Security Event Management –

Staying on track and not missing the forest for the trees

A GIAC Practical Assignment

Ilango S Allikuzhi

June 29, 2004 Revision 002

Table of contents

Security Event Management – Staying on track and not missing the forest for the trees

Abstract

The purpose of this paper is to share some of my experiences, setting up and managing a security event management infrastructure on a large private network that connects GSEC Inc. to its clients and partners. This talks about our experimental venture into consolidating the logs and getting an insight into troublesome events, the roadblocks encountered, and pitfalls and successes seen.

I would like to express my sincere thanks to our mentor, Beth Binde of Rutgers University for her guidance.

The network had a number of firewalls and intrusion detection systems already deployed. We embarked on a lofty mission of setting up a central security event management infrastructure that deals with OPSEC events, Syslog messages, SNMP events and other log encapsulation methods to support 24x7 Security Operations Center.

The management was particularly keen that we leveraged the existing network management infrastructure. Besides providing a central event console in the Network Operations Center (NOC) which doubles up as a Security Operations Center (SOC), we are required to notify appropriate support personnel of critical events through automated paging system, analyze and provide meaningful reports based on access-list violation logs of packet-filtering routers and intrusion detection systems' alerts.

The existing network management infrastructure is built on a protected network – the Management VLAN is isolated from the campus network and from the access VLAN, which connects the customers to GSEC Inc' s data centers. The management network is protected from access network and the campus network through CheckPoint Firewall-1 enforcement modules. Needless to say only essential services such as ICMP, SNMP (161/udp), SNMP trap (162/udp), TACACS+ (49/udp), NTP (123/UDP), TFTP (69/udp), DNS (53/udp), Telnet (23/tcp), SSH (22/tcp), FTP (20/tcp and 21/tcp) are allowed into specific servers on the management VLAN in the appropriate direction.

The management servers – Central Syslog server, Openview Network Node Manager, CiscoWorks 2000 server, CiscoSecure/ACS TACACS+ server, Micromuse Netcool/Omnibus Object Server, Checkpoint Management Server, SourceFire Management Console, and Concord NetHealth server as well the management interface of the SourceFire network sensor appliances, SourceFire RNA (Real-time Network Awareness) appliances and firewall enforcement modules are on management VLAN. The sniffing interface of the network sensors and RNA are on the access VLAN to capture the production network traffic between GSEC Inc. and its clients over the private network. There are firewall rules in place to allow specific workstations on campus network to access services such as SSH (22/tcp), Telnet, FTP and specific HTTP/HTTPS ports on specific servers on the management VLAN. This is primarily for administrators and users to access specific hosts and applications.

This private network connects over hundreds of customers to GSEC Inc.'s data centers. This infrastructure involves a number of core routers and hundreds of edge routers with extensive deployment of standard and extended access-lists in Cisco IOS environment. We have at our disposal network management platforms and tools such as HP Openview Node Manager 6.2, CiscoWorks 2000 Resource Manager Essentials 3.5 with syslog analyzer and Micromuse Netcool/Omnibus 3.6 - all deployed on Solaris 8 servers in our management VLAN. Netcool/Omnibus plays the role of Manager of Managers (MoM); it provides event correlation and de-duplication capabilities and browser-based operator consoles (Netcool Webtop). Netcool/Omnibus ObjectServer stores the events in its memory-resident database and the events are pushed to a backend SQL database on regular basis through Netcool/Omnibus ODBC gateway. As there is expertise available in-house to customize rules and build correlation logic, we seemed to lean towards deploying Netcool/Omnibus as our security event management platform.

The Problem that we are trying to address

The IP access-lists are deployed on the core routers and edge routers to achieve the following objectives:

- Ingress and egress filter
- Telnet access filter
- Snmp-server access filter
- TFTP Access filter
- Specific application services access filter

Intrusion detection engines (IDS) deployed on the access network capture and analyze the production network traffic and generate alerts. The IDS alerts in the form of Syslog messages are in turn sent to central syslog servers, which also receive Syslog messages from the routers. SourceFire IDS provides an excellent web interface for real-time events and reports and better yet, when you consolidate all SourceFire events on a central management console, you can view correlated events on the Management Console through a web interface. But our management was particularly keen on getting these events on the central console on the SOC as the staff would not need any additional training. Hence we decided on deploying Netcool/Omnibus Syslog probe on the central Syslog server and Firewall-1 probe on the CheckPoint Firewall Management modules to enable real-time notifications through console.

We almost decided to consolidate the security events from SourceFire Intrusion Detection Systems (IDS), CheckPoint FW-1 NG Firewalls, CiscoSecure/ACS TACACS+ and Cisco IOS devices (access-lists denial logs, in particular) to Netcool/Omnibus platform. Micromuse does offer a security platform offering, called Netcool for Security Management (NfSM), to address the SIMS market but we decided to implement security event management around the standard ObjectServer platform, using appropriate probes as this would help us leverage our existing infrastructure and avoid incurring additional expenditure on the platform.

The concern we had was that most of these devices are chatty in nature and we would run the risk of flooding the central console so badly that it would be no more be useful to the operators. We were sure the Netcool/Omnibus Object Server could handle very high message rate as it relies on memory-resident database versus traditional SQL database but we were contemplating whether we should have just one ObjectServer handling them all or we should build a multi-tier ObjectServer architecture. After a lot of deliberations, we decided to experiment with a simple single-tier ObjectServer architecture, which proved to be adequate during the implementation phase.

Unlike network fault management events, it is not easy to define “actionable” security management events. It was a real challenge to come up with a list of critical events, which would go to the console. Moreover we had to come up with a policy as to what events should be handled out of real-time console and what should be handled out of scheduled reports.

Unless the rules are very carefully fine-tuned, there is always a chance of flooding the console with messages and rendering it useless. We wanted to put only operator actionable events on the console and the rest of the informational events would be viewed only from point-solution consoles like SourceFire Management Console or CheckPoint Management Console.

We could always build separate views for IDS, Firewalls, ACS and IOS Access logs with filters and make sure that we create separate console displaying different views, using Omnibus.

While Netcool/Omnibus does an excellent job of managing the Syslog, SNMP and OPSEC events on real-time basis with a state-based console, we realized we are still required to develop systems in-house to do a detailed analysis of these security logs to figure out how much of it is mere noise - broadcast and protocol traffic, what kind of services are being denied, and who are the clients who are being denied access. It is not practical to manage this deluge of events out of a real-time console and hence one needs to deal with them through scheduled daily reports.

We find that roughly about 350,000-500,000 access list related (%SEC-6-IPACCESSLOG*) syslog messages are generated on daily basis, which detail access denied by the packet filtering routers and it is required to understand what is causing these.

Missing the forest for the trees!!!

After detailed analysis, we discovered that on average, 97.45% of access log messages were either mere broadcast traffic or the traffic between customer nodes on the remote ethernet segment on the edge routers. This kind of traffic can easily be ignored. To put it in simple words, the need of the hour is to ensure that *we don't miss the forest for the* *trees*. It is required to analyze only the remaining 2.55% of the access violation logs very carefully for possible intrusion attempts.

With logs being so large and even the daily event summary we produced being so long that it was impossible for any administrator to analyze the report on daily basis to take any corrective action. So it was decided to remove the noise such as broadcast, multicast, the LAN traffic on the remote side to generate more meaningful report of genuine access attempts that were denied by the access-lists.

The SourceFire IDS engines were coughing tons of false positives and we started our exercise with about 160,000 events a day from our primary IDS engine on our access VLAN. We were simply drowned in these events while our management started pushing us to forward these alerts to a central management console for operators to monitor these events. This would have made the existing management consoles completely unusable. We had to decide what events should go to a central console and what should not. Before setting up event forwarding, we had to inspect all the existing alerts and weed out the "False Positives". We had to struggle for more than 3 months to identify all the false positives and apply suppression rules to them and also implement throttle logic to other events to bring down the daily alerts volume down to few hundreds.

The bulk of our "False Positives " were generated by our management traffic, which we could easily suppress. There were other tricky "False Positives" triggered by snort rules for the DLSw+ traffic between the core and edge routers.

Now we are ready to pass the network sensor events to the SourceFire management console as well as forward snort syslog alerts to our network management platform to display on NOC consoles. I personally don't believe an operator on a console can react to these alerts in a meaningful way, but we had to do it as mandated by the management. I would rather look at SourceFire management console for events a few times a day and spend some quality time analyzing them to figure out what they really are, what caused them, and if they should be suppressed in future. In any case, we had to deal with this problem at the source (network sensors) rather than at the central console. We had to normalize the snort alerts the same way the Omnibus/Firewall-1 probe normalizes the Firewall-1 events to do any event correlation in future.

Firewall-1 probe did its part of the damage in terms of generating a deluge of abnormal condition messages such as "port overuse", and "protocol overuse" warnings for management traffic and legitimate application traffic and we were once again caught up amidst the trees and we had to get past them to see the real problems. Again getting rid of alerts due to management traffic was simple; we had to define management network variable and discard all alerts for known protocols. The rest of legitimate traffic generated "abnormal" firewall messages in large numbers and it was taking up enormous time to deal with this menace. These false positives were really threatening to ground the project and we knew there is nothing like weeding them at the source, the firewall probe rules.

The Solution and its Implementation

We set up a centralized syslog server on our primary and contingency site to handle this. We built it on a SunFire 280R server running Solaris 8 with all recommended and security patches applied.

Our daily volume of syslog messages from routers and IDS at this point of time is around 500,000 messages on average and in other words, about 40-60 MB of data. As mentioned earlier %SEC-6-IPACCESLOG messages from Cisco routers alone contributed nearly two-third of the daily syslog volume.

As mandated by regulatory authorities, we have to maintain the logs for a minimum of 6 months time and about 3 months of logs are expected to be available online on the syslog server.

As syslog messages are logged in different files in the /var file system, it is imperative that /var is mounted on the separate partition which is sufficiently large. We ensured that the /var file system was over 18GB and another file system of capacity of 20 GB, /var1 was created on another spindle for archiving syslog messages, SNMP traps and router configurations for 3 months. The archived router configuration has bailed us out of a lot of difficult situations when routers configurations need to be restored to a previous state.

We also transfer the logs from /var1 directory onto a PC workstation on regular basis using scp (Secure Copy, using port 22/tcp) and archive them on WORM media along with md5 hash.

Syslog files are maintained as per standards recommended by CERT. The local host's syslog messages are sent to /var/adm/messages, while the remote syslog messages are sent to /var/log/messages.

The following document provides an excellent description of the best practices in configuring syslog.

http://colin.bitterfield.com/Syslog_for_the_datacenter.html

/etc/syslog.conf is configured in such a way that all router syslog messages get into /var/log/messages.

Here is the syslog configuration we use.

```
#--------------------------------------------------------------------------------
/etc/syslog.conf
#ident  "@(#)syslog.conf       1.5     98/12/14 SMI"   /* SunOS 5.0 */
#
# Copyright (c) 1991-1998 by Sun Microsystems, Inc.
# All rights reserved.
#
```

```
# syslog configuration file.
#
# This file is processed by m4 so be careful to quote (`') names
# that match m4 reserved words.  Also, within ifdef's, arguments
# containing commas must be quoted.
#

*.err;kern.debug;daemon.notice;mail.crit        /var/adm/messages

*.alert;kern.err;daemon.err                     operator
*.alert                                         root

*.emerg                                         *

# Added for Cisco Syslog (begin)
local6.info;local6.debug                                /var/log/messages
local7.info;local7.debug                                /var/log/messages
#
local0.emerg;local0.alert;local0.crit;local0.err;local0.warning;local0.notice;local0.info;local0
.debug  /var/adm/dmgtd.log
#
#-------------------------------------------------------------------------------
```

If access-lists log (%SEC-3-IPACCESSLOGP) message volume gets too big, it may be worthwhile writing ACL logs into a separate file. In order to split these messages to separate log files, we can modify syslog.conf as follows:

```
#-------------------------------------------------------------------------------
#
# All LOCAL7 messages  (debug and above) go to the ciscoacl
# This includes ACL logs, which are logged at severity debug
#
local7.debug /var/log/cisco/ciscoacllog
#
# LOCAL7 messages (notice and above) go to the ciscoinfo
# This excludes ACL logs which are logged at severity debug
#
local7.notice /var/log/syslog_info
#
#-------------------------------------------------------------------------------
```

Since the logs grow extremely large in size, it is essential to implement a log rotation scheme to keep the logs within manageable size and to help streamline the archival process. The log rotation is set up so that the logs are rotated every day and the rotated logs are kept for 7 days in the /var/log directory in compressed form (gz). The previous day’s syslog file, messages.1 is copied under “/var1/log/" as 'messages-MMDDYY.gz' and kept online for 3 months. Please note we rotate only /var/log/messages file on daily basis and /var/adm/messages is never rotated.

We prefer using perl Logfile::Rotate as opposed to Sun’s newsyslog script as this works across all our Unix platforms.

The following crontab entry is added to ensure syslog rotation

```
59 23 * * *  /usr/local/bin/logrot2
```

We find about 8:1 compression is achieved by compressing syslog files with gzip and it really helps conserve disk space. So our syslog archives are in .gz format.

One downside of rotating the logs is that we run into issues when we run scripts to track dial-backup ISDN calls and there are many instances we come across an ISDN call closure event for a call which was actually established the previous day. We have to uncompress the previous day's log and merge it with the current log to do analysis of this kind.

The log rotation script generates MD5 hash and writes it into another file (messages.date.gz.md5) to be archived along with the syslog file (messages.date.gz) in the archive directory. This file contains the hash for the compressed (GNU zip format, of .gz extension) syslog file and any one accessing this file at a later date would be able to verify its authenticity by generating md5 hash once again and comparing it with the original hash stored in the file.

Here is the content of a sample hash file

```
MD5 (/usr1/log/messages.060204.gz) = 3a16fd3c046130994fd898bf96803582
```

Jim Ellis (jte@cert.org) provides an excellent description of MD5 hash in the following URL
http://www.dsinet.org/tools/crypto/md5/MD5.README

The log rotation scheme ensures last 7 day’s compressed logs are kept in the /var/log directory on any given day.

```
#-------------------------------------------------------------
#!/usr/local/bin/perl

use File::Copy;
use Logfile::Rotate;
use POSIX 'strftime';
# rotate syslogs
   $log = new Logfile::Rotate( File  => '/var/log/messages',
                               Gzip  => '/usr/bin/gzip');
# process log file
```

```
   $log->rotate();
   undef $log;

#archive the day's syslog in the /var1 directory
$date  = POSIX::strftime("%D",localtime());
$fdate = $date;
$fdate =~ s/\///g;
$orig_syslog = "/var/log/messages.1.gz";
$arch_syslog = "/var1/log/messages.$fdate.gz";
$hash_syslog   = "/var1/log/messages";
copy($orig_syslog,$arch_syslog) || die("FATAL: copy $orig_syslog: $!\n");
system("/usr/local/bin/md5 $arch_syslog > $hash_syslog");


#-----------------------------------------------------------------------------
```

We ensured all unwanted services are disabled on the syslog server, including telnet and FTP. You can disable services handled by inetd daemon by editing /etc/inetd.conf. Also shut off all unwanted, vulnerable services like nfs client, nfs server; these are started by the startup scripts in etc/rc*.d directories which have to be disabled by renaming the startup scripts. In other words, simply move /etc/rc3.d/s15nfs.server startup script file as /etc/rc3.d/DISABLED.s15nfs.server to disable nfs server completely.

Solaris installation puts Sun SNMP agent and DMI agent, which can be completely shut off by removing the startup scripts. If you do not want to remove them but want to properly configure the community strings to make them secure, refer the following document.

http://ist.uwaterloo.ca/security/howto/2000-10-04/

OpenSSH is set up for administrative access to the Syslog server and all file transfers are done using scp. We installed OpenSSH and added /etc/init.d/sshd startup script to start and stop secure shell daemon. /etc/allow.hosts currently allows access to specific hosts. OpenSSH daemon is configured only with Protocol 2 and in other words, the susceptible Protocol 1 is completely disabled and related keys have been removed. If public key authentication is not possible, it falls back to password authentication, but it sends out only encrypted password.

```
#----------------------------------------------------------------------------
/usr/local/etc/sshd_config

#$OpenBSD: sshd_config,v 1.65 2003/08/28 12:54:34 markus Exp $
# This is the sshd server system-wide configuration file.

Port 22
Protocol 2
# For protocol version 2
HostbasedAuthentication no
```

```
IgnoreRhosts yes
PasswordAuthentication yes
PermitEmptyPasswords no
#---------------------------------------------------------------------------
```

Cisco routers have been configured to send the syslog messages to the central syslog server as follows:

```
logging history informational
logging source-interface Loopback0
logging  a.b.c.d1     <------ IP address of syslog server
logging  a.b.c.d2     <-------|
ntp server a.d.c.e1 prefer
ntp server a.d.c.e2
```

The Essence of Synchronized Time

As we consolidate logs on to a single server, the notion of accurate time is essential to determine the order in which the network events occurred. It is imperative to implement Network Time Protocol (NTP) to ensure the timestamp on the logs are accurate to do any useful analysis.

The top NTP hierarchy for the private network consists of core routers, fcore1, fcore2, fcore3, and so on - which are NOT synchronized directly with external servers but with hosts in the protected network, which in turn are synchronized with external source.

```
Router fcore1
ntp server a.d.c.e1 prefer
ntp server a.d.c.e1
```

Additionally ntp peers can be set up so that time remains synchronized even if the connectivity with the master source is lost.

```
Router fcore1
ntp peer fcore2
ntp peer fcore3
```

Edge routers located at the customer premises are configured to use the core routers for NTP synchronization:

```
Router remote1
ntp server fcore1
ntp server fcore2
ntp server fcore3
```

The management server also points to the core routers for the NTP source. Apparently there are some known NTP vulnerabilities as detailed below:
http://www.cisco.com/warp/public/707/NTP-pub.shtml

There is an excellent reference on how to setup NTP securely on Cisco routers, including access-list implementation for NTP
Hardening Cisco Routers By Thomas Akin
http://www.oreilly.com/catalog/hardcisco/chapter/ch10.html

Sun provide a detailed blue print, which explains the NTP implementation on Solaris servers.
http://www.sun.com/blueprints/0701/NTP.pdf

Analyzing all the logs that we have collected - the difficult part

As far as access-lists logs are concerned, the following logic is used arriving at the access list log summary

Discard the entries pertaining to broadcast packets

Over 60% of %SEC-6-ACCESSLOGP messages appear to be broadcasts.

Major contributor of this is the NETBIOS broadcasts generated by Windows systems on the customer's network (which enter Interface Ethernet 0 of edge routers).

These are harmless irritants, generated due to carelessness or ignorance on the part of Windows administrators, not to have disabled NETBIOS on hosts on a network segment that connects to another organization's network. These entries are safely ignored.

Next discard the entries pertaining to communication between client's nodes on the Ethernet segment. We can safely assume 24-bit network mask, and discard access log entries if the traffic is between hosts on the client network (which connect to the network through interface Ethernet 0 on the edge router). If the first 3 octets of the source and destination IP addresses match, and if the address is a public address, then we can safely discard these messages

This constitutes roughly another third of the access-list syslog messages.

Once we remove these, we are left with about 5000 messages of which 15-20% are protocol traffic being denied for valid reasons.

Once we discard the protocol traffic, we are actually left with about 4000+ messages to deal with.

Please note that all public addresses have been changed to imaginary addresses.

The following is an example of kind of broadcast packets denied by access-lists; these events fill up the log to the extent of over 95%

# The following entry is a NETBIOS broadcast from a client workstation being blocked by our edge router

```
May 16 23:40:19 rtr38sa 163590: May 16 23:38:34.870 EDT: %SEC-6-IPACCESSLOGP: list 101 denied udp a.b.c.211(137) -> a.b.c.255(137), 1 packet
```

# The following is a RIP (Routing information Protocol V2) routing protocol broadcast, which can be ignored

```
May 16 23:40:19 rtr432sa 327923: May 16 23:38:11.802 EDT: %SEC-6-IPACCESSLOGP: list 101 denied udp a.b.d.65(520) -> a.b.d.255(520), 7 packets
```

# The following is an example of udt_os broadcast from a client on the remote side

```
May 16 23:40:19 rtr526sa 572329: .May 16 23:38:12.323 EDT: %SEC-6-IPACCESSLOGP: list 101 denied udp a.c.b.197(7331) -> a.c.d.255(3900), 2 packets
```

# The following is another example of NETBIOS irritant

```
May 16 23:40:15 rtr58sa 20366565: May 16 23:38:31.678 EDT: %SEC-6-IPACCESSLOGP: list 101 denied udp x.y.z.251(138) -> x.y.z.255(138), 1 packet
```

So we are finally getting past the trees and getting a glimpse of the real forest!!!

I use a port table file to convert the ports seen in the logs to service names. We can create this port table containing services for TCP/UDP ports, by downloading from IANA site,

http://www.iana.org/assignments/port-numbers

We created a file containing services for all well-known and registered ports. What if our organization's in-house applications use some of these registered ports for some other purpose? The port table downloaded would not provide me with the TCP/UDP ports used by our in-house applications, which are neither well-known nor registered ports. So we gather data on in-house applications and add these entries to the port table at the very end. Finally we parse these port table entries and load them into an associative-array (perl mongers, call this hash). In the hash, %services, we would have all ports pointing to the appropriate service.

Now we can parse the access logs with Perl and create a summary report detailing each rejected transaction with count, each denied service with count of attempts, each denied source IP address with count, or each denied destination IP address with count. Assuming that there are many repeats of a particular type of denied transaction, we would finally end up with an event report of about 500-1000 lines that details all denied transactions with the appropriate service name against them and the count. It's still a large report and people can easily miss an important rejected transaction or al least they complain they can do so.

The forest is not easy to conquer!

We can't possibly reduce the number of transactions being reported but we can alter the presentation of them so that the critical transactions show up on top.

First we list packets for well-known ports (which would include most critical transaction types, such as, telnet, ftp, http, https et al.), followed by rejected transactions for ports used by our in-house applications and then rest of the registered and non-registered ports.

This report is generated every day, archived under /var1/reports and kept online for 6 months.

Here is a sample report.

```
*** Rejected connections
=====================================================================================
 Port/proto   Service        Source -> Destination          ACL      Router    Attempts
=====================================================================================

 111-tcp      sunrpc         a.b.c.1->a.b.d.229             list 101  rtr958sa      1
 111-tcp      sunrpc         a.b.c.1->a.b.d.229             list 101  rtr958sa      1
 113-tcp      auth           a.d.250.34->a.d.244.0          list 101  rtr958sa      1
 123-udp      ntp            a.e.229.10->a.k.10.3           list 101  rtra12s       148
 135-tcp      epmap          a.u.2.181->a.ss.32.65          list 101  rtri66sa      31
 135-tcp      epmap          a.qq.84.93->a.q.28.229         list 101 rtr958sa       1
 135-tcp      epmap          a.dd.81.78->a.q.28.229         list 101  rtr958sa      2
 139-tcp      netbios-ssn    a.ff.212.137 ->a.d.244.0       list 101  rtr958sa      1
 162-udp      snmptrap       a.i.77.34->a.w.o.130           list 101 rtr206sa       2
 21-tcp       ftp            a.dd.p.136 ->a.ss.34.1         list 101  rtr310sa      1
 21-tcp       ftp            a.u.p.137 ->a.ss.34.1          list 101 rtr310sa       10
 21-tcp       ftp            a.u.207.145 ->a.h.19.151       list 101  rtrj78sa      2
 21-tcp       ftp            a.i4.157.246->a.h.19.151       list 101  rtrj78sa      2
 22-tcp       ssh            a.p.185.198->a.mx.61.118       list 101  rtre54sa      1
 25-tcp       smtp           a.ee.5.39 ->a.mx.61.118        list 101  rtre54sa      1
 443-tcp      https          a.e8.140.7 ->a.s.89.15         list 101  rtrg62sa      1
 443-tcp      https          a.e8.140.7 ->a.s9.93.119       list 101  rtr970sa      1
 443-tcp      https          a.o.253.197->a.h.19.151        list 101  rtrj78sa      2
 445-tcp      microsoft-ds   a.l.8.85->a.q.28.229           list 101  rtr958sa      2
 80-tcp       www-http       a.w.203.84->a.q.28.229         list 101  rtr958sa      1
 80-tcp       www-http       a.zz.59.192->a.mn.134.30       list 101  rtrf74sa      5
 80-tcp       www-http       a.zz.159.192->a.mn.197.59      list 101  rtrf74sa      4
 80-tcp       www-http       a.f.65.46 ->a.mx.61.118        list 101  rtre54sa      2
 80-tcp       www-http       a.y.126.212->a.mx.61.118       list 101  tre54sa       2
 80-tcp       www-http       a.i3.82.28->a.h.19.151         list 101  trj78sa       2
….
…
=====================================================================================

***Service denials:

|       epmap       135-tcp      683
|       epmap       135-udp      9
|       ftsrv       1359-udp     1
|     timeflies     1362-udp     1
|    netware-csp    1366-tcp     1
|    netbios-ns     137-udp      272
|       fc-cli      1371-tcp     2
|                   13714-tcp    1
|    chromagrafx    1373-udp     1
|    netbios-dgm    138-udp      14
|  telesis-licman   1380-udp     1
|        gwha       1383-tcp     1
|        gwha       1383-udp     1
|    netbios-ssn    139-tcp     31
|     iclpv-sc      1390-tcp    2
|      igi-lm       1404-tcp    2
|       hiq         1410-tcp    2
|       hiq         1410-udp    1
|    gandalf-lm     1421-tcp    1
|      essbase      1423-udp    2
|      mloadd       1427-udp    1
|       rgtp        1431-udp    1
|     ms-sql-s      1433-tcp    16
|     ms-sql-m      1434-udp    28
|      https        443-tcp     4
|                   4431-udp    1
|                   4432-udp    1
|                   4442-udp    1
|   microsoft-ds    445-tcp     365
|                   4462-tcp    2
|     www-http      80-tcp      138
|      irdmi        8000-tcp    4
|                   8015-udp    1
|      pro-ed       8032-udp    1
|                   8059-udp    1
|     http-alt      8080-tcp    3
|                   8090-udp    2
```

Taming the Firewall-1 OPSEC Events

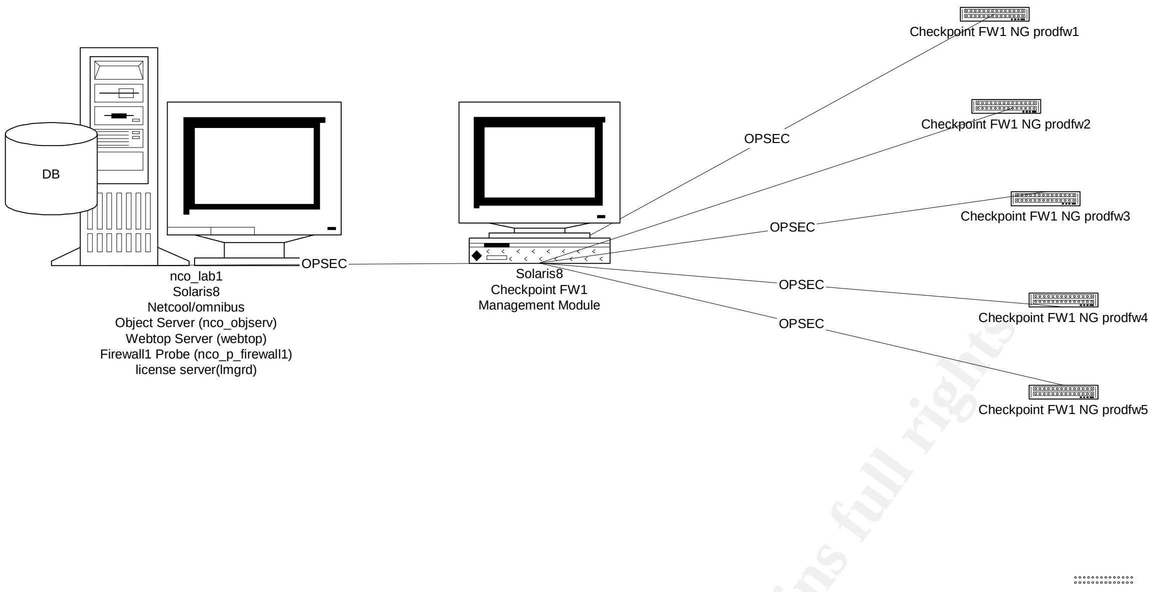
We were really keen on choosing a robust, secure solution to handle firewall events. In other words, we did not want to rely on traditional mechanism like syslog or snmp. As we had Checkpoint Firewall-1 NG already deployed enterprise-wide and Micromuse Netcool/OMNIBUS already in place as a platform for network management, we considered Netcool Firewall-1 probe, among other solutions.

We are quite convinced about Netcool's scalability and its event correlation capabilities. We were confident that it could handle very high burst rate of events without any problems, thanks to its memory-resident database architecture. The management preferred a firewall event management solution, which is OPSEC-certified and a vendor who has strong relationship with Checkpoint as a member of OPSEC Alliance. As our Proof-of-Concept exercise with Netcool Firewall-1 probe proved to be quite successful and the probe met most of our objectives, we decided to deploy Netcool Firewall-1 probe in a pilot project, covering 5 enforcement modules. This eliminated the learning curve on the management platform and we had to only learn the probe module.

I suppose the same results could be achieved securely with syslog-ng implementation, but we were to implement this using a vendor product and hence we ventured into setting up Netcool Firewall-1 probe on our Netcool management server. We were not too comfortable putting the probe on the Firewall management server itself which would have been ideal as we would be trapping the events locally.

You would encounter a lot of Checkpoint Firewall-1 jargons in rest of this section. So let me define some of these terms first.

Enforcement modules are used to monitor and apply the Checkpoint Firewall-1 security policy. Whenever the enforcement module detects a session, it generates an event, which is then forwarded to the management module.


Checkpoint FW1 NG prodfw1
Checkpoint FW1 NG prodfw2
Checkpoint FW1 NG prodfw3
Checkpoint FW1 NG prodfw4
Checkpoint FW1 NG prodfw5
OPSEC
OPSEC
OPSEC
OPSEC
OPSEC
DB
nco_lab1
Solaris8
Netcool/omnibus
Object Server (nco_objserv)
Webtop Server (webtop)
Firewall1 Probe (nco_p_firewall1)
license server(lmgrd)
Solaris8
Checkpoint FW1
Management Module

The management module logs all of the connections across many enforcement modules. The Firewall-1 probe connects to the management module, using LEA and the connection is authenticated. OPSEC provides the authenticated connections between OPSEC clients and server, which reduces the risk involved. Typically the management module plays the role of OPSEC server while the Firewall-1probe is the client.

When the Firewall-1probe is started, it reads the following configuration files in that order:

- Properties file
- Rules file
- Management file
- LEA configuration
- Policy file
- Abnormal behavior definition file

These configuration files control the behavior of the probe.

Properties file ($OMNIHOME/firewall/firewall1/firewall.props) needs to be customized to suit the environment. This defines the location of all configuration files including the rules file.

The Firewall-1 probe requires some additional fields in the Object Server schema. So the default Netcool/OMNIBUS database schema should be extended using the modified schema provided in the $OMNIHOME/firewall/objectserver.

Known Offenders file lets you handle known offenders differently. Whenever a source or destination matches the IP address in the known offenders file, the probe generates an event.

Probe Policy file is used to classify events from Firewall-1 enforcement modules.

LEA Server is a process running on the Firewall-1 management module. LEA configuration file defines the location of the management module host machine and contains the port number and IP address of the management module.

LEA Configuration File (firewall1.lea.conf) is used for authentication between probe and management module

#opsec_sic_name "CN=opsec_application_name,O=org_code"

opsec_sic_name "CN=NETCOOL,O=fwmgt1..99jybd"

“opsec_sic_name” is the distinguished name(DN) of the client application

opsec_sslca_file "opsec.p12"
*opsec_sslca*_file is the name of the certificate file on the client system

fwmgmt1 is the name of the Firewall-1 management module

fwmgt1 host 10.30.6.4

This indicates to the probe the address of the Firewall-1 Management module

fwmgt1 auth_port 18184

auth_port is the port used by the LEA server running on the management module. The authentication is established, using 18184/tcp between OPSEC server (Checkpoint Management module) and client (the host running Firewall-1 probe

fwmgt1 auth_type sslca

auth_type is the authorization method used by the firewall, the Secure Sockets Layer Certificate Authentication (SSLCA)

```
fwmgt1 opsec_entity_sic_name "CN=cp_mgmt,O=fwmgt1..99jybd"
```

opsec_entity_sic_name is the firewall management module distinguished name (DN)

It is required to complete the following tasks to ensure proper authentication of the probe, using Checkpoint Policy Editor:

- Add a “Client workstation”, for the host running Firewall-1 probe
- Create an OPSEC application, called “NETCOOL”,
  - with the “Client workstation” created above as “host”

    - "LEA" and "SAM" against "Client Entities"
- *Create a certificate for the OPSEC application*
    - Enter a unique password in the "Communications" window for the OPSEC application, "NETCOOL"
    - Choose "Initialize" to generate a certificate

- Transfer the certificate to the OPSEC client
- Log into client, the Firewall-1 probe host
- Run ./opsec_pull_cert –h <address of FW-1 management> –n <OPSEC Application> -p <password to the certificate> –o <cert_name> –od <sicname>

Management file (firewall1_management.fw)

It defines the management modules the probe connects to.

```
bash-2.03# more /opt/OMNIbus/probes/solaris2/firewall1_management.fw
# sample management file
MAN {
name: 'fwmgt1'
type: 'normal'
position: 'end'
mode: 'current'
}
MAN {
name: 'fwmgt1'
type: 'account'
position: 'end'
mode: 'current'
}
```

"type" is the type of the log to capture events from; "NORMAL" means the events are acquired from the current firewall log and "ACCOUNT" means from the current accounting log
"position" indicates where to start reading the events from
"current" mode means probe acquires events as they are generated

SAM configuration file (firewall1_sam.conf)

```
bash-2.03# more firewall1_sam.conf
#opsec_sic_name "CN=opsec_application_name,O=org_code"
opsec_sic_name "CN=NETCOOL,O=fwmgt1..99jybd"
opsec_sslca_file "opsec.p12"
management_module host 10.30.6.4
management_module auth_port 18183
management_module auth_type sslca
management_module opsec_entity_sic_name "CN=cp_mgmt,O=fwmgt1..99jybd"
```

To capture events from the FireWall-1 Enforcement Modules that are managed by FireWall-1 Management Stations you are connecting to their details must be added the probe's policy file: $OMNIHOME/firewall/firewall1/firewall1_policy.fw.

Here are the environment variables that are required:

```
OMNIHOME=/opt/OMNIbus
OPSECDIR=/opt/OMNIbus/firewall/etc
```

Finally we are required to configure the probe to accept events from FireWall-1 Enforcement Modules.

To capture events from the FireWall-1 Enforcement Modules that are managed by FireWall-1 Management Stations you are connecting to their details must be added the probe's policy file: $OMNIHOME/firewall/firewall1/firewall1_policy.fw.

The probe can be started by running $OMNIHOME/firewall/firewall1/nco_p_firewall. It should ideally be added under the control of what Micromuse calls as Process Control (NCO_PA) so that NCO_PA watches and automatically restarts when the process dies, but we could not really do that as the Firewall-1 probe 3.5 expects $OMNIHOME variable to be set to /opt/OMNIbus whereas the rest of the version 3.6 sub-systems expect this variable to be set to /opt/netcool/omnibus.

The probes come with abnormal behavior definition file. This file contains a set of usage templates to detect abnormal behavior. Here are some of the interesting events detected by the Firewall-1 default behavior definition file:

- Machines Scan
  - Multiple machine scan
  - a single IP hitting 50+ destination machines
- Denial of Service/Application mis-configuration

  - HTTP/POP3/SMTP/TCP/UDP Port Overuse
  - 100+ hits from one machine to another Machine on port 80 (HTTP) or 110/tcp (POP3) or 25/tcp (SMTP) /any single TCP/UDP port from one machine to another
- Machine overuse
  - 1000_accepts between any 2 machines
- Password Guessing
  - Telnet password guessing
  - FTP password guessing
- Port Scanning
  - TCP port san
  - UDP port scan
- Application Overuse
  - Large FTP/HTTP/POP3/SMTP/TCP/UDP transfer check
- Excessive ICMP echo request/replies

We found we were generating numerous false positives for legitimate traffic in terms of excessive port usage and excessive UDP usage. Again it took enormous efforts to fine-tune the rules to suppress these messages to get the firewall events down to a few hundreds a day. I am still not fully convinced that these are actionable events that can be sent to central console but we nevertheless sent them over as mandated. These are certainly useful to security administrators or network administrators, and should make their life a lot easier.

Dealing with Access Control Server messages

You can forward all syslog messages from CiscoSecure ACS server by adding the following line to /etc/syslog.conf in CiscoSecure Solaris server:

```
local0.debug     @loghost
```

Needless to say we need to make syslogd reread its configuration file, with SIGHUP.

If you are not too comfortable adding few thousands of messages to your central syslog server on any given day, it may be worthwhile logging it locally and running SWATCH or a vendor syslog probe to parse it.

It is possible to make it log locally with the following entry in the syslog.conf

```
local0.debug     /var/log/csacslog
```

Venturing into the world of Network Sensors

Snort engine based SourceFire network sensors were deployed to implement network intrusion detection on the private network. This provides real-time traffic analysis and packet logging on IP networks. Sourcefire Intrusion Sensors are delivered as complete appliances, with hardware, software and operating system optimized for and they provide an easy to use web-based interface for all aspects of sensor management. BTW, SourceFire is founded none other than Snort's author, Martin Roesch.

http://www.sourcefire.com/technology/whitepapers.html#detection

Snort can be configured to send alerts when a network packet matches the rule in the configuration file. The management interface of network sensors are on management VLAN while the sniffing interface of the network sensors are on the access VLAN to capture the production network traffic between GSEC Inc. and its clients over the private network.

Most organization prefer to let network sensors send the alerts to the central management console to form a security operations console. We were required to send snort alerts to a central console, which is normally monitored by personnel at the NOC round the clock. This also opens up the possibility for notifying security personnel of critical alerts through paging and other notification systems, which are in place.

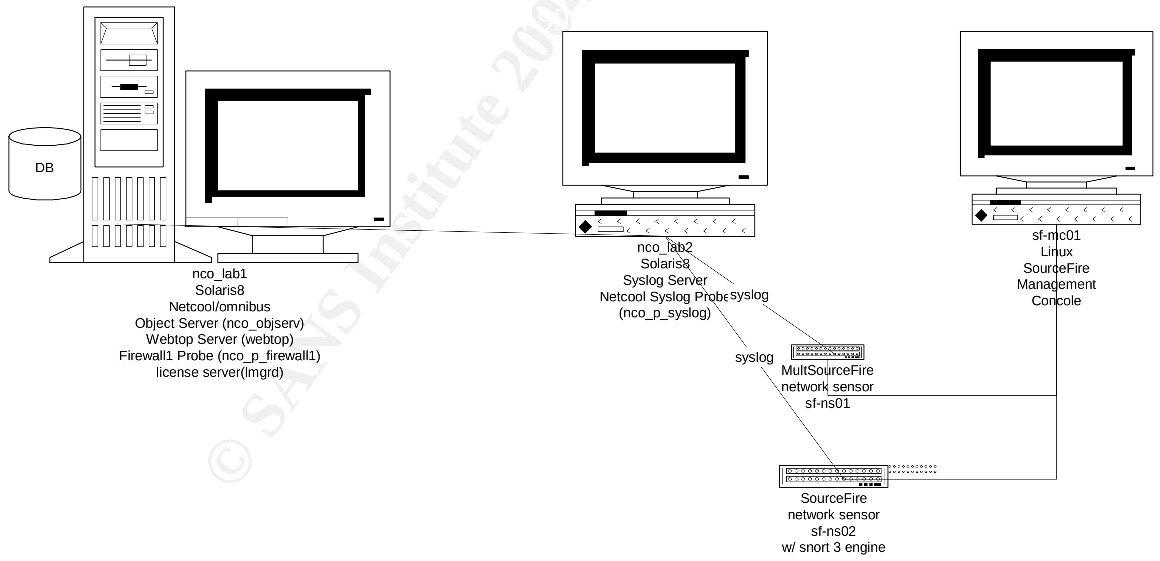
We have deployed highly customized snort rules in our network sensors to ensure there are not too many alerts on regular basis. Our policy editor using latest version of SourceFire provides us additional features such as suppression of alerts based on source IP or destination IP as well as throttling of alerts, besides standard snort "pass" rules. This has helped bring down the alert volume to a manageable label and hence there is absolutely no threat of flooding a central management console.

As we were required to forward the alerts to Omnibus management console, SourceFire network sensors were configured to generate syslog messages to the central syslog server where Netcool/omnibus syslog probe is deployed. Syslog probe intercepts the messages and sends it to Netcool object server to be displayed on webtop clients.

Netcool/omnibus Syslog probe (nco_p_syslog) is installed and started on the central syslog server.

SourceFire network sensor appliance (Snort intrusion detection engine) has been set up to generate and forward syslog alerts the central syslog servers in GSEC primary and contingency sites.

```
root@sf-ns01:/etc/sf# more bysyslog.conf
processor dp_alert
output alert_syslog2: facility: LOCAL1;
severity: ERROR;
tag: SFIMS;
syslog_host: 192.168.1.2;
syslog_port: 514;
```



We had to implement the following steps to configure Netcool/OMNIBUS to handle SourceFire syslog messages and Cisco ACL log messages.

The following line added to /etc/syslog.conf in the host running syslog probe

```
*.debug                                /var/adm/ncolog
```

Syslog probe properties file is configured with /var/adm/ncolog as syslog file.

Add the following to Netcool/OMNIBUS syslog rules file /opt/netcool/omnibus/probes/solaris2/syslog.rules

```
else if(regmatch($Token6, "^SFIMS:")) {
$agent = "SourceFire"
}

case "SourceFire":
@Agent = $agent
@Summary = extract($Details, "SFIMS: \[[0-9]+:[0-9]+:[0-9]+\] (.*)")
@AlertKey = extract($Details, "SFIMS: \[[0-9]+:[0-9]+:[0-9]+\] (.*)")
@Severity = 4

if(regmatch($Details, "SEC-6-IPACCESSLOG.*"))
{
#   discard
   @Type=2
   @Severity=2
}
```

Add the following lines under section case "Cisco":

```
#               @Summary = extract($Details, "%(.*)")
                 else if (regmatch(@Summary, ".*SEC-6-IPACCESSLOGP.*"))
                 {
                        @Severity = 2
                        @Type = 2
                        @AlertKey = extract($Details, ".*list ([^ ]+).*")
                 }

"syslog.rules" 1035 lines, 23866 characters
```

Check the modified to ensure there are no syntax errors

```
/opt/netcool/omnibus/probes/nco_p_syntax -rulesfile /opt/netcool/omnibus/probes/solaris2/syslog.rules
```

Load the new rules by sending KILLHUP to Netcool/OMNIBUS's syslog probe process (nco_p_syslog)

```
/opt/netcool/omnibus/probes>kill -HUP 2812
```

Here are some of syslog messages we received from SourceFire (snort) network sensor with custom policy

May 17 18:16:34 sfns01 sfns01SFIMS: [1:499:3] Snort Alert [1:499:0] [Classification: Potentially Bad Traffic] [Priority: 2] {ICMP} m.n.o.28 -> a.b.c.129

DESCRIPTION: http://www.snort.org/snort-db/sid.html?sid=499
RULE: alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ICMP Large ICMP Packet"; dsize: >800; reference:arachnids,246; classtype:bad-unknown; sid:499; rev:3;)

Suspect TFTP Get
May 17 18:18:16 sfns01 sfns01SFIMS: [1:1444:2] Snort Alert [1:1444:0] [Classification: Potentially Bad Traffic] [Priority: 2] {UDP} a.b.c.130:1634 -> a.n.m.66:69
DESCRIPTION: http://www.snort.org/snort-db/sid.html?sid=1444

RULE: alert udp $EXTERNAL_NET any -> $HOME_NET 69 (msg:"TFTP Get"; content:"|00 01|"; offset:0; depth:2; classtype:bad-unknown; sid:1444; rev:2;)

Our inference: This is turned out to be due to bad router configuration due to which router keeps attempting TFTP Get from a non-existent TFTP server

May 17 18:18:27 sfns01 sfns01SFIMS: [119:2:1] Snort Alert [119:2:0] [Classification: Unknown] [Priority: 1] {TCP} a.m.n.35:55639 -> a.b.c.1:80
DESCRIPTION: http://www.snort.org/snort-db/sid.html?sid=119

RULE: alert tcp $HOME_NET 6789 -> $EXTERNAL_NET any (msg:"BACKDOOR Doly 2.0 access"; flow:established,from_server; content:"Wtzup Use"; depth:32; reference:arachnids,312; sid:119; classtype:misc-activity; rev:4;)

May 17 20:08:08 sfns01 sfns01SFIMS: [121:4:1] Snort Alert [121:4:0] [Classification: Not Suspicious Traffic] [Priority: 2] {PROTO255} a.b.c.4 -> a.e.c.74
DESCRIPTION: http://www.snort.org/snort-db/sid.html?sid=121
RULE: alert tcp $EXTERNAL_NET 1000:1300 -> $HOME_NET 146 (msg:"BACKDOOR Infector 1.6 Client to Server Connection Request"; flow:to_server,established; content:"FC "; sid:121; classtype:misc-activity; rev:4;)

May 17 20:08:08 sfns01 sfns01SFIMS: [121:4:1] Snort Alert [121:4:0] [Classification: Not Suspicious Traffic] [Priority: 2] {PROTO255} a.c.a.6 -> a.c.d.42
Our inference: These are certainly "False Positives" from the sensor

Conclusion

Now we have the real-time security console fully up and running in the SOC, which fulfilled a key management objective. The console provides a central view of all network security events.

Our journey towards weeding out IDS false positives, false Firewall-1 abnormal events and eliminating broadcast noise off access syslog messages continues. Besides keeping us busy, Snort has helped us figure out a lot of network issues due to misconfigured routers. The Firewall-1 probe has certainly helped figure out and fix a lot of issues due to bad rules or remove logging of unnecessary drops. Thanks to de-duplication, we are seeing a few events on the console unlike the firewall log viewer which does not de-duplicate events, this is a big plus. We have just one entry for a connection request that has been dropped thousands of times a day with the count going up each time a duplicate event occurs.

The operators would require a good amount of training on security related aspects to understand and react to the security messages that show up on the console. It would take a lot of hand holding to get the SOC personnel up to speed before we make the SOC productive. It's also important to continue the efforts to eliminate non-actionable events reaching the console.

This exercise has certainly rendered a useful security console, which could certainly increase the productivity of security administrators and prevent possible attacks. The correlation of events from different devices would be the next big challenge facing us. The ongoing efforts to suppress the "false positives" and update the rules to incorporate new signatures continues

After all this, I realize that the "forest" is really a moving target (reminds one of Shakespeare' s Macbeth) and we can never afford to be complacent that we have already conquered it!!! These tools certainly do provide us with a good insight into what goes on in the network, but they don't make us feel safe. We find some nay sayers in the organization dismiss these efforts as "much ado about nothing". Well, the forest could turn out to be just a bunch of insignificant trees on many occasions or it could be Macduff's invaders, who had cut branches of trees to hold in front of them as camouflage and we better be prepared to face them before they get us. So our efforts go on.

Appendix –A

```
#!/usr/local/bin/perl
## acl_insight
## provides daily summary of router IP access-list violations
##

use Getopt::Std;
use File::Basename;
use Sys::Hostname;
use POSIX 'strftime';
# Initialize
$PGM   = basename($0);
$HOST  = hostname();
$time  = POSIX::strftime("%D %T",localtime());
$date  = POSIX::strftime("%D",localtime());
$fdate = $date;
$fdate =~ s/\///g;
print "$time $PGM starting on $HOST\n";
$BASE_DIR    = '/var1/log/';
#$BASE_DIR    = './';
$ACCESS_LOG  = $BASE_DIR.'syslog_info_day';
$SERVICES    = './port_table';
open (LOG,"<$ACCESS_LOG") || die "ERROR:Could not open $ACCESS_LOG:$!";

if (open(SERVICES,"$SERVICES")) {
   while (<SERVICES>) {
        chomp;
        next if /^#/;
        /^(\S+)\t+(\S+)/ || next;
        $services{$1} = $2;
    }
   close(SERVICES);
} else {
    die ("FAILED to open $SERVICES: $!");
}

foreach $svcport (sort keys %services) {
   #print "$svcport  $services{$svcport}\n";
   $denial_by_service{$svcport} = 0;
}

print "Router Access-list Violations  Summary  - Dated $date\n";
print "generated by \"$PGM\" on $HOST at $time\n";
```

```
#Aug  8 14:32:58 rtr66sa 930423: .Aug  8 14:26:08.486 EDT: %SEC-6-IPACCESSLOGP: list 101
denied udp a.u.2.57(1025) -> a.u.255.255(42508), 2 packets


$msgcount = 0;
$deny     = 0;
$count    = 0;
$discard  = 0;

while (<LOG>) {
   chomp;
   $msgcount++;

   if ($_ =~ /\S+\s+\d+ \d+:\d+:\d+ (\S+) \d+: .* E[S|D]T: %SEC-6-IPACCESSLOGP: list (\d+)
denied (\S+) (\S+)\(\d+\) -> (\d+\.\d+\.
\d+\.\d+)\((\d+)\),/) {

     $deny++;
     ($host,$acl,$proto,$sourceip,$destip,$destport) = ($1,$2,$3,$4,$5,$6);
     #if destination is broadcast address, discard the entry
     $svcport = $destport."-".$proto;
     $srcoctet3  = $sourceip;
     $srcoctet3 =~ s/\.\d+$//;
     $destoctet3 = $destip;
     $destoctet3 =~ s/\.\d+$//;

     if ($destip =~ /\.255$/) {
       #print "broadcast - discard \n";
       #last;
       $discard++;

     } elsif ($srcoctet3 eq $destoctet3) {
       #assuming 24 bit network mask, discard access log
       #if the traffic is between hosts on the client network
       #(interface ethernet 0 on the edge router)
       #log discard
       $discard++;
     } else {

     #print "$acl denied $proto $sourceip -> $destip($destport) $services{$svcport}\n";
     $denial_by_reason{$destport, $proto, $sourceip, $destip, $acl, $host }++;
     #$denial_by_source{$sourceip}++;
     $denial_by_destination{$destip}++;
     $denial_by_service{$svcport}++;
     #$denied_packets_by_service{$destport/$proto} += $packets;
     #$denied_packets_by_source{$sourceip} += $packets;
```

```
      #$denied_packets_by_destination{$destip} += $packets;
      #$denied_packets_by_service{$destport} += $packets;
      }
    } #if
} #while

close (LOG) || die "ERROR:Could not close $ACCESS_LOG:$!";

# Desired output
print "Service denials\n";

for $key (sort keys %denial_by_service) {
        printf("|  %15s %-15s -> %-15s \n", $services{$key}, $key, $denial_by_service{$key}) if
$denial_by_service{$key} != 0;
}

if (keys %denial_by_reason) {
    print <<EOF;
*** Rejected connections
================================================================================
==========================
 Port/proto   Service        Source          Destination      access_list  Router    Attempts
================================================================================
==========================
EOF

    for $key (sort keys %denial_by_reason) {
        $attempts = $denial_by_reason{$key};
        $count   += $denial_by_reason{$key};
        ($destport, $proto, $sourceip, $destip, $acl, $host ) = split( $;, $key );
        $svcport = $destport."-".$proto;
        $services{$svcport} = 'unknown' if !$services{$svcport};
        printf("| %-10s %-15s | %-15s->%-15s |list %-3d | %-10s| %6d |\n", $svcport,
$services{$svcport}, $sourceip, $destip, $acl,
 $host, $attempts ) if $destport < 1024;
    }

    for $key (sort keys %denial_by_reason) {
        #$attempts = $denial_by_reason{$key};
        #$count   += $denial_by_reason{$key};
        ($destport, $proto, $sourceip, $destip, $acl, $host ) = split( $;, $key );
        $svcport = $destport."-".$proto;
        $services{$svcport} = 'unknown' if !$services{$svcport};
        printf("| %-10s %-15s | %-15s->%-15s |list %-3d | %-10s| %6d |\n", $svcport,
$services{$svcport}, $sourceip, $destip, $acl,
 $host, $attempts ) if $destport > 1024;
```

```
    }

}

print "\n\n";
print "Total count of syslog messages processed        : $msgcount\n";
print "Total count of denied access attempts              : $deny\n";
print "Total count of broadcast attempts dropped        : $discard\n";
print "Total count of genuine access attempts dropped   : $count\n";
exit(0);
```

References

Wealth of information available on http://www.sans.org/rr

Sun Solaris documentation
http://docs.sun.com/db/coll/47.8

CERT provides some very useful advice on how to harden Solaris.
http://www.cert.org/security-improvement/practices/p038.html

Snort 2.0 Intrusion Detection by Brian Caswell, Jay Beale, James C Foster and Jeffery Posluns

SourceFire White papers and documentation
http://www.sourcefire.com/technology/whitepapers.html

Real-time Alerting with Snort
By Jack koziol
http://www.linuxsecurity.com/feature_stories/feature_story-144.html

http://www.cert.org/security-improvement/implementations/i003.01.html

http://www.snort.org/

Cisco Systems Documentation
http://www.cisco.com/univercd/cc/td/doc/product/software/ios11/sbook/ssysmgmt.htm

http://www.micromuse.com/sols/ent_sec_man.html