Global Information Assurance Certification Paper

Title: Implementation of a Stealth Log Server

David Munson
11/17/03 3:09 PM
GSEC Practical Version 1.4b, Option 2

Contents

Abstract/Summary

Stealth logging is a topic of interest for systems administrators who have the responsibility for ensuring the security, and integrity of the systems in their charge. A key aspect of the job is maintaining the integrity of logging information. Stealth logging is not a panacea. It can offer an extra layer of support for attempting to insure log data integrity by further restricting attack vectors on the data itself in the overall layered approach to system and network security. It seems most appropriate for critical servers where continuity of logging information may be critical for several reasons. One critical reason may be for preserving an audit trail for dealing with legal issues related to the compromise of a key system.

The purpose of this practical is to document the implementation of a stealth log server for the critical linux servers on the network in a university department. The goal is to increase the reliability and integrity of log information generated by these critical servers. In an open environment such as a university, this has added importance.

To accomplish this, I installed and configured a RedHat linux machine with no IP address on its monitoring interface, and installed and configured Snort to log syslog traffic arriving at the monitoring interface. I configured the servers to log to a bogus IP address, and also configured the switch port on which the stealth logger interface listens.

Before Snapshot

The current server network consists of a traditional log host, a machine allowing interactive logins, a print server, two DNS servers (a primary, and a fallback), two mail machines (a primary, and a fallback), a disk server, an rsync mirror of the disk server, a web mail server, a web server, and a small Mosix cluster.

The machines noted above all currently log to the conventional log host as do a number of other linux desktop and research machines located throughout the network. The log host runs a simple script to cull information from the logs. The scripts search for details regarding specific protocol activity such as ftp, telnet, the “r” commands (rlogin, rhost, etc..), and for protocol information on packets hitting the firewall.

The environment in the university department is fairly open as the user base we serve is diverse, and has diverse and unique needs for computing based on individual research programs. The university purpose and philosophy is oriented toward providing the free flow of information in a secure manner that meets the needs of the researchers. While we strive to offer this in the safest manner possible, we are mindful that problems can arise from internal machines used by unknown undergraduate, graduate students, faculty and staff members, and from external sources as close as the students living in the dorms on campus.

The current log host has served a valuable function as the central repository of logging information. It offers both the chaos and convenience of having several machines and facilities log to one central location with restricted access. It facilitates cross checking of events between machines in addition to ensuring the integrity of the logged data by providing a backup for the local log files on each system, as the logging information is generated. This results in the high availability of logging data even in situations where hard drives fail between backups. With sufficient storage, it also allows retention of logs beyond what might be suitable on a server with a significant amount of syslog activity. Log rotation can be allowed to occur in the default manner even for machines where it might be desirable to maintain logs in excess of those maintained on a production machine. The central log host zips and time stamps the logs before moving them to a storage directory. In general there are in excess of one year's worth of logs in this directory accessible at any given time. These are written to CD periodically for archival purposes. Since the logs are text, they compress quite nicely with gzip. Prior to zipping several reports are produced, and mailed out daily to a list of appropriate administrators to look through for anomalous entries.

Centralized logging ensures, to some degree, that the log data is out of the immediate reach of an attacker who might gain privileged access to a compromised machine. Savvy attackers often try to clean up traces of their activities by scrubbing the logs clean of any traces that they may have left on the machine. This is a critical aspect of the motivation for centralized logging. With the typical centralized log host, as with any machine with a network presence, there is always the possibility of system compromise. These compromises could be accomplished through vulnerabilities in the software on the log host, failures to appropriately configure the software on either the log host, or on the machine sending its logs to the log host; and denial of service attacks against the syslog service used on the log host, or denial of service against the syslog, and/or other services on the machine attempting to log to the log host. Any of these could cause the loss of logging data. This loss of data from a critical system such as a mail server, or web server could be highly detrimental to the security and integrity of a particular system, and potentially detrimental to other machines on the network.

This directs attention to the advantages of protecting the logging data even further if possible; especially log data for critical, and more highly visible, higher traffic systems. Such systems may, by their function and traffic load, be more susceptible to compromise and attack. They usually have ports open that have histories of exploitation, such as ftp servers.

The vulnerability of the log host has lead some[1] to propose the construction of log hosts that do not use an IP address. This minimizes some of the human-caused potential for defeating centralized logging. A log host without an IP address associated with its monitoring interface in no way completely assures the integrity of logged data, but places another stumbling block in the path of attackers. While an attacker on a compromised machine might become aware of the technique of stealth logging, the

[1] Bauer, URL : http://www.linuxjournal.com/article.php?sid=6222

logger itself is less likely to be compromised, but could still be attacked via denial of service, or via the manipulation of the configuration of the compromised host to stop logging to the stealth logger.

During Snapshot

This section contains detailed, step-by-step installation and configuration steps, and a brief description of the log host hardware. This also contains network configuration changes made to take advantage of the stealth log server. Since my environment is switched, it required the use of a monitoring interface on the switch containing the devices that will use the stealth log server.

The hardware used to construct the stealth log host consists of a single processor AMD Athlon 1400 cpu with 256 MB of memory, two 40GB IDE hard drives, a dual interface gigabit network card, a video card, a CD drive, and floppy drive all in a rack case with a 300 W power supply. I installed RedHat linux 9.0 and ran up2date to be sure that the machine was patched. I also installed Snort 2.0.2 by building it from sources on the stealth machine. One of the disks contained the operating system partitioned appropriately, and the other contained a single partition mounted as /log to contain the log files. I installed tripwire and configured it to fit my installation, and created the signatures database. I reran tripwire in audit mode to verify the database.

The logging interface on the network card is attached to a port on a HP ProCurve 4000M switch. The switch port was configured according to the manual instructions[2] for the switch to monitor all of the ports on that switch (Appendix A). This switch is a key node in the network infrastructure. It is a point where all traffic from the main switches in the building converges, and is the network connection to the firewall.

After these basic steps, I configured the network card on the machine so that one interface was configured normally with an appropriate IP address and settings. This allowed me to have an interface with an IP that I could take up and down via *ifconfig* to allow the movement of files, and the transfer of mail as needed. The other interface I configured without an IP address[3] (Appendix B). This serves as the monitoring interface on which Snort listens for the UDP 514 syslog packets coming from the hosts for which I wanted to capture logging information.

I then constructed a Snort rules file to serve my purpose[4] (Appendix C) with the logging directory configured to be my /log partition, and to dump character data only. I then tested the switch configuration by running snort to check that the switch was in fact sending all packets to the switch port to which the monitoring interface was connected :

---

[2] HP ProCurve Switches 1600M, 2424M, 4000M, and 8000M Management and Configuration Guide, 6-34 – 6-38

[3] Bauer, URL : http://www.linuxjournal.com/article.php?sid=6222

[4] Bauer, URL : http://www.linuxjournal.com/modules.php?op=modload&name=NS-lj-issues/issue102&file=6222l3

snort –dve [5]

This dumped all the packets to the screen for me to view so that I could verify that I was in fact seeing packets of all sorts from multiple sources.

The key issue with getting the machines that are the desired sources of the logs to actually send logs to the log host is setting a bogus IP address associated with the MAC address of the monitoring interface on the stealth log host into to the ARP table on the machine from which the information to log comes. The *arp –s* command makes this trivial :

/sbin/arp –s 10.0.1.66 aa:bb:cc:dd:ee:ff:gg

I created a few trivial scripts to add, delete, and check the ARP tables (Appendix D). I placed the *arptest* script in /etc/cron.hourly. There is always a chance that an attacker could spot the arp table addition and delete it from the table. The cron entry does not do much to ameliorate this, but it does not hurt to have it in place just in case he does not check the cron directories. The crontab might be an even better choice since it is slightly less obvious, but still vulnerable to detection.

I started snort in daemon mode using the configuration file I had created, and let it wait for packets from my hosts even though I had not yet configured the syslog.conf file :

snort –D –c /usr/local/etc/snort.conf

The syslog.conf needs to be modified to add a line to log the bogus IP. In this case, everything is logged there, and picking out the desired packets is left to the scripts on the log host. The amount of data logged is reasonable in this particular situation, but may not be in a context where many more high traffic servers log to the machine :

. @10.0.1.66

All the port 514 traffic will go into a directory designated by the source machine's IP address on the stealth log host in the log directory, /log in this case. I then sent a HUP signal to the syslog process to read in my change to the log file, and start sending the packets to the stealth log host.

Once the hosts to be logged were configured, and syslog restarted, directories for each IP address were created in the /log directory by snort. In each directory, a file named UDP:514-514 contains the logged packets for the machine associated with that IP. For convenience, symbolic links from the directories with IP address names to host name links to those directories were created (eg. *ln –s 10.0.0.5 dnsa*). Directories were

[5] Green, Chris & Roesch, Marty, P, URL :
http://www.snort.org/docs/writing_rules/chap1.html - tth_sEc1.1

created for each machine where the log files could be moved on a daily basis (eg. mkdir dnsa_save).

At this point everything was functional in terms of the logging. The machines from which logging data is to be captured actually logged to the log host, and the log host saved the packets destined to port 514 of the stealth log host to the /log directory. For a glance at what the directory looks like see Appendix E.

The next step was to create a script to automate the operation of the logging and the movement of the log files to their respective directories. The files can then be manipulated to produce reports, and stored appropriately. The top level script moves the files to the appropriate save directory, stops snort, moves each file, restarts snort, opens the IP configured interface, runs the report scripts, attempts to set the clock to an accurate time, and shuts down the IP configured interface (Appendix F). A report script can be customized for each host for which logging is done, and the reports mailed to the appropriate person or list. The scripts in the Appendixes do not use loops to make obvious the simplicity of their function, but should be streamlined.

The report scripts in each directory use the *strings*, and *grep* commands to pull information from the file, and save it to a report file for mailing to an appropriate person, or list. The files are then gzipped to a file named with the name of the host, and a date stamp (Appendix G contains a sample script).

The final step was to add the command to start snort logging in the /etc/rc.d/rc.local file to be sure that the logging begins as soon as the stealth log machine boots. The IP configured interface is activated on boot, and then in rc.local the ifconfig command to shut it down is run.

After Snapshot

This section contains testing information, and reports generated.

The testing phase was partially incorporated in the previous section when the logger was actually built, the snort software was installed and configured on the logger, and the machines that were to log data to the stealth log host were configured to make this logging possible. While fairly straightforward on the whole, some points are worth revisiting. Beyond this the actual log files and reports speak for the functionality of the logger, and the configured hosts as a system.

The use of the second IP address configured interface allowed reports to be reviewed in a timely fashion. “Out of sight is out of mind,” as they say. The arrival of the logs in mail helps keep the stealth log host from being forgotten. It also focuses attention to activity on the critical hosts logging there. At the same time, the configured interface could be shut down when not in use. The intention was to bring it up only once a day just long enough to allow the reports to be mailed out, and then to shut it down. This presents a window of risk, but a fairly narrow one compensated for by the benefits. I configured

Iiptables to reject everything incoming except responses from DNS host name lookups as an added precaution.

The initial tests on the log host followed the installation of Snort, and the configuration of both the switch port to which the monitoring interface was connected, and the configuration of the interface itself to function appropriately in promiscuous mode without an IP address for the capture of packets off of the network. The purpose of the test was to verify that all of these had been configured correctly. Once each piece was configured, I brought up the monitoring interface (ifconfig eth1 up). I then started snort with the packets dumped to stdout (snort –dve). I observed plenty of packets coming in from both of my subnets, and packets coming in from the router that connects the network to the rest of the campus. If the switch port had not been correctly configured, the traffic I would have seen would have been much more restricted, and limited.

The next phase was to construct an appropriate snort configuration file. I wanted to be able to be able to log from both of the subnets on my network. This is seen in the snort.conf file in the definition of HOME_NET[6]. This seemed to give me exactly what I wanted. The first host configured had its IP address directory created on the stealth log host within seconds of the SIGHUP I sent to its syslog daemon to read the modified syslog configuration file. In the directory was the UDP:514-514 log file, and there was the logging data along with the other information snort had logged. A sample raw record minus control characters can be seen in Appendix H. I saw the directories of the servers whose syslog.conf files and arp tables had been configured, and had the SIGHUP sent to their syslog daemons appear one by one.

I modified the snort configuration file several times mainly out of curiosity as to what I might be able to do with the file. I found that simplicity made the whole process easier to deal with and returned to the original configuration file listed in Appendix C.

The script I constructed to automate the log cleanup is in Appendix F. At first, it was placed in /etc/cron.daily, but it failed to restart snort appropriately, and logging halted completely. Rather than attempt to determine the reason for the failure, I scheduled it in root's crontab with this entry :

```
0 5 * * * /usr/local/etc/snort_log_clean
```

This solved the problem of snort not restarting.

I inserted individual reports into each save directory each with the same general format that is in the example script in Appendix G. I could then customize each report according to the function of each system, and according to what I wanted to track for a particular system. The example in the appendix is the simplest example since the machine is the login node of the Mosix cluster. Logins have to be allowed from users. The main focus is to look for errors, and failure messages. As with all of the report

[6] Roesch, section 3.3

scripts of the other servers, the report facility needs significant tuning and refinement. In many of the other reports, the scripts also pull out ssh related log entries as in the sample report in Appendix I. Obviously, most of these servers should have restricted logins if any are allowed. The usernames of legitimate users are known. Appropriate login activity is also well understood. With only one exception, ssh is the only form of interactive connection allowed. The example report in Appendix I shows some understood ssh activity between the dns machines that keeps the files containing the IP tables for my network synchronized between the two DNS machines on an hourly basis. In our environment, this is an appropriate granularity for this activity, and the log entries are expected.

A default nmap TCP scan of the stealth log host reveals that in fact it looks as though the machine behind the bogus IP is down even though it is actively accepting the logging data (Appendix J). The second nmap scan with –P0 yields the same conclusion, but might make the savvy attacker curious because of the "Strange error from connect (64)." This might lead to some careful, or brute force probing to determine if there is really something there. A UDP scan of the interface yields the third result in Appendix J. The program scans, finds nothing, backs off alternately 15 and then 60 seconds with the same result, "Host is down."

An nmap scan of the IP configured interface when it is down simply returns the same results as for the stealth interface. The results of nmap scans on this interface are in Appendix K. The TCP scan when the IP configured interface is up shows that all ports are closed. I have turned off all unneeded services and rejected all TCP traffic via iptables. The UDP scan looks very similar to the TCP scans since the only UDP traffic that is allowed in when the interface is up is DNS lookups from two specific IP addresses. Unless the attacker knew which hosts those were it would be difficult to find them without actually capturing the traffic when the interface is up. This is a distinct possibility for someone doing a long-term scan of the network. It is a hole that opens for only a short time, but currently at predictable intervals that could be mapped. This is an aspect that could use some randomization. Randomization suggests that more detailed time stamping of the log files will be required. It is a reasonable to strive to keep the open window moving.

On the whole it appears the overall goals of this project have been accomplished. There is room for refinement of certain aspects of the reporting, and the functioning of the system as a whole. The randomization of the open window within which the IP configured interface is up is a goal to strive for. The overall impact of the project further secures many aspects computing on the network.

Impact

This section contains conclusions; lessons learned; the impact on system administration, the security of the servers, and the security of the network as a whole.

The presence of the log host for the critical servers has improved the security of the network as a whole by ensuring that the activity on the critical servers is logged more securely. The possibility of logging failure is diminished. By expanding the logging to three locations from two there is less chance of a total compromise of logging data. This improves the integrity of every machine that uses and connects to these hosts. This includes every other machine on the network. Every machine on the network is using the mail hosts and the DNS servers even if they are using none of the other servers on an active basis. The DNS servers are also serving DHCP clients. The conventional log host still provides a central location for examining cross machine events, and maintains the logs of desktop linux machines throughout the network. My intention was not to replace the conventional logger with the stealth log host at this time. Moving forward with expanding its use on a case by case basis is worth consideration. There are less critical high activity machines that could benefit from the extra security. In particular, since Mac OS X is a flavor of BSD Unix, we have some Macs with departmental importance that would be better secured by logging to the stealth log host.

While a number of the servers in question do not have significant numbers of interactive logins, and even then only by certain staff, this may be a prime reason for being able to isolate their logging data since even a watchful eye tends to overlook things. The example report in Appendix I is evidence of this. The ssh activity between the dns hosts is routine. It would be easy to overlook other kinds of ssh activity. The stealth log host is a convenient place to search through the logs for each individual server for relevant data. The conventional central log host is logging, and searching for information in a more broad scoped manner. This makes events impacting multiple machines possible to trace. The restricted amount of data separated by machine into separate directories on the stealth log host makes searches for information quicker, and more concentrated on particular hosts. Specifically, it allows careful observation of the most critical hosts for anomalous events.

The impact on system administration takes many forms. It would be unwise to ignore the fact that this is one more machine to manage. It is a machine that by nature of its configuration, and purpose is not manageable remotely. For obvious reasons, console-only administration is the lone management point for such a server. To make it manageable remotely, might be possible with some significant effort. It could not be done without complicating the current setup, and introducing an increased risk of compromise by offering a greater number of modes of attack. In any case, remote administration is not a requirement that needs consideration in this particular case. It requires hands on management, but attempts have been made to make it have as "stealthy" a profile from the network viewpoint as was possible. The possible exception to this is the regularity with which the configured interface is actually up and functional.

The positive impact is, in part, the additional peace of mind that an extra layer of security offers. Adding an extra layer of precaution is a good thing. While our mission does not encompass some of the high functional impact that operations in a financial institution or a health care facility do, the protection of the security and integrity of our key servers is critical to the smooth operation of the department. Careful logging is a

key to this. The stealth log host adds to the assurance that accurate logs of the key server functions are kept in a secure manner. The particular manner in which snort creates and writes to the log files makes separation of machine specific logs, and reporting on those logs trivial. This is a bonus. This along with the continued use of the conventional log host offers the best of situations. The conventional logger offers the convenience of being able to get a broad view across the spectrum of machines logging on this machine. The stealth logger offers separation, and a higher level of security where it is most desired. The logs of the key servers can be individually, and critically examined in isolation. The two machines could serve as a means to cross check log integrity. While I have not implemented this via a script as yet, I think it will be valuable, and is worth the effort as a tool that could be run on some regular basis to verify the integrity of stored log files.

The main purpose of the stealth log host is to shut off as much unneeded and unwanted access to the critical log data as possible. This is of primary importance in the situation where a key server, or a machine connected to a key server is compromised. The compromise could be a direct attack on a server, an attack on a machine using the servers in a way that might cause the compromise of a main server, or an attack on the network as a whole. In any of these cases, it might be an attack, or machine failure of some other sort that might need to be documented via relatively pristine log files. I believe this goal has been successfully accomplished. My one caveat is that there seems to be a timing issue with regard to mailing the reports out. It seems that the *sendmail –q* may require that the IP configured interface be left open longer than the script allows as mail delivery is slightly erratic, but functional. This must be dealt with via testing. I intend to rectify this as soon as possible.

The process was quite enjoyable. While the concepts involved are not difficult, the idea of using a log host with as little network presence as possible to the point of not configuring an IP address on the logging interface is attractive from a security point of view. It was also a rewarding project that allowed the use of snort in a role other than its frequently used purpose as intrusion detection software. This adds to the value of already useful system administration tool.

The ease of construction of a stealth log host, and relatively simple reconfiguration of the hosts to log to it should make it a part of the system administration tool kit. It offers a lower profile log host with security advantages, especially for a relatively small to moderate group of key servers, but could be implemented all at once, or incrementally I across a whole organization. Much of the configuration task could be scripted. I used a trivial script to move the various arp scripts etc. into their appropriate places, and to run them initially. If desired, the IP configured interface on the stealth host could be removed. This would give the machine an even stealthier profile as long as someone was charged with actually sitting at the machine on a regular basis to assure proper function. My one fear with regard to the removal of the IP configured interface is that "out of sight is out of mind". Another layer in the security posture is never a bad thing as long as it can be conscientiously administered.

Appendix A

HP switch - monitoring port configuration

(pages 6-36 – 6-38 of HP ProCurve Switches 1600M, 2424M, 4000M, and 8000M Management and Configuration Guide)

- From the Console Main Menu. Select:

3.Switch Configuration
3.Network Monitoring Port

- In the Actions menu select Edit.
- Enable monitoring by pressing the space bar.
- Use the arrows to go to the port on which you wish to do the monitoring, and press the space bar.
- Select each port using the arrow keys and the space bar in turn, and exit the facility.[7]

[7] HP ProCurve Switches 1600M, 2424M, 4000M, and 8000M Management and Configuration Guide, 6-34 – 6-38

Appendix B

IPless interace configuration

(refer to Paranoid Penguin: Stealthful Sniffing, Intrus...)[8]

In /etc/sysconfig/network-scripts/ifcfg-eth1

DEVICE=ETH1
USERCTL=no
ONBOOT=yes
BOOTPROTO=
BROADCAST=
NETWORK=
NETMASK=
IPADDR=

[8] Bauer, URL : http://www.linuxjournal.com/article.php?sid=6222

Appendix C

snort configuration file

(refer to Paranoid Penguin: Stealthful Sniffing, Intrus...)[9]

 # snort.conf

```
var EXTERNAL_NET any
var HOME_NET [10.0.1.0/24,10.0.2.0/24]

config dump_payload
config decode_data_link
config interface: eth1
config dump_chars_only
config logdir: /log

preprocessor frag2

log udp $HOME_NET any -> 10.0.1.66/32 514
```


[9] Bauer, URL : http://www.linuxjournal.com/modules.php?op=modload&name=NS-lj-issues/issue102&file=6222l3

Appendix D

arp scripts : arpit, arpdel, arptest

arpit

```
if /sbin/arp | grep stoopy > /dev/null
then
        :
else
          /sbin/arp –s 10.0.1.66 aa:bb:cc:dd:ee:ff:gg
fi
```

arpdel

```
if /sbin/arp | grep stoopy > /dev/null
then
        /sbin/arp -d stoopy.my.domain
else
        :
fi
```

arptest

```
if /sbin/arp stoopy | grep aa:bb:cc:dd:ee:ff:gg > /dev/null
then
        :
else
        /sbin/arp -s aa:bb:cc:dd:ee:ff:gg
        cat /etc/cron.hourly/starp.emerg | mail -sSTEALTH_ARP_CHANGE root
fi
```

Appendix E

/log directory

10.0.2.37	10.0.2.46	dnsa	mailb
10.0.2.38	10.0.2.47	dnsa_save	mailb_save
10.0.2.39	alert	dnsb	login
10.0.2.40	crinch	dnsb_save	login_save
10.0.2.41	crinch_save	lost+found	webber
10.0.2.42	deptd	lpr	webber_save
10.0.2.43	deptd_save	lpr_save	webml
10.0.2.44	deptm	maila	webml_save
10.0.2.45	deptm_save	maila_save	

Appendix F

snort_log_clean

```
# kill snort & move logs to their save directories
pkill snort
mv /log/dnsa/UDP* /log/dnsa_save/
mv /log/dnsb/UDP* /log/dnsb_save/
mv /log/maila/UDP* /log/maila_save/
mv /log/mailb/UDP* /log/mailb_save/
mv /log/deptm/UDP* /log/deptm_save/
mv /log/deptd/UDP* /log/deptd_save/
mv /log/webber/UDP* /log/webber_save/
mv /log/webm/UDP* /log/webm_save/
mv /log/lpr/UDP* /log/lpr_save/
mv /log/crinch/UDP* /log/crinch_save/
mv /log/login/UDP* /log/login_save/
# restart the snort daemon
# and open the ethernet interface for report mail
/usr/local/bin/snort -D -c /usr/local/etc/snort.conf
ifconfig eth0 up
# run all the reports
cd /log/dnsa_save/
nice /log/dnsa_save/0report
cd /log/dnsb_save/
nice /log/dnsb_save/0report
cd /log/maila_save/
nice /log/maila_save/0report
cd /log/mailb_save/
nice /log/mailb_save/0report
cd /log/deptm_save/
nice /log/deptm_save/0report
cd /log/deptd_save/
nice /log/deptd_save/0report
cd /log/webber_save/
nice /log/webber_save/0report
cd /log/webm_save/
nice /log/webm_save/0report
cd /log/lpr_save/
nice /log/lpr_save/0report
cd /log/crinch_save/
nice /log/crinch_save/0report
cd /log/login_save/
nice /log/login_save/0report
# make sure the mail got sent & set clock
```

```
sendmail -q
/root/setclock
#shutdown the outbound interface
ifconfig eth0 down
```

Appendix G

sample report script

0report

```
strings UDP\:514-514 | grep -i error > 0report.out
strings UDP\:514-514 | grep -i fail >> 0report.out
cat 0report.out | mail -scrinch_stealth_log me@my.domain
rm -f 0report.out
gzip -9 -c UDP\:514-514 > crunch.`date +%F`.gz
rm -f UDP\:514-514
```

Appendix H

raw snort packet

11/21-16:59:07.125635 0:A0:BC:D7:8F:91 -> 0:7:E9:11:4B:98 type:0x800 len:0x7A

10.0.2.40:514 -> 10.0.1.66:514 UDP TTL:64 TOS:0x0 ID:0 IpLen:20 DgmLen:108 DF
Len: 80

<78>CROND[19978]: (mailman) CMD (/usr/bin/python -S /var/mailman/cron/qrunner**) .**

Appendix I

sample log report

dnsb my.domain ssh log report showing the update of dns table file information from dnsa.my.domain

```
l icon' time=2s (failed).
<86>sshd[23582]: Accepted publickey for root from 10.0.2.38
port 46822 ssh2.
<38>sshd(pam_unix)[23582]: session opened for user root by (uid= 0)
<38>sshd(pam_unix)[23582]: session closed for user root.
<86>sshd[23590]: Accepted publickey for root from 10.0.2.38
port 46823 ssh2.
<38>sshd(pam_unix)[23590]: session opened for user root by (uid= 0)
<38>sshd(pam_unix)[23590]: session closed for user root.
<86>sshd[23598]: Accepted publickey for root from
port 46824 ssh2.
<38>sshd(pam_unix)[23598]: session opened for user root by (uid= 0)
<38>sshd(pam_unix)[23598]: session closed for user root.
<86>sshd[23648]: Accepted publickey for root from 10.0.2.38
port 46827 ssh2.
<38>sshd(pam_unix)[23648]: session opened for user root by (uid= 0)
<38>sshd(pam_unix)[23648]: session closed for user root.
<86>sshd[23666]: Accepted publickey for root from 10.0.2.38
port 46828 ssh2.
<38>sshd(pam_unix)[23666]: session opened for user root by (uid= 0)
<38>sshd(pam_unix)[23666]: session closed for user root.
<86>sshd[23674]: Accepted publickey for root from 10.0.2.38
port 46829 ssh2.
<38>sshd(pam_unix)[23674]: session opened for user root by (uid= 0)
<38>sshd(pam_unix)[23674]: session closed for user root.
```

Appendix J

nmap scans to the interface without an IP

---
nmap stoopy

Starting nmap V. 2.54BETA30 (www.insecure.org/nmap/)
Note: Host seems down. If it is really up, but blocking our ping probes, try -P0

Nmap run completed -- 1 IP address (0 hosts up) scanned in 54 seconds

---
nmap -P0 stoopy

Starting nmap V. 2.54BETA30 (www.insecure.org/nmap/)
Strange error from connect (64):Host is down
All 1549 scanned ports on stoopy.my.domain (10.0.1.66) are: closed

Nmap run completed -- 1 IP address (1 host up) scanned in 19 seconds

---
nmap -sU stoopy

Starting nmap V. 2.54BETA30 (www.insecure.org/nmap/)
sendto in send_udp_raw: sendto(5, packet, 28, 0, 10.0.1.66, 16) => Host is down
Sleeping 15 seconds then retrying
sendto in send_udp_raw: sendto(5, packet, 28, 0, 10.0.1.66, 16) => Host is down
Sleeping 60 seconds then retrying
sendto in send_udp_raw: sendto(5, packet, 28, 0, 10.0.1.66, 16) => Host is down
Sleeping 15 seconds then retrying
sendto in send_udp_raw: sendto(5, packet, 28, 0, 10.0.1.66, 16) => Host is down
Sleeping 60 seconds then retrying
(etc)

Appendix K

nmap scans to the interface with an IP configured

Interface down

---

nmap sneaky

Starting nmap V. 2.54BETA30 (www.insecure.org/nmap/)
Note: Host seems down. If it is really up, but blocking our ping probes, try -P0

Nmap run completed -- 1 IP address (0 hosts up) scanned in 60 seconds

---

nmap –P0 sneaky

Starting nmap V. 2.54BETA30 (www.insecure.org/nmap/)
Strange error from connect (64):Host is down
All 1549 scanned ports on sneaky.my.domain (10.0.2.17) are: closed

Nmap run completed -- 1 IP address (1 host up) scanned in 524 seconds

---

nmap –sU sneaky

Starting nmap V. 2.54BETA30 (www.insecure.org/nmap/)
sendto in send_udp_raw: sendto(5, packet, 28, 0, 10.0.2.17, 16) => Host is down
Sleeping 15 seconds then retrying
sendto in send_udp_raw: sendto(5, packet, 28, 0, 10.0.2.17, 16) => Host is down
Sleeping 60 seconds then retrying
sendto in send_udp_raw: sendto(5, packet, 28, 0, 10.0.2.17, 16) => Host is down
Sleeping 15 seconds then retrying
sendto in send_udp_raw: sendto(5, packet, 28, 0, 10.0.2.17.179, 16) => Host is down
(etc.)

interface up

---

nmap sneaky

Starting nmap V. 2.54BETA30 (www.insecure.org/nmap/)
All 1549 scanned ports on sneaky.my.domain (10.0.2.17) are: closed

Nmap run completed -- 1 IP address (1 host up) scanned in 1544 seconds

---
nmap –sU sneaky

Starting nmap V. 2.54BETA30 (www.insecure.org/nmap/)
All 1452 scanned ports on sneaky.my.domain (10.0.2.17) are: closed

Nmap run completed -- 1 IP address (1 host up) scanned in 1444 seconds

References

"A Guide to Linux Security:Tools, for Defense" URL : http://www.nwo.net/security/tools.html

Bauer, Mick, "Issue 92:Paranoid Penguin: syslog Configuration". 1 December 2001, URL : http://www.linuxjournal.com/article.php?sid=5476

Bauer, Mick, "Issue 102:Paranoid Penguin: Stealthful Sniffing Intrusion Detection andLogging", 1 October 2002, URL : http://www.linuxjournal.com/article.php?sid=6222

Bauer, Mick, "Issue 102:Paranoid Penguin: Stealthful Sniffing Intrusion Detection and Logging", URL : http://www.linuxjournal.com/modules.php?op=modload&name=NS-lj-issues/issue102&file=6222l3

Campi, Nathan, "Central Loghost Mini-HOWTO", URL : http://www.campin.net/newlogcheck.html

Green, Chris & Roesch, Marty, "Snort Users Manual\\ Snort Release: 2.0.0", Sourcefire, Inc., 2003, URL : http://www.snort.org/docs/writing_rules/

HP ProCurve Switches 1600M, 2424M, 4000M, and 8000M Management and Configuration Guide, Roseville, Hewlett Packard Company, 1999, 6-34 – 6-38.

Roesch, Marty etal., "The Snort FAQ ", 9 April 2003, URL : http://www.snort.org/docs/FAQ.txt

Snort.org URL : http://www.snort.org/

Last Updated: May 23rd, 2017

Upcoming Training

SANS Stockholm 2017	Stockholm, Sweden	May 29, 2017 - Jun 03, 2017	Live Event
Community SANS Ottawa SEC401	Ottawa, ON	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
Security Operations Center Summit & Training	Washington, DC	Jun 05, 2017 - Jun 12, 2017	Live Event
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Rocky Mountain 2017 - SEC401: Security Essentials Bootcamp Style	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
Community SANS Portland SEC401	Portland, OR	Jun 12, 2017 - Jun 17, 2017	Community SANS
SANS Secure Europe 2017	Amsterdam, Netherlands	Jun 12, 2017 - Jun 20, 2017	Live Event
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Minneapolis SEC401	Minneapolis, MN	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Munich Summer 2017	Munich, Germany	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Phoenix SEC401	Phoenix, AZ	Jul 10, 2017 - Jul 15, 2017	Community SANS
Mentor Session - SEC401	Ventura, CA	Jul 12, 2017 - Sep 13, 2017	Mentor
Mentor Session - SEC401	Macon, GA	Jul 12, 2017 - Aug 23, 2017	Mentor
Community SANS Colorado Springs SEC401	Colorado Springs, CO	Jul 17, 2017 - Jul 22, 2017	Community SANS
Community SANS Atlanta SEC401	Atlanta, GA	Jul 17, 2017 - Jul 22, 2017	Community SANS
SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
SANSFIRE 2017 - SEC401: Security Essentials Bootcamp Style	Washington, DC	Jul 24, 2017 - Jul 29, 2017	vLive
Community SANS Charleston SEC401	Charleston, SC	Jul 24, 2017 - Jul 29, 2017	Community SANS
Community SANS Fort Lauderdale SEC401	Fort Lauderdale, FL	Jul 31, 2017 - Aug 05, 2017	Community SANS
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event