ServiceTrak Meets NLOG/NMAP

Jon Finke – Rensselaer Polytechnic Institute

ABSTRACT

Network port scanning tools can be helpful in mapping services and exposures, but in large environments you often get more information than you can handle. This paper describes a project to take the output from NMAP/NLOG and merge it with the existing enterprise host management system. This makes it simple for service or platform specific administrators to study the machines in their purview.[1]

Introduction

Due to some recent security problems at our site where a number of machines were cracked due to exposures in in.statd, we decided to step up our project of removing or disabling unneeded services on the workstations and servers maintained by our department. Unfortunately, the press of other projects prevented our system administrators responsible for the various operating system (AIX, Solaris, and IRIX) from working on the investigation. So rather than try to look at these systems from the inside, we decided to try looking at them from the ''outside''.

NLOG and NMAP

At about the time that this project started, our networking department had installed NLOG [5] and NMAP [4] and had started scanning all of our address space. This meant that we had all of the raw data we needed on ports (services), and we just needed to break it down and display it so that we could understand it effectively. This method has the added advantage of being able to scan machines that might not be directly supported by the staff in our department.

To quote from the NMAP web page, ''nmap is a utility for port scanning large networks.'' It is actually a collection of modules that can scan TCP, UDP and ICMP services, and has facilities for bypassing firewalls and packet filters, as well as techniques to reduce the chance of scans being detected. While the stealth features are not of interest to us, the general port scanning facility is quite useful. Another interesting feature of NMAP is identifying the operating system via ''TCP/IP fingerprinting''. By sending a number of special packets with assorted options set (and other techniques), NMAP is often able to identify what operating system, and sometimes even the version of the operating system that is running on the target computer based on the responses to the test packets.

There is a related package called NLOG that works with NMAP. The NLOG home page states ''Nlog is a set of perl scripts that help you manage your NMAP log files. Included are conversion scripts, a CGI Interface, and the documentation to build your own analysis applications.'' Although the NLOG package offered some nice query options, it is somewhat limited by not being able to break down the information into the format that we needed. My group supports a number of public access Unix workstations, as well as some, but not all, of the computers owned by institute faculty and researchers, spread over a number of different buildings, subnets and domains. Thus a simple sort by IP address or domain is not enough for our needs. It is also nice to know who owns and administers each computer, and NLOG doesn't have that information readily available.

Service Trak

In 1997, we started a project called *ServiceTrak* [3] to assist in documenting our meta-system configuration. ServiceTrak generates a number of web pages of system configuration and server information. This seemed to be a natural place to hang the port scan information. We added the services (ports) discovered by NMAP to each of system (computer) pages, and added a new tree of discovered services, with all of the systems (hostnames) providing that service. One of the neat features in NMAP is to identify the operating system being run by the target system. It was interesting to see how this related to what we thought was there.

Since the actual port scans were being done by another department, I was only concerned with the database portion of the project. This broke down into four parts; loading the raw NLOG data into Oracle, comparing the raw data to the previous scans, defining some controls and ''safety levels'' on services and finally generating some web pages.

Converting and Comparing

The process of comparing new port scan data to our existing port scan information is closely coupled to the Oracle load process. One of the things we want to maintain is some historical record of what ports were open on a host, even if they are no longer open. This can help to track what problems and unexpected services that have been cleaned up. To this end, we want to keep track of when we last checked for a host or port, as well as when we last found something.

[1]Normally, I would use the term ''domain'', but in many cases, the machines in question are NOT grouped by internet domain.

Raw Data Load

The NLOG database format is intended to be processed by PERL scripts, so that it lends itself to easy reformatting. One of the utilities in the NLOG package converts NMAP scan files into NLOG ".db" files. For each host, it lists the IP Address, the number of ports found, the actual port list and NMAPs best guess at the OS of the target host. The port list includes each of the port numbers and the associated protocol (TCP/UDP) that it found for that host.

Our first step is to get the raw data from NLOG loaded into Oracle. This will allow the rest of the processing to take place inside the Oracle database server.

These files are processed by a shell script, which figures out the scan date for each file from the file system (ls -lt), pushes the contents of the .db files through awk to generate one .sql[2] file for each .db file. The script also generates a master .sql file that invokes each of the generated .sql files and then calls the rest of the processing. All together, this collection of scripts and generated scripts manages to load all of the raw data into the database and handles the housekeeping details. The generated .sql files are mostly made up in SQL insert statements to load host data into the Raw Host Data table (see Table 1) and the port information into the Raw Port Data table. After the raw data is loaded, the Domain_Id field is filled in using the Oct_1-Oct_4 values in the Hostmaster [1] IP Address table[3]. The Domain_Id plays an important part in comparing the raw data with the historic data.

[2]The Oracle command line interface, SQL*PLUS will read .sql files for oracle commands and expressions.

[3]All of our Hostmaster (DNS) data are maintained and stored in our Oracle database. The Domain_Id is an internal, unique identifier for each host at our site.

Comparing Raw Host Data To The Historical Data

Now that we have all of the raw host and port data split up and loaded into the database, we need to compare it with our historical data. Rather than pulling the data back out of the database to compare it, we wrote a set of PL/SQL[4] procedures. The original scans are done as a range of one or more subnets, so a given set of data files to be processed will have some, but not all of the hosts on our network.

Preparing and Retrieving the Host Data

Extracting the raw host data is easy. Since we are interested in processing all of the raw data we have, a simple database select is possible. In the processing code (see appendix 1), we define the following cursor[5]:

```
Cursor Raw_Host_Scan is
Select IP_Add, OS_Type,
       Load_date, Domain_Id,
       Rowid
  from Raw_Host_Data
 order by domain_id;
```

Although the historic data (stored in the NMAP_System_List, see Table 2) is derived from raw host data, we don't need to include all of the raw data. Since we can obtain the IP address from Domain_Id, we don't need to include it here. We have added some

[4]PL/SQL is Oracle's procedural extension to the SQL Standard. It allows for programs to be executed on the database server which can yield performance gains, as well as storing the code in the database itself.

[5]A cursor allows you to define a query in PL/SQL

Name	Type	Size	Description
IP_Add	Varchar2	32	IP Address of host. This is the primary key.
Oct_1	num	3	First Octet of the IP address. Breaking this out simplifies some database operations.
Oct_2	num	3	Second Octet of the IP address.
Oct_3	num	3	Third Octet of the IP address.
Oct_4	num	3	Forth Octet of the IP address.
OS_Type	Varchar2	255	The type of operating system as determined/guessed by NMAP
Load_Date	Date		Date we loaded this record.
Domain_Id	num	9	Internal identifier to match with Hostmaster tables.

Table 1: Raw_Host_Data Oracle table definition.

Name	Type	Size	Description
Domain_Id	num	9	Internal identifier to match with Hostmaster tables. This can be used to get the hostname and the IP address.
OS_Type	Varchar2	255	The type of operating system as determined/guessed by NMAP
First_Detected	Date		When we first detected this IP address.
Last_Updated	Date		When we most recently found something at this address.
Last_Checked	Date		When we last looked on this subnet.

Table 2: NMAP_System_List Oracle dable definition.

additional date fields to keep track of past scans. However, since we may not be processing all subnets[6], we just want to work on the historical records for the subnets found in the current set of raw records. To do this, we define the Historic_Host_Scan cursor:

```
Cursor Historic_Host_Scan is
Select nsl.Domain_Id, nsl.Rowid
  from Nmap_System_List nsl,
       dns_ip_address dia
 where nsl.domain_id =
       dia.domain_id
   and dia.ip_octet_3 in
        (select distinct oct_3
           from Raw_Host_Data)
 order by nsl.domain_id;
```

This query will get the all of the previous NMAP scan entries we have processed on this set of subnets[7]. This basically has four steps. The first step is the sub query Select distinct oct_3 from Raw_Host_Data, which will create a list of all the different subnets found in the raw data. The next step is to identify all registered IP addresses in the DNS_Ip_Address table and return those Domain_Id s. This list of Domain_Id s is then matched against our historic host scan returning the Domain_Id and Rowid of the historic record. Finally, this list is sorted by the Domain_Id . The Rowid is an internal Oracle identifier for that specific entry in the database. It can be used for rapid (efficient) access to that row.

Processing the Host Data

We now have two lists, each sorted by the Domain_Id : the ''raw'' data from the most recent scan, and related historic data. By comparing the Domain_Id s while stepping through both lists together, we can detect newly found (scanned) hosts, historic hosts that were not in the latest scan, and hosts that are in both lists. (The PL/SQL source code is available in Appendix 1.)

If it ''raw'' Domain_Id is lower, that means we found a new system that was not previously in the historic data. We insert this record into the database and get the next entry in the ''raw'' list. (If there are no other changes, the lists should be ''in synch'' again.) If the two values are the same, the records are for the same host and we need to compare the port values (see the next section).

If the ''raw'' Domain_Id is higher than the Domain_Id from the historic record, then the record in the historic list represents a previously known host that was not detected in the most recent scan. There are several possible reasons for this: the host may have been retired, there could have been a network problem, or the host may be down for some reason. We want to maintain the historic data for this host, including the fact that it was not found. We do this by updating the Last_Checked field in the historic data. (We don't do anything with the ports, since we don't have any data.) This condition will be identified by the reporting tools. The historic list is then advanced to the next record, and the process repeats until the lists are in synch again.

Consider the example data in Table 3. There are both raw and historic data for sam.rpi.edu (Domain_Id 7), so we update the port information and then get the next raw and next historic Domain_Id s. Now we find that the raw id (12 - george.rpi.edu) is less than the historic id (15 - fred.rpi.edu), so we know that george.rpi.edu is a new record, and we insert it. We then select the next raw entry where we get fred.rpi.edu (15) which matches the existing historic entry. These two match, so we update the ports and get the next two entries in the list. This time, the historic entry dave.rpi.edu is less than the raw entry, sharon.rpi.edu. Since we didn't detect dave.rpi.edu in the most recent NMAP scan, so we need to update the Last_Checked_Field it and then get the next historic entry, sharon.rpi.edu, which matches the raw entry and we do the port compare and we are done.

Raw Id	Historic Id	Name
7	7	sam.rpi.edu
12		george.rpi.edu
15	15	fred.rpi.edu
	22	dave.rpi.edu
25	25	sharon.rpi.edu

Table 3: Example data.

Handling Port Data

We also need to keep track of what ports (services) were encountered for each host. To this end, we have a raw port table (Table 4) and a historic port table

[6]The NMAP scans are done on a ''Class C'' subnet basis, even if target host subnet is different. This makes it easier for the people running NMAP, and makes the processing easier.

[7]All of the IP address information happens to be stored in this same database, making this selection on IP address very easy. Alternately, the subnet information could also be stored in the NMAP_System_List table.

Name	Type	Size	Description
IP_Add	Varchar2	32	IP Address of host. This is the primary key.
Port	Num		Port number.
Family	Varchar2	12	Port type such as ''TCP'' or ''UDP''.
Load_Date	Date		Date we loaded this record.

Table 4: NMAP_Raw_Ports Oracle table definition.

(Table 5). As with the historic host information, we don't need to keep track of the IP Address in the port table, instead we can use the Domain_Id to associate the ports with the host.

For each host we are updating or inserting, we select the ports to go along with the raw and historic records. For the raw records, we use the IP address as the key, and for the historic records, we use the domain_id as the key. We order the records from both sets by port and family, and again loop through, comparing ports and families, adding, deleting and updating records as needed. To get the raw port data, we use the following cursor, using the address value to limit the ports to just the ones for the current host (address) in question:

```
Cursor Raw_Port_Scan
     (address varchar2) is
Select IP_Add, Port, Family,
    Domain_Id, load_date,rowid
 from Nmap_Raw_Ports
where IP_Add = address
order by Port, Family;
```

In a similar way, we get the historic port information for this particular host, using the Domain_Id as the key:

```
Cursor Cooked_Port_Scan
      (Did number) is
Select Port, Family, rowid
  from Nmap_Port_List
 where domain_id = Did
 order by Port, Family;
```

Both of these selections are sorted by the Port, and then by the address family. These two lists are then compared, stepping through them, much like we did with the host comparison.

Controls

When we display ports (services), it would be nice to include a bit more information about the port than simply the service name. As a starting point, we loaded */etc/services* into a table. While this gives us a service name and sometimes a brief description of the service, we added another column called "safety". Since the primary objective of this project was to improve security, we rated each service as to how "safe" it is to have running. In some cases, while the named service might be considered safe, common security exploits may be using that port as a back door into a system. For these cases, seeing that this port active might be a good indicator that a computer had been hacked. In other cases, some versions of that service may have known holes, so additional checking

Name	Type	Size	Description
Domain_Id	num	9	Internal identifier to match with Hostmaster tables. This can be used to get the hostname and the IP address.
Port	Num		Port number.
Family	Varchar2	12	Port type such as "TCP" or "UDP".
First_Detected	Date		When we first detected this IP address.
Last_Updated	Date		When we most recently found something at this address.
Last_Checked	Date		When we last looked on this subnet.

Table 5: NMAP_Historic_Ports Oracle table definition.

Type	Description
Safe	Service is safe to have running
Server	Ok for server class machines, questionable for desktops
NoCrypt	This service exposes sensitive information to network sniffing
Suspect	Often indicates a hacked machine
Unknown	We haven't decided yet.

Table 6: Safety Validation Table NMAP_Service_Danger_Types.

Port	Fam	Name	Safety	Description
9	tcp	discard	safe	Discard packets
21	tcp	ftp	server	
22	tcp	ssh	safe	Secure shell
23	tcp	telnet	NoCrypt	Remote access
31	tcp	msg-auth	Unknown	MSG Authentication
1524	tcp	ingreslock	Suspect	ingres

Table 7: Services NMAP_Service_List.

may be required to determine if the computer in question is running a safe version of the service. Other protocols are inherently insecure and we may want to discourage their use. We defined the safety values[8] listed in Table 6.

The more challenging job of course, was to rate the services as to their safety level, and expand the descriptions where needed. Naturally, our safety ratings reflect our department's own needs and policies. A partial list is shown in Table 7.

One of our security objectives is to eliminate clear text passwords on the network. Thus, we flag "telnet" as a somewhat dangerous service. We also want folks to use central FTP servers, so that we mark FTP as "server". While we don't have anything against Ingres, we don't run it on our machines, and the ingres lock port was recently used in a particular exploit script. Machines with the ingres lock port open tend be hacked, rather than offering Ingres. Therefor we mark ingress lock as "suspect". We have not classified all services (our service file listed 924), rather we are concentrating on classifying the ones that are showing up on machines we are interested in.

The last control element we need, is some way to define how we want to group hosts in the output pages. We already have a number of host groups set up in our database, these are used for access control (like generating /etc/hosts.lpd), usage statistics [2] and so on. Thus, we had a start of the host groupings that we could use. This resulted in another table, Service_Track_NMAP_List.

This table is used in the generation of the top level of the NMAP web tree. At present, it allows breakdowns by host group and a dump of all hosts with detected services.

Outputs

The top of the NMAP web tree looks something like Figure 1[9]. There is a standard heading with logos and other boilerplate that we omit from this paper. Each of name entries is a link to a tree of ports for that group. A typical host group web page looks something like

[8] One nice thing about Oracle is the ability to define validation tables that define what values a column can have.

[9] In general, the generated web pages are simply HTML tables. Rather than taking screen shots from a web browser, we have reproduced the tables here. HyperLinks are indicated by italic.

Name	Type	Size	Description
Name	Varchar2	32	The name used for grouping.
Tag	Varchar2	16	Used in filenames in the web tree.
Group_Id	Number	12	Points to the top level of the host group "branch".
Description	Varchar2	255	A short description to be included in the web pages.

Table 8: Service_Track_NMAP_List Oracle table definition.

Name	**Hosts**	**Ports**	**Description**
All	3222	365	All scanned hosts
CIS Staff Workstations	60	35	computer center staff Unix workstations
Remote Access	4	19	Public Access Timeshare machines
CUSSP	217	53	Faculty Unix Support Program machines
RCS Workstations	284	45	Public Unix Workstations
NIC Cluster	62	101	High Performance Computing Cluster

Figure 1: NMAP Services.

Service Sample Page
ssh (22/tcp)
Remote Access NMAP Ports

- Secure Shell
- Safety Level: Safe
- Tab Separated Values Page

Hostname	**First Detected**	**Last Detected**	**Last Checked**	**OS**
cortez.sss.rpi.edu	12-May-99	12-May-99	12-May-99	AIX 4.2
rcs-sun1.rpi.edu	12-May-99	12-May-99	12-May-99	Solaris 2.6-2.7
vcmr-12.rcs..rpi.edu	12-May-99	12-May-99	12-May-99	Solaris 2.6-2.7
vcmr-19.rcs..rpi.edu	12-May-99	12-May-99	12-May-99	

Figure 1a: Service Sample Page

what is shown in Figure 2. Again, the standard heading is omitted.

From the host group page, we can continue to drill down the tree, selecting one of the services of interest. As seen in Figure 3, we add a little more to the basic table, along with "danger level" and description, we include a link to a Tab Separated Values Page. The TSV file has an entry for each of the systems listed, including useful information such as owner, sys admin, location, and a lot of other information that is already in the Service Trak database. This has proven very useful, since it can easily be loaded into a spreadsheet. Service Trak generates a TSV file for just about any page the has a list of computers.

We can continue to drill deeper. Each hostname listed is a link to that host's entry in the Service Trak tree. We have expanded that entry to include what ports were detected on that host by NMAP. Of course, you can click on those service names and end back up near the top of the NMAP tree.

Conclusions and Surprises

The first reaction of most of the staff to see the pages was "cool", followed almost immediately by "Hmm, I didn't know we were running that there, I will have to look in to that." In that, the primary objective of the project was achieved. It also pointed out a number of anomalies in our configurations. For instance, our remote access machines are all supposed to provide the same services. However, it was quickly noticed that the "host counts" for services did not match.

The last column in the list of hosts for a particular service, is NMAPs guess at what the operating system being used on that machine. We were somewhat disturbed to find some of our "trusted host" Unix machines (trusted for LPR) were in fact running Windows NT[10].

We also added the NMAP "guess" of the operating system to our hinfo program. The hinfo program takes a host name or IP address, and returns owner, admin and change history and anything else we have in the database for a machine on our network. This program is frequently used by our NOC staff when investigating oddities.

For security and privacy reasons, we do not allow public access to the Service Trak web pages. Although much of the information is available to the public, and NMAP is easy to install, we don't want to make things too easy for people who want to attack our site. The web pages are on a protected web server, limiting access by either IP address (the staff subnet) or via ID and Password authentication.

At present, we only output information via the web. However, since many of the Service Trak pages have a TSV option, staff interested in other formats can quickly download the page to a spreadsheet and produce reports in other formats.

Service Trak does have a list of "sanity checks" it runs on a daily basis, and we could add checks to

[10]The disturbing part was that we still trusted them, not that they were running NT. At present, our NT machines do not use our Unix name space, and "administrator" is NOT a valid Unix user!

Name	Port	Family	Danger	Count	Description
echo	7	tcp	safe	4	
discard	9	tcp	safe	4	
daytime	13	tcp	safe	4	
chargen	19	tcp	Unknown	4	
ftp	21	tcp	server	2	
ssh	22	tcp	safe	4	Secure shell
telnet	23	tcp	NoCrypt	4	
time	37	tcp	safe	4	
sunrpc	111	tcp	Unknown	4	
ident	113	tcp	safe	2	
loc-srv	135	tcp	Unknown	2	Location Service
smux	199	tcp	Unknown	2	snmpd smux port
exec	512	tcp	Unknown	4	
login	513	tcp	Unknown	4	
shell	514	tcp	Unknown	4	no passwords used
printer	515	tcp	Unknown	4	line printer spooler
uucp	540	tcp	Unknown	1	uucp daemon
ta-rauth	601	tcp	Unknown	2	For AFS kerberized services
writesrv	2401	tcp	Unknown	2	Temporary Port Number

Figure 2: RCS Remote Access Services.

look for changes or ‘‘unsafe’’ NMAP entries and generate an alert. However, we are not making regular NMAP scans. Each scan takes a day or so (this is mostly how we are doing it, my understanding is that NMAP can run a lot faster) and we have not resolved all of the policy issues with scanning the network. Since there are tools other than Service Trak may be more appropriate for detecting problems in real time, we have not pursued this avenue.

NMAP Quirks

We noticed that the NMAP scan appears to kill inetd on some platforms. This was later confirmed in some bug reports. These reports also mentioned some other systems that might react poorly to getting scanned. We certainly noticed some other system administrators who reacted poorly to being scanned. We are now exploring some of the policy implications of port scanning (by our own staff) as well as ways of running NMAP in less ‘‘destructive’’ modes. We have also removed some of our major network gateways from the scans (at the request of our network operations staff), as the scans generated a number of SNMP traps that generally indicate hostile activity.

Alternatives

Clearly, most places do not run ServiceTrak, and many do not even use a relational database to manage their system configuration. However, the thing that gave us the biggest boost, was the ability to look at systems based on our host groupings. The NLOG package may allow for simple integration of your own site configuration information. This would be a good avenue for exploration. Given that we already had ServiceTrak in operation, we did not explore expanding and enhancing NLOG.

At present, the Service Trak ‘‘output’’ is a huge tree of web pages that are regenerated on a daily or weekly basis. This regeneration takes a few hours to run, generates 34,000 files, and uses 120Mb of disk space. While disk space is still pretty cheap, I expect that some parts of Service Trak will by moving to dynamic web pages, generated on demand. Still, a lot less things have to work in order to make files available versus dynamic web pages, so I expect that at least part of Service Trak will remain in files, regenerated daily.

References and Availability

All source code for the Simon system is available on the web (or via AFS). See http://www.rpi.edu/campus/rpi/simon/README.simon for details. In addition, all of the Oracle table definitions as well as PL/SQL package source are available at http://www.rpi.edu/campus/rpi/simon/misc/Tables/simon.Index.htm .

Acknowledgments

I would like to thank David Hudson of Network Support Services, Rensselaer Polytechnic Institute for his work with installing and running NMAP and NLOG. I also wish to thank David Parter of the Computer Science department of the University of Wisconsin for his comments and suggestions on all the drafts of this paper.

Author Information

Jon Finke graduated from Rensselaer in 1983, where he had provided microcomputer support and communications programming, with a BS-ECSE. He continued as a full time staff member in the computer center. From PC communications, he moved into mainframe communications and networking, and then on to Unix support, including a stint in the Nysernet Network Information Center. A charter member of the Workstation Support Group he took over printing development and support and later inherited the Simon project, which has been his primary focus for the past eight years. He is currently a Senior Systems Programmer in the Server Support Services department at Rensselaer, where he continues integrating Simon with the rest of the Institute information systems, and also deals with information security concerns.

Reach him via USMail at RPI; VCC 319; 110 8th St; Troy, NY 12180-3590. Reach him electronically at <finkej@rpi>.edu. Find out more via http://www.rpi.edu/~finkej.

References

[1] Jon Finke. Simon system management: Hostmaster and beyond. In *Proceedings of Community Workshop ’92*, Troy, NY, June 1992. Rensselear Polytechnic Institute. Paper 3-7.

[2] Jon Finke. Monitoring usage of workstations with a relational database. In *USENIX Systems Administration (LISA VIII) Conference Proceedings*, pages 149-158. Rensselear Polytechnic Institute, USENIX, September 1994. San Diego, CA.

[3] Jon Finke. Automation of site configuration management. In *The Eleventh Systems Administration Conference (LISA 97) Proceedings*, page Unknown. Rensselaer Polytechnic Institute, USENIX, October 1997. San Diego, CA.

[4] Fyodor. Nmap free security scanner. Web Site, 1999. http://www.insecure.org/nmap .

[5] HD Moore. Nlog: Nmap log management tools. Web Site, 1999. http://nlog.ings.com .

Appendix 1 – PL/SQL Process host data

```
procedure Add_Host ( Raw in Raw_Host_Scan%Rowtype ) is
begin
        Insert into Nmap_System_List
               (System_Id, Domain_Id, Os,
                 First_Detected, Last_Updated, Last_Checked)
        values (Raw.System_Id, Raw.Domain_Id, Raw.Os_Type,
                Raw.Load_date, Raw.Load_date, Raw.Load_date);
        Update_Port_List(Raw);
        Delete from Nmap_Raw_Host_List
         where rowid=raw.rowid;
end Add_Host;
procedure Delete_Host ( Historic in Historic_Host_Scan%Rowtype ) is
begin
        Update Nmap_System_List
           set last_checked = Global_Check_Date
         where rowid = historic.rowid;
end Delete_Host;
procedure Update_Host( Raw in Raw_Host_Scan%Rowtype,
                       Historic in Historic_Host_Scan%Rowtype) is
begin
        Update Nmap_System_List
           set System_Id = Raw.System_Id,
               last_checked = raw.load_date,
               last_updated = raw.load_date
         where rowid = historic.rowid;
        Update_Port_List(Raw);
        Delete from Nmap_Raw_Host_List
         where rowid=raw.rowid;
end Update_Host;
procedure Compare_Host_Lists
is
        Raw     Raw_Host_Scan%Rowtype;
        historic  Historic_Host_Scan%RowType;
begin
Open Raw_Host_Scan;
Open Historic_Host_Scan;
Fetch Raw_Host_Scan into Raw;   -- Prime the pump
Fetch Historic_Host_Scan into Historic;     -- and this one
loop
   exit when Raw_Host_Scan%NotFound and Historic_Host_Scan%NotFound;
   <<InnerLoop>> loop
   if Raw_Host_Scan%NotFound then
     Delete_Host(Historic);
     Fetch Historic_Host_Scan into Historic;     -- and this one
        exit InnerLoop;         -- continue
   end if;
   if Historic_Host_Scan%NotFound then
   then
         Add_Host(Raw);
         Fetch Raw_Host_Scan into Raw;
         exit InnerLoop;         -- continue
   end if;
   -- Ok, we know they are BOTH here.... Lets compare
   if Raw.Domain_Id > Historic.Domain_Id then    -- Raw list MISSED a historic,
         Delete_Host(Historic);
         Fetch Historic_Host_Scan into Historic;     -- and this one
```

```
          exit InnerLoop;         -- continue
    end if;
    -- Lets try the other way
    if Raw.Domain_Id < Historic.Domain_Id then    -- New Raw record
          Add_Host(Raw);
          Fetch Raw_Host_Scan into Raw;
          exit InnerLoop;         -- continue
    end if;
    -- Not greater, not less, must be the same
    Update_Host(Raw, Historic);  -- Do the ports
    fetch Raw_Host_Scan into raw;
    fetch Historic_Host_Scan into historic;
    exit InnerLoop;         -- InnerLoop is just for a continue function
   end loop InnerLoop;
 end loop;
```