens that we've never seen before.

This would be a fairly simple cron job. Operationally, I'd expect to have a cron job run every 20 minutes that collected various bits of never-before seen stuff into a directory of output files. I'd then put a master script together that wrapped them all up into an Email message to the administrator.

NBS Apache Logs

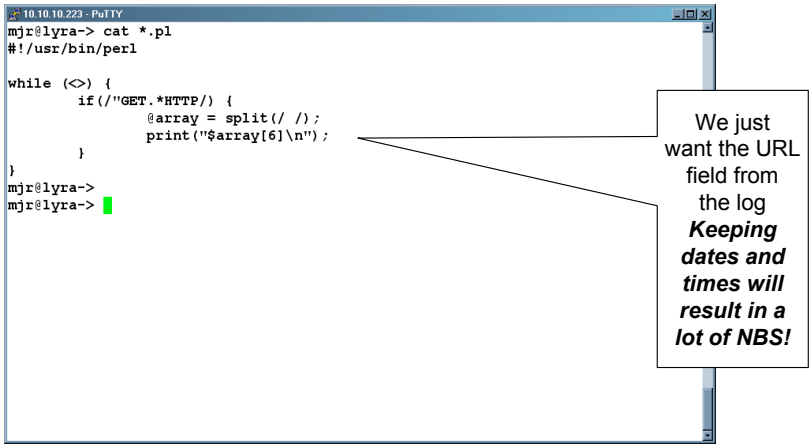


Here we're dumping web server access logs into an NBS database.

You'll notice that most of the URLs going into the database are ones that it has seen before, even during the training run. This works well on my website because there are not a lot of files or dynamic content.

NBS does not work very well on sites with a lot of dynamic content; since everything is "never before seen" if it changes all the time.

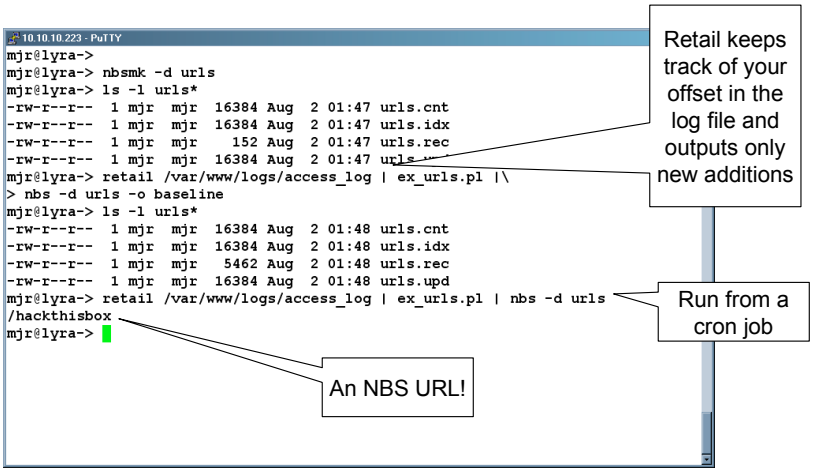
Did I mention I am a Perl newbie?



This is my second-ever perl program, so please be forgiving.

All that this script does is pulls the URL out of “get” methods in the log file. So we’re just going to build a database of URLs that have been requested from our system. Very simple.

Batch Log Processing w/Retail

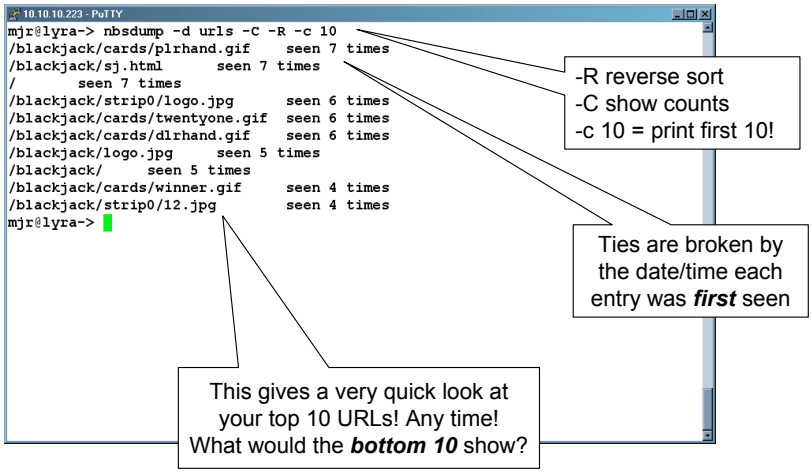


The retail program is extremely useful for passing data into an nbs database. Basically, it is a stateful "tail" that dumps whatever has been added to a log file since the last time it was called. This is useful since it gets you out of having to keep the log monitoring process constantly running. In this example you might put the command line as used right into a cron job to run every 15 minutes.

In the example above, I "trained" the nbs database by running it once with my log records. I sent the output to "baseline" with the "-o baseline" option, and then threw the baseline file away once the database was constructed. After the baseline is built, any new requests that come in will appear as NBS!

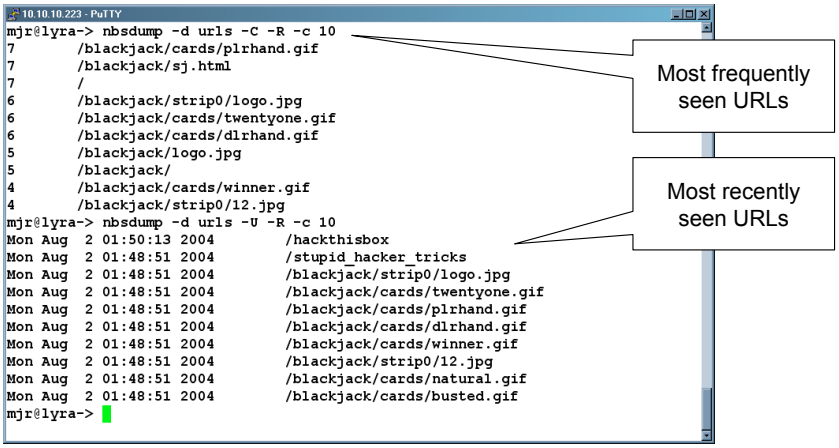
For the sake of making this an interesting viewgraph, I went in another window and opened a browser to my web site and entered a URL for a document that does not exist. You can see that it came out the back of NBS on the next run, just like it was supposed to!

Statistics from NBS



Nbsdump is also very useful for collecting statistics about arbitrary things. Because the data is all stored sorted in a Btree it's extremely quick if you want to retrieve "top 20" or "bottom whatever" values. In this example, we use nbsdump to retrieve counts (-C) in reverse order (-R) highest-to-lowest and to print the first 10 (-c 10) values.

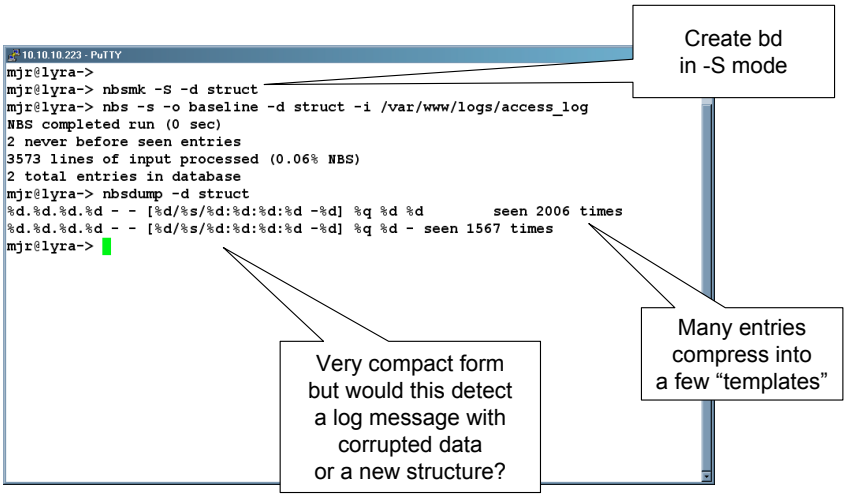
Statistics from NBS



Depending on the options you give nbsdump, you can get a variety of forms of output. Generally, the output is sorted by whichever key you asked it to be retrieved with. In the first example, we asked it to dump counts (-C) in reverse order (-R), so we got a sorted frequency chart. In the second example we asked it to dump in order of date and time, so we got the data back as most recently seen URLs.

Whenever nbs is sorting data if there are duplicate entries, it sorts based on which ones entered the database *first*.

Structural Analysis Mode

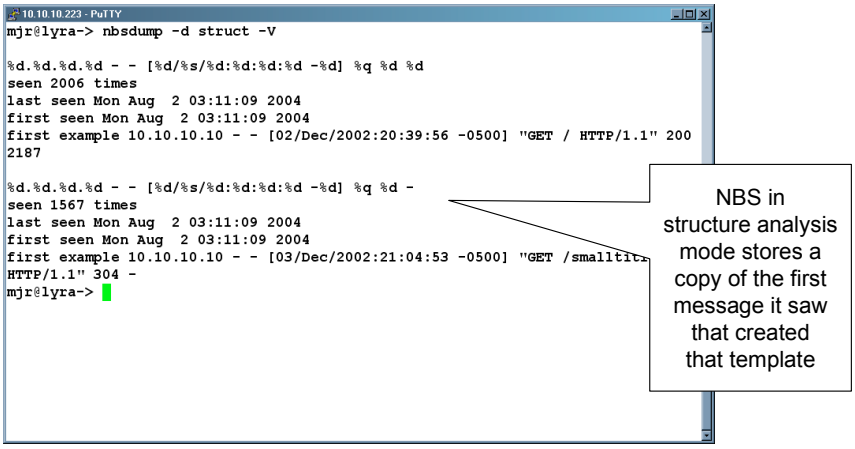


Nbs also includes an experimental capability called *structural analysis mode*. In this mode, what nbs does is creates a database of "templates" that represent the structure of a message with all the potentially variable fields knocked out into %s-style parameter strings. The variable fields are then thrown away ad al that is stored is the structural templates.

The call to nbsmk -S tells it to create the database in structural analysis mode. Then we send my web server access logs into it. Here you see that my entire web server logs reduce to 2 templates!

In this case, this particular rule would not be super interesting because the templates compress down to mostly a quoted string (see the "%q" in there?) If we wanted to do more effective watching of just templates, we'd look only at the URLs as non-quoted strings. The potential transforms are limited only by your imagination!

Show Examples of Templates



If you dump a structural analysis mode database in verbose mode, it will print out the first example it saw of an entry that had that structure. It's potentially deceptive, though, since simple structures (e.g.: "%s.%s") may match a lot of things!