NetState : A Network Version Tracking System*

Nancy Durgin Yuqing Mai Jamie Van Randwyk

Sandia National Laboratories
Livermore, CA 94550

nadurgi@sandia.gov yuqingm@hotmail.com jvanran@sandia.gov

Abstract

Network administrators and security analysts often do not know what network services are being run in every corner of their networks. If they do have a vague grasp of the services running on their networks, they often do not know what specific versions of those services are running. Actively scanning for services and versions does not always yield complete results, and patch and service management, therefore, suffer. We present NetState, a system for monitoring, storing, and reporting application and operating system version information for a network. NetState gives security and network administrators the ability to know what is running on their networks while allowing for user-managed machines and complex host configurations. Our architecture uses distributed modules to collect network information and a centralized server that stores and issues reports on that collected version information. We discuss some of the challenges to building and operating NetState as well as the legal issues surrounding the promiscuous capture of network data. We conclude that this tool can solve some key problems in network management and has a wide range of possibilities for future uses.

1 Introduction

As computer networks grow larger, it becomes more difficult to manage those networks. It is increasingly difficult for information technology (IT) departments to manage large numbers of computers and similar devices on their networks. As users become more savvy, it is more difficult to control the network services that users run on their computing devices. In addition, viruses, Trojan-horses and worms may install "back-door" network services. Firewall and corporate policies are only able to control the spread of network services to a limited degree.

Because IT departments cannot always control which network services are being run on their networks, they must find a way to identify which services are being run on which devices. In the past, port scanning (using a tool such as Fyodor's *Nmap* [1]) was a reasonably airtight technique used to identify services running on a given network-enabled device. Now users install common network services on non-standard ports to get around corporate firewall restrictions. Some users install multiple operating systems on a single computer, rendering port scans incomplete. Trojan-horses use proprietary network protocols on seemingly random ports to conduct their nefarious activity.

Not knowing what services are running on one's network makes patch management and service management extremely difficult. This can open network devices up to compromise because the IT staff cannot identify and patch all instances of the affected service after a new vulnerability announcement.

We built NetState to passively monitor, store, and report application and operating system version information for a network. NetState includes sniffer modules that monitor traffic across a network, a backend database for storing service name and version information, and a GUI client for querying the database. NetState was built for internal use but is now being made available in the public domain.

We have organized the rest of this paper as follows: In Section 2 we survey existing open source and commercial tools that attempt to catalog service versions in existence on a network. Then, in Section 3 we discuss both our design requirements and our implementation. Our discussion of the implementation describes the three main components of NetState: the NetState Sniffer, the NetState Server, and the data query interfaces. In Sections 4, 5, and 6 we describe the performance of NetState

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

in a low-bandwidth network and relate our experiences and perceived challenges in using NetState on a day-to-day basis. Next, in Section 7, we give an overview of legal issues surrounding the "sniffing" of data in both the employer-employee and Internet service provider-customer scenarios. We conclude by discussing future work in Section 8.

2 Related Work

When we surveyed the commercial and open source communities for software to perform our desired functions we did not find anything that fit our requirements.

Several companies sell products that monitor and record network traffic for further analysis [2, 3]. Presumably, these products offer the ability to report network service versions, but we did not test for this. Company literature was also not clear in identifying the existence of this feature. We did not fully evaluate these products because their overall utility was far more than we needed.

Novell sells the desktop management product *ZENworks* that allows centralized management of many independent systems via a *management agent* running on those machines [4]. From a central location a network or security administrator can enforce a standard desktop environment, migrate personal settings, deploy software patches, and monitor system performance. The central management server provides a network analyst with in-depth information about each of the managed hosts. This includes information about network service versions. We would not be able to rely on such software to accurately represent our entire network though. Our laboratory research environment demands that some systems be managed solely by the researcher that owns them, often meaning that remote management utilities cannot be installed by our desktop support team. More importantly, ZENworks only supports the NetWare, Windows, and Linux operating systems. Our networks are home to systems running many operating systems beyond those supported by ZENworks, also including Linux distributions not officially supported. Finally, the capabilities of ZENworks are far beyond what we wanted to implement; the Novell software would duplicate functionality already established by competing products on our networks.

Nmap recently introduced a network service version scanning feature. Using the '-sV' or '-A' options, an analyst can identify the application name and version information when available. Nmap performs this inquiry for each open port that it discovers during the port scanning procedure. The community involvement with keeping the service version database up-to-date is especially valuable to Nmap. Nmap is designed to be an active scanning tool though. It cannot detect when a multi-boot system has been booted into a different operating system or when a machine that is often powered off has been powered on. In these situations, Nmap could fail to identify open and potentially vulnerable network services.

We have decided to release the NetState source code to the open source community for several reasons. Primarily, we believe that secure public networks are of benefit to everybody. A tool that allows network administrators to be more aware of the behavior of the machines on their networks is one more step towards this goal. In addition, while we have implemented version detection for many common protocols, we feel the open source community can contribute support for additional protocols or improve upon the current detection methods.

3 Architecture

Our design was formed after discussing goals with our security analysts, studying our existing security architecture and evaluating a prototype tool that we had written. We first discuss some of our design requirements and then describe how we met those in our implementation.

3.1 Design Requirements

- **Fits into our existing network security framework:** The server computer should exist on a private security network. Only security personnel, other authorized personnel, and machines owned by security should be able to access this machine via the network. The network traffic being monitored should be on a separate network.

- **Comprehensive data collection:** Traffic should be monitored at every network ingress and egress point. This includes wide-area Internet links, VPN concentrators, modem pools, wireless access points, point-to-point links with collaborators, etc.

- **Pertinent information only:** All information about network application versions should be collected for all computers, but we should not store any more than that. We should be able to determine which application version is/was running on which port on each device at any point in time.

- **Major network applications:** We should track information for "major" applications on our network. This includes services and client applications that are commonly used or are mission-critical. We implement this in such a way as to allow expansion to additional applications in the future.

- **Passive monitoring:** All version information should be collected passively by monitoring all the traffic on the target network.

3.2 Passive vs. Active Scanning

A major distinction of NetState's design is that it uses passive scanning techniques as opposed to the active techniques employed by tools such as *Nmap* and *Nessus* [5]. While active scanning techniques can often yield more detailed or precise data, for example by sending specially crafted packets that yield a definitive signature, passive scanning offers several advantages:

- Active scanning tools are "noisy", creating additional, and often unnecessary, traffic on a network.

- If a particular machine is turned off, a single machine boots multiple operating systems, or multiple machines share the same IP addresses at different times (via DHCP), it is difficult to guarantee detecting these situations via active scans. Since passive scanning monitors network traffic at *all* times, it yields information about what actually happens on the network, rather than just a snapshot of what the network looked like when the scan was performed.

These active scanning tools can still be used to assist NetState in gathering application version information. We can populate the NetState database by performing an active scan that receives its results under the nose of a NetState Sniffer.

3.3 Implementation

Our design goals led us to implement a distributed system consisting of several modular programs all working together as NetState. The architecture is shown in Figure 1.

The core of our system is a server process that accepts network traffic information from distributed Sniffer processes and places the information into a database. The NetState Server receives connections from NetState Sniffers via a private "security" network. The Server receives application version information over those connections and stores the information in a database. These connections could be established over the open network as well, but NetState needs built-in authentication before that is practical. The Server also responds to queries from authorized clients that are allowed to access the application version database. Access control is maintained using operating system-level firewall rules.

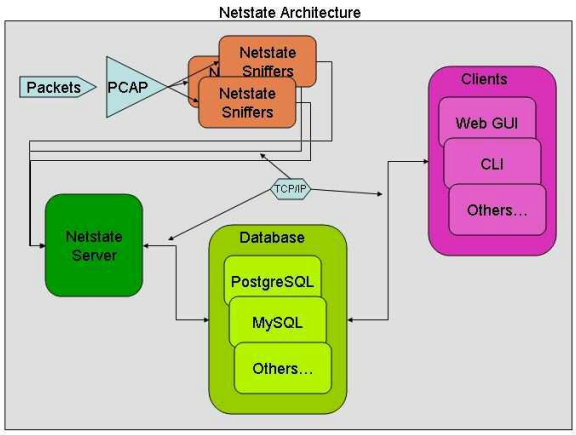


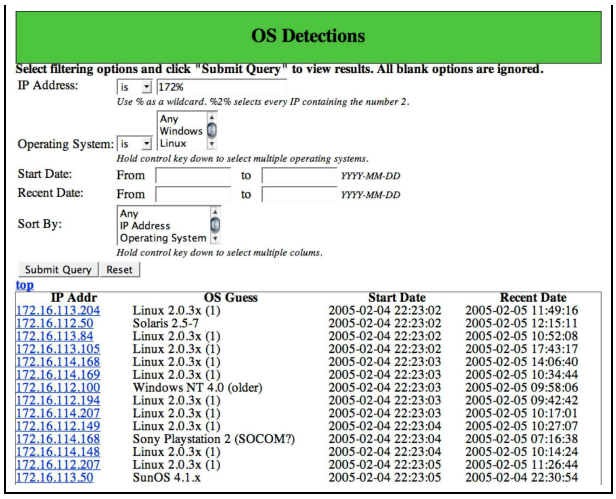
Figure 1: The NetState architecture includes distributed *Sniffers* that capture Ethernet frames and parse them for version information. The Sniffers send this information on to the NetState *Server* which inserts the data into a database. The NetState Web GUI (graphical user interface) and the database CLI (command-line interface) can be used to query the database and retrieve version information.

3.4 NetState Sniffers

The NetState Sniffers are designed to be deployed in many locations on a network. The Sniffers capture network traffic using libpcap [6], the packet capture library available for most UNIX and UNIX-like operating systems. The Sniffers listen passively on a network interface that is given access to all traffic on the to-be-monitored network link, whether by a switch's port-mirroring function, a network tap or some other method.

Operating system detection is performed using the open source program *p0f* (version 2) [7] [Figure 2]. (The data used for our Web GUI figures were PCAP files collected by MIT Lincoln Labs as part of the DARPA Intrusion Detection Evaluation project [8]. We replayed the traffic on a private network using Tcpreplay [9].) We modified p0f to tightly integrate it into NetState, calling it directly as a subroutine. OS detection is performed at the beginning of each new connection, on the synchronize (SYN) packet.

Application version detection is performed by looking at the first few data packets of a connection [Figures 3, 4]. The first data packet is examined for "magic strings" which indicate it likely contains traffic of a specific type. For example the magic string for the FTP and NNTP protocols is the number "202" at the beginning of a line, while the magic string for an HTTP server is "HTTP" at the beginning of a line. If a magic string is found, then further processing is done to find a version string in that packet or from one of the next few packets, depending



OS Detections

Select filtering options and click "Submit Query" to view results. All blank options are ignored.

IP Address: is 172%

Use % as a wildcard. %2% selects every IP containing the number 2.

Operating System: is Any / Windows / Linux

Hold control key down to select multiple operating systems.

Start Date: From to *YYYY-MM-DD*

Recent Date: From to *YYYY-MM-DD*

Sort By: Any / IP Address / Operating System

Hold control key down to select multiple colums.

Submit Query | Reset

top

IP Addr	OS Guess	Start Date	Recent Date
172.16.113.204	Linux 2.0.3x (1)	2005-02-04 22:23:02	2005-02-05 11:49:16
172.16.112.50	Solaris 2.5-7	2005-02-04 22:23:02	2005-02-05 12:15:11
172.16.113.84	Linux 2.0.3x (1)	2005-02-04 22:23:02	2005-02-05 10:52:08
172.16.113.105	Linux 2.0.3x (1)	2005-02-04 22:23:02	2005-02-05 17:43:17
172.16.114.168	Linux 2.0.3x (1)	2005-02-04 22:23:03	2005-02-05 14:06:40
172.16.114.169	Linux 2.0.3x (1)	2005-02-04 22:23:03	2005-02-05 10:34:44
172.16.112.100	Windows NT 4.0 (older)	2005-02-04 22:23:03	2005-02-05 09:58:06
172.16.112.194	Linux 2.0.3x (1)	2005-02-04 22:23:03	2005-02-05 09:42:42
172.16.114.207	Linux 2.0.3x (1)	2005-02-04 22:23:03	2005-02-05 10:17:01
172.16.112.149	Linux 2.0.3x (1)	2005-02-04 22:23:04	2005-02-05 10:27:07
172.16.114.168	Sony Playstation 2 (SOCOM?)	2005-02-04 22:23:04	2005-02-05 07:16:38
172.16.114.148	Linux 2.0.3x (1)	2005-02-04 22:23:04	2005-02-05 10:14:24
172.16.112.207	Linux 2.0.3x (1)	2005-02-04 22:23:05	2005-02-05 11:26:44
172.16.113.50	SunOS 4.1.x	2005-02-04 22:23:05	2005-02-04 22:30:54

Figure 2: These query results, using the NetState Web GUI, show a partial listing of the operating system versions detected on the 172.*.*.* network. The dates indicate when the operating system was first detected and when it was most recently detected.

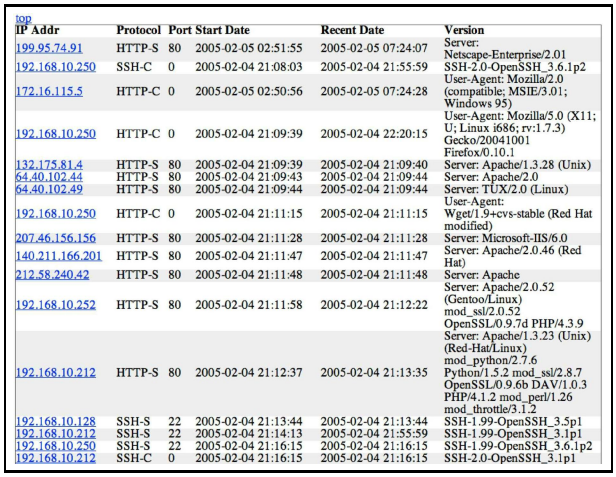
on the specific protocol.

In most cases the version string that is stored in the database for applications is simply the entire string in which the version appears. No attempt is made to pull a numeric value out of a string, because in most cases the format of the version string is not well-defined, but instead, tends to follow common conventions.

In a few cases, e.g. for the file transfer protocol (FTP) and the simple mail transport protocol (SMTP), some implementations append a time/date stamp to the version. Since the timestamp would cause each version string for sessions occurring at different times with the same server to be logged as a new version, this information is stripped off. These sorts of issues need to be discovered and handled on a case-by-case basis.

Currently NetState does not use the port number to infer that a particular application is running. It will find hypertext transfer protocol (HTTP) traffic on any port, FTP on any port, etc. By not using the port number as a "hint", we are more restricted in what applications we can currently detect, but since we want to be able to detect rogue applications on unusual ports, this seemed like the correct design decision.

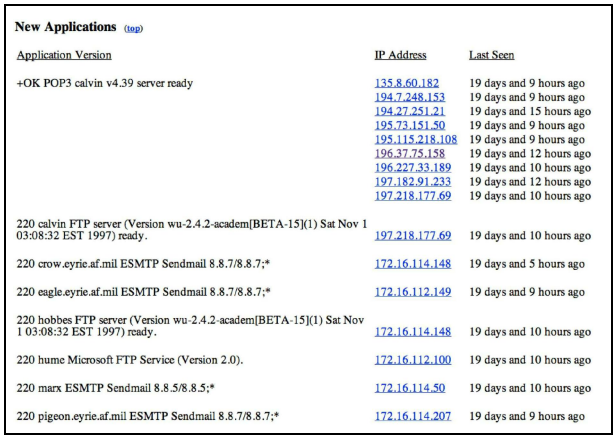
The NetState Sniffer keeps a cache of recently seen version numbers by IP address and port. If a version string is detected that was seen recently, the timestamp is updated internally in the Sniffer but not updated to the Server component right away. A timeout can be configured to control how often the cache updates. This caching feature was added to improve database performance on busy networks by reducing the number of



top

IP Addr	Protocol	Port	Start Date	Recent Date	Version
199.95.74.91	HTTP-S	80	2005-02-05 02:51:55	2005-02-05 07:24:07	Server: Netscape-Enterprise/2.01
192.168.10.250	SSH-C	0	2005-02-04 21:08:03	2005-02-04 21:55:59	SSH-2.0-OpenSSH_3.6.1p2
172.16.115.5	HTTP-C	0	2005-02-05 02:50:56	2005-02-05 07:24:28	User-Agent: Mozilla/2.0 (compatible; MSIE/3.01; Windows 95)
192.168.10.250	HTTP-C	0	2005-02-04 21:09:39	2005-02-04 22:20:15	User-Agent: Mozilla/5.0 (X11; U; Linux i686; rv:1.7.3) Gecko/20041001 Firefox/0.10.1
132.175.81.4	HTTP-S	80	2005-02-04 21:09:39	2005-02-04 21:09:40	Server: Apache/1.3.28 (Unix)
64.40.102.44	HTTP-S	80	2005-02-04 21:09:43	2005-02-04 21:09:44	Server: Apache/2.0
64.40.102.49	HTTP-S	80	2005-02-04 21:09:44	2005-02-04 21:09:44	Server: TUX/2.0 (Linux)
192.168.10.250	HTTP-C	0	2005-02-04 21:11:15	2005-02-04 21:11:15	User-Agent: Wget/1.9+cvs-stable (Red Hat modified)
207.46.156.156	HTTP-S	80	2005-02-04 21:11:28	2005-02-04 21:11:28	Server: Microsoft-IIS/6.0
140.211.166.201	HTTP-S	80	2005-02-04 21:11:47	2005-02-04 21:11:47	Server: Apache/2.0.46 (Red Hat)
212.58.240.42	HTTP-S	80	2005-02-04 21:11:48	2005-02-04 21:11:48	Server: Apache
192.168.10.252	HTTP-S	80	2005-02-04 21:11:58	2005-02-04 21:12:22	Server: Apache/2.0.52 (Gentoo/Linux) mod_ssl/2.0.52 OpenSSL/0.9.7d PHP/4.3.9
192.168.10.212	HTTP-S	80	2005-02-04 21:12:37	2005-02-04 21:13:35	Server: Apache/1.3.23 (Unix) (Red-Hat/Linux) mod_python/2.7.6 Python/1.5.2 mod_ssl/2.8.7 OpenSSL/0.9.6b DAV/1.0.3 PHP/4.1.2 mod_perl/1.26 mod_throttle/3.1.2
192.168.10.128	SSH-S	22	2005-02-04 21:13:44	2005-02-04 21:13:44	SSH-1.99-OpenSSH_3.5p1
192.168.10.212	SSH-S	22	2005-02-04 21:14:13	2005-02-04 21:55:59	SSH-1.99-OpenSSH_3.1p1
192.168.10.250	SSH-S	22	2005-02-04 21:16:15	2005-02-04 21:16:15	SSH-1.99-OpenSSH_3.6.1p2
192.168.10.212	SSH-C	0	2005-02-04 21:16:15	2005-02-04 21:16:15	SSH-2.0-OpenSSH_3.1p1

Figure 3: These query results show the detected network services and their corresponding versions and port numbers on the local network. The "start date" and "recent date" columns indicate when the particular service was first detected and most recently detected, respectively. NetState does not track port numbers for client version strings, so the ports for the HTTP and SSH clients are filled with a placeholder "0".



New Applications (top)

Application Version	IP Address	Last Seen
+OK POP3 calvin v4.39 server ready	135.8.60.182	19 days and 9 hours ago
	194.7.248.153	19 days and 9 hours ago
	194.27.251.21	19 days and 15 hours ago
	195.73.151.50	19 days and 9 hours ago
	195.115.218.108	19 days and 9 hours ago
	196.37.75.158	19 days and 12 hours ago
	196.227.33.189	19 days and 10 hours ago
	197.182.91.233	19 days and 12 hours ago
	197.218.177.69	19 days and 10 hours ago
220 calvin FTP server (Version wu-2.4.2-academ[BETA-15](1) Sat Nov 1 03:08:32 EST 1997) ready.	197.218.177.69	19 days and 10 hours ago
220 crow.eyrie.af.mil ESMTP Sendmail 8.8.7/8.8.7;*	172.16.114.148	19 days and 5 hours ago
220 eagle.eyrie.af.mil ESMTP Sendmail 8.8.7/8.8.7;*	172.16.112.149	19 days and 9 hours ago
220 hobbes FTP server (Version wu-2.4.2-academ[BETA-15](1) Sat Nov 1 03:08:32 EST 1997) ready.	172.16.114.148	19 days and 10 hours ago
220 hume Microsoft FTP Service (Version 2.0).	172.16.112.100	19 days and 10 hours ago
220 marx ESMTP Sendmail 8.8.5/8.8.5;*	172.16.114.50	19 days and 10 hours ago
220 pigeon.eyrie.af.mil ESMTP Sendmail 8.8.7/8.8.7;*	172.16.114.207	19 days and 9 hours ago

Figure 4: These query results show the IP addresses detected to be running a given network service and version number (listed by services). Also included is how long ago that service was last detected. NetState does not currently remove host names from version strings; thus, identical versions of Sendmail appear to be different versions to NetState.

database updates performed by the server component. The result of the caching is that the most-recently-seen time value in the SQL database may not be completely up-to-date at any given time.

The NetState Sniffer maintains information on all active connections, as well as the version cache information, in memory. Its memory fingerprint can be quite large, approaching 512 MB on a busy network (e.g. 1000+ hosts). It loads some configuration files (e.g. the p0f fingerprints) from disk but does not maintain any state on disk.

3.5 NetState Server

The NetState Sniffers capture data off of the network including the application version string, IP address and port number. This three-tuple of information is then sent to the NetState Server. The Server collects this information and writes it to a database along with the current date and time. If the three-tuple creates a new application-version entry, the timestamp is also stored both in a "first-seen" field and a "most-recently-seen" field. If the three-tuple already exists in the database, the Server updates the "most-recently-seen" field with the current timestamp. The database thus stores five-tuple entries containing the application version string, IP address, port number, first-seen timestamp and last-seen timestamp.

The NetState Server is implemented as a daemon listening on a socket on a specific TCP port (the default is 2003) for messages from a Sniffer. If it detects a new connection on the port, it spawns a copy of itself to handle that connection. The Server is implemented as a simple loop that translates messages received from the Sniffer into appropriate SQL database commands to update the database. It does not have any significant memory or disk structures to maintain (other than the SQL database itself).

The database may be located on the same system as the NetState Server, or it may be located on a separate back-end database server. NetState currently supports both the MySQL and PostgreSQL open-source databases. Each of the NetState components was designed to be run on Linux and BSD-derived operating systems. Tested operating systems include *RedHat Linux 9.0*, *Fedora Core 2* and *FreeBSD 4.8*.

3.6 Reporting Interfaces

A Web interface can be used to query the NetState Server for information regarding the service applications on a network. The client includes functionality for several "canned" queries that answer questions including:

- What versions of software (for supported protocols) are currently running on IP x?
- What ports are open on IP x?
- What are all the versions of protocol x (e.g. SSHD) running on network y?
- What IPs on network x are running protocol/version y less than version z (e.g. *OpenSSH* versions $<$ protocol 2)?

In addition to the Web interface, scripts can be written in any language with SQL library support (e.g. *Perl*), to generate reports about the hosts on the network in any desired format.

Some examples of typical SQL queries are shown in Tables 1 and 2. These are the types of queries that can be integrated into a graphical GUI or a Perl script, as desired. The query in Table 1 lists the OS strings from all the machines in the database. The record name is `os_detect`, and the string for the `os_version` field comes from the p0f fingerprint file.

Another example of a useful query is shown in Table 2, which shows the software version for all the machines on the network that are running an HTTP server. Note that in this example, there are several duplicate entries for a particular IP address. This can happen when a web-proxy is being used. In this query, `version` is the name of the database record. The `version` field is the string that was detected by NetState. The `description` field is a mnemonic human-readable field that is determined by NetState. Note that "HTTP-S" refers to "HTTP Server"; we use HTTP-C to refer to the version for the client side. It does *not* mean "secure HTTP" (the *HTTPS* protocol).

Another interesting query is

```
select ip_addr_dot, port, description,
version from version where description =
'HTTP-S' where port != 80;
```

This query will list all the HTTP servers on the network that are not running on the standard port 80.

4 Performance

The first version of NetState did not do any internal caching of version information in the Sniffer component. The information for each version string was handed directly to the NetState Server, where duplicate version strings were handled by updating the "most recent time seen" field in the database record. Performance testing indicated that on a busy network the SQL queries would bottleneck the system. A version cache was added to the Sniffer component to mitigate this bottleneck. The cache works by watching for a version string associated with an IP address that is identical to one that was seen

```
mysql> select ip_addr_dot, recent_date, os_version from os_detect;
+-----------------+---------------------+-----------------------------------------------------------+
| ip_addr_dot     | recent_date         | os_version                                                |
+-----------------+---------------------+-----------------------------------------------------------+|
| 192.168.10.182  | 2003-11-26 15:18:44 | Linux 2.4.2 - 2.4.20                                      |
| 192.168.10.202  | 2003-11-20 17:20:33 | Linux 2.4.2 - 2.4.20                                      |
| 192.168.10.97   | 2003-11-25 14:12:36 | Windows XP Pro, Windows 2000 Pro                          |
| 192.168.10.20   | 2003-11-25 09:33:49 | Windows 98 or Windows 2000 SP4                            |
| 192.168.10.31   | 2003-11-21 15:33:42 | FreeBSD 5.0-RELEASE or Macintosh PPC Mac OS X (10.2. ...|
| 192.168.10.31   | 2003-11-21 15:33:45 | Macintosh PPC Mac OS X (10.2.1 and v?)                    |
| 192.168.10.5    | 2003-11-21 16:55:51 | Windows 2000                                              |
| 192.168.10.4    | 2003-11-25 16:55:16 | Windows 98 or Windows 2000 SP4                            |
| 192.168.10.218  | 2003-11-26 12:00:40 | Windows 2000                                              |
 (...)
| 192.168.10.51   | 2003-11-26 11:24:59 | Windows 98 or Windows 2000 SP4                            |
| 192.168.10.84   | 2003-11-25 16:04:28 | Windows XP Pro, Windows 2000 Pro                          |
| 192.168.10.133  | 2003-11-26 13:25:31 | Windows 2000                                              |
+-----------------+---------------------+-----------------------------------------------------------+
17 rows in set (0.00 sec)
```

Table 1: This query and the corresponding results list all detected IP addresses and their corresponding operating systems. Here we also requested the date and time at which the operating system was last seen.

```
mysql> select ip_addr_dot,port, description, version from version where description = 'HTTP-S';
+---------------+------+-------------+-------------------------------------------------------------+
| ip_addr_dot   | port | description | version                                                     |
+---------------+------+-------------+-------------------------------------------------------------+
| 192.168.10.11 |   80 | HTTP-S      | Server: GWS/2.1                                             |
| 192.168.10.11 |   80 | HTTP-S      | Server: SmallWebServer/2.0                                  |
| 192.168.10.10 |   80 | HTTP-S      | Server: Apache/1.3.28 (Unix)                                |
| 192.168.10.10 |   80 | HTTP-S      | Server: Apache/1.3.26 (Unix)                                |
| 192.168.10.10 |   80 | HTTP-S      | Server: GWS/2.1                                             |
| 192.168.10.10 |   80 | HTTP-S      | Server: SmallWebServer/2.0                                  |
| 192.168.10.10 |   80 | HTTP-S      | Server: Microsoft-IIS/5.0                                   |
| 192.168.10.10 |   80 | HTTP-S      | Server: Barista/3.2.7.0005                                  |
| 192.168.10.10 |   80 | HTTP-S      | Server: ValueAdExpress Server 2.0 UNIX (FreeBSD)            |
| 192.168.10.11 |   80 | HTTP-S      | Server: Squid/2.3.STABLE2                                   |
| 192.168.10.8  |   80 | HTTP-S      | Server: Squid/2.3.STABLE2                                   |
| 192.168.10.10 |   80 | HTTP-S      | Server: Stronghold/2.4.2 Apache/1.3.6 C2NetEU/2412 (Un ...|
| 192.168.10.8  |   80 | HTTP-S      | Server: GWS/2.1                                             |
| 192.168.10.8  |   80 | HTTP-S      | Server: Stronghold/2.4.2 Apache/1.3.6 C2NetEU/2412 (Un ...|
 (...)
| 192.168.10.8  |   80 | HTTP-S      | Server: Microsoft-IIS/4.0                                   |
| 192.168.10.8  |   80 | HTTP-S      | Server: Apache/1.3.27 (Unix) mod_throttle/3.1.2 mod_pe ...|
+---------------+------+-------------+-------------------------------------------------------------+
58 rows in set (0.00 sec)
```

Table 2: This query and the corresponding results list all Web servers seen on the local network. Web servers are recorded as "HTTP-S" to differentiate them from Web clients ("HTTP-C").

"recently" (where "recently" is configurable but defaults to five minutes). In that case the Sniffer does not immediately update the database. The new most-recent time information is cached, and the database is updated later, either by a housekeeping routine or when the Sniffer exits. This caching means that the information in the database will not be as up-to-date as it would be without caching, but the performance increase is substantial. In concrete terms, without this caching, NetState was not able to keep up with the network traffic on our target network (averaging ~7 Mb/s combined inbound and outbound traffic). With the caching, dropped packets were essentially reduced to zero as reported by `pcap_stats()`.

5 Experiences

After running NetState on our internal network for several months, we have already found some useful results. Mainly, NetState is useful for finding out what is really happening on the network and for spotting unusual activity that might not be detected by active scanning. For example, if multiple machines are located behind a NAT (network address translation) device, they will appear to have a single IP address. By monitoring the OS and application versions coming from that IP address, it is easy to detect a NAT device (or a single machine that boots multiple operating systems at different times). This sort of information is useful both because it might be in violation of network security policies and because we might want to identify all machines running a certain OS for patching and vulnerability assessment/remediation. NetState can also detect information about a machine that is used infrequently – such a machine might not even be turned on when an active scanning program is run, but if it is ever booted and communicates on the network, NetState can detect it.

Because NetState does not rely on "known ports" to identify application versions, it can detect services running on unusual ports. These might be unsanctioned HTTP servers, or they could be indications of a compromised machine "phoning home" to the attacker. Active scanning, obviously, can only detect the ports that happen to be open at the time the scan is performed. Attackers often only open ports for very short windows of time. Again, NetState can detect and log this activity whenever it happens to occur.

6 Challenges

As in any project, we were presented with some challenges in the course of our implementation. One challenge was designing a system that could handle tracking application versions over a very large IP address space (i.e. CIDR /16 and larger spaces). This required a large database with capability to hold information on, potentially, thousands of addresses and ports and, sometimes, multiple services per port corresponding to a single IP address because of multiple installed operating systems.

Another difficulty is application version obfuscation. Some network services issue version strings with varying degrees of specificity. Some services do not issue version strings at all, leaving version identification to a process of identifying protocol differences between versions. We do not currently use this technique for application version identification in NetState.

NetState cannot account for the situation created when the Sniffers are located on the outside of a NAT device. The NAT device causes many service versions to appear as if they are associated with one IP address or computer, creating many collisions in the NetState database. Many service versions for one given port can be recorded in a very short period of time causing an administrator great confusion. The solution to this situation, of course, is to design the monitoring architecture in such a way that a NetState Sniffer is behind every NAT device. Knowing where NAT devices are located on one's network is, of course, the most important help for an administrator. This same issue exists when detecting Web browser versions for machines behind a Web proxy server. As mentioned in Section 5 above, this aberrant behavior can be helpful in detecting NAT devices and Web proxy servers on networks, especially when these devices need to be regulated by an administrator in some way.

7 Legal Issues

Some network users may object to software such as NetState because it is a form of monitoring software and has the potential to invade one's privacy. We can appreciate that opinion and can assure users that NetState evaluates and stores data exactly as described previously and does not store data from further down in the data stream. Due diligence requires us to look at the legality of one's corporation, university, or ISP (Internet Service Provider) conducting such "monitoring" as well.

Corporations (such as our laboratory) are legally allowed to monitor their own networks for "business purposes" which could include monitoring for misuse and potential vulnerabilities [10, 11, 12]. In addition, we use banners in local login windows and remote logins to indicate that all network traffic is subject to monitoring. Logging into the system indicates consent to monitoring, though consent is not essential for a company to monitor employee communications. In almost all cases, employees should have no expectation of privacy relating to their network traffic including email (whether a work

account or a personal account), Web-surfing habits, etc. [13]. When using company-owned equipment to access a data network, all network traffic is fair game for corporate snooping.

According to US code, ISPs are allowed to monitor their networks for misuse and potential vulnerabilities as well [14]. An ISP is allowed to "intercept, disclose, or use" the network traffic for the purposes of rendering service and for protecting its property. It can easily be seen that a system used for tracking service versions and thus potential vulnerabilities on an ISP's network, though not on systems owned by the ISP, can be used to ensure a properly functioning network for customers and protecting the ISP's own assets (servers, bandwidth, etc.).

It appears that universities can also sniff network traffic under the same US code section as above. Since most universities provide a "wire or electronic communications service," they can also protect their property using a tool such as NetState.

Employees, customers, students, researchers, etc. may not like that their Internet communications can be watched, but US law appears to allow such actions. Again, our tool does nothing more than watch for and record network service version information. Nevertheless, we remind users to deploy encrypted network applications or to tunnel their applications over an encrypted link for true data confidentiality.

8 Future Work

We would like to extend NetState to detect version strings for more network services. Eventually we would hope to have a list containing regular expression-based signatures for version strings so that we can easily add more detection capability. This could be similar to the signature file used by the open source intrusion detection system, Snort.

As mentioned earlier, the Nmap scanning tool has the capability to actively probe open ports for service and version information. It would be easy to quickly populate the NetState database upon initial installation using this feature of Nmap. NetState could piggyback on an organization's routine scanning activities to aid in database population as well.

Because NetState learns about service versions passively, it cannot learn information about specific software versions being run inside of SSL connections. Nmap invokes OpenSSL when it discovers an SSL-enabled service and then initiates further probes to obtain version information. We may add a module to NetState that invokes Nmap when SSL-enabled ports are discovered, storing those results in the NetState database.

NetState is a query-based tool. In other words, a network/security analyst will not get information out of NetState if he/she does not specifically ask for it. We would like to build a small set of signatures that constantly look for service version anomalies and automatically notify appropriate personnel. For example, we would like to know in a short amount of time if an OpenSSH service version changed to an earlier version than was last known.

Our current design using passive sniffing could aid in performing network-based anomaly detection in the future. Since the anomaly detection data would come from current traffic that was scanned passively, it can be directly compared to the data from NetState – i.e. the data will contain the same type of information. We think this will make the anomaly detection task more tractable.

We have begun to study creating network profiles for each device on our network using NetState. Because we have Sniffers placed in many strategic locations, it is easy to record and store information about the typical network traffic patterns seen from each network device. We have experimented with storing information about each session that a device establishes that terminates with hosts outside our networks. After enough time building a database of session data, we hope to extend the NetState Server so that it detects anomalies in network traffic between hosts.

9 Acknowledgements

We would like to thank Tim Toole, Tristan Weir, Archer Batcheller, Kami Vaniea, and Eric Thomas for their contributions and insight into this project. Special thanks goes to Randy McClelland-Bane for his hard work gathering screenshots and data for our consumption.

References

[1] Fyodor. (2005) Nmap. Insecure.org. [Online]. Available: http://www.insecure.org/nmap/

[2] eEye Digital Security. (2005) Iris network traffic analyzer. [Online]. Available: http://www.eeye.com/html/products/iris/index.html

[3] Javvin Company. (2005) Network packet analyzer. [Online]. Available: http://www.javvin.com/packet.html

[4] Novell Inc. (2005) Zenworks suite. [Online]. Available: http://www.novell.com/products/zenworks/

[5] Renaud Deraison. (2005) Nessus open source vulnerability scanner project. Tenable Network Security. [Online]. Available: http://www.nessus.org/

[6] Lawrence Berkeley National Laboratory Network Research Group. (2005) libpcap. [Online]. Available: http://ftp.ee.lbl.gov/nrg.html

[7] Michal Zalewski. (2005) p0f v2. [Online]. Available: http://lcamtuf.coredump.cx/p0f.shtml

[8] Darpa intrusion detection evaluation data. MIT Lincoln Laboratory. [Online]. Available: http://www.ll.mit.edu/IST/ideval/data/1999/training/week3/

[9] Aaron Turner and Matt Bing. (2005) Tcpreplay. [Online]. Available: http://tcpreplay.sourceforge.net/

[10] (2004) Workplace privacy. EPIC. [Online]. Available: http://www.epic.org/privacy/workplace/

[11] (2002) Fact sheet 7: Workplace privacy. Privacy Rights Clearinghouse. [Online]. Available: http://www.privacyrights.org/fs/fs7-work.htm

[12] Karen L. Casser, "Employers, employees, e-mail and the internet," in *The Internet and Business: A Lawyer's Guide to the Emerging Legal Issues*. Computer Law Association, 1996. [Online]. Available: http://www.cla.org/RuhBook/chp6.htm

[13] (2003) Fact sheet 18: Privacy in cyberspace. Privacy Rights Clearinghouse. [Online]. Available: http://www.privacyrights.org/fs/fs18-cyb.htm

[14] United States Federal Government, "Interception and disclosure of wire, oral, or electronic communications prohibited," in *US Code, Title 18, Part I, Chapter 119,* §*2511*, 2004. [Online]. Available: http://www.law.cornell.edu/uscode/html/uscode18/usc_sec_18_00002511----000-.html