Custom Full Packet Capture System

GIAC (GSEC) Gold Certification

Author: Derek Banks, itsecderek@gmail.com
Advisor: Ty Purcell

Accepted: February 27, 2013

Abstract

With server hardware cheaper and faster than ever custom full packet capture systems can now be included in many Information Security budgets. Full packet capture is the most detailed form of network information and can provide historical information about attacks and malicious activity for as long as there is enough storage for the data. There are some commercial offerings that fill this space, but they are expensive and can lack the adaptability and customization that comes with building a custom solution. A custom full packet capture solution can act as flight data recorder for information security analysts and incident response teams to be able to reconstruct what occurred during an attack.

1. Introduction

The goal of a full packet capture system is to acquire the total sum of raw network traffic as it flows from the computers and devices on one network to the destinations on another network. Capturing full packet data, also known as full content data, allows an information security analyst to be able to perform detailed network forensics when a question about what transpired on the network arises. When all incoming and outgoing packets are captured, network traffic can be analyzed to discover and verify communications with malicious external threats. Capturing full content data provides an analyst a higher degree of granularity and flexibility compared to other forms of information such as log files or network flows (Bejtlich, 2012).

There are a number of commercial solutions available that fill the full packet capture market space and they do have unique and useful features, however they all share one thing in common - they are expensive. Using a server with adequate hardware, a custom solution can be created for less cost than a commercial system. Hardware requirements such as storage space, RAM, and processing power will vary based on capture requirements. As a general rule, the faster the disks that can be used, the better such as 15k RPM or SSD disks. The use of a RAM disk can also help alleviate any disk bottlenecks.

In addition, it would be ideal to use a specialized high speed network capture card. Specialized cards, like the DAG Packet Capture Cards offered by Endace, will offload the capture process from the server's CPU and will handle capturing all traffic off the wire. Using a specialized card should eliminate any packet loss from the system assuming that the network speeds match the capacity of the card.

Designing and implementing a custom solution allows analysts to use whatever tool they need to be successful where a commercial solution forces an analyst to rely on only what is provided by the vendor (Bejtlich, 2012). However, while a custom solution is attractive for cost and flexibility, they can be difficult to configure from scratch and there are not a lot of resources readily available on the Internet that describe potential configurations.

Creating a custom solution does not have to be an impossible task. The first step should be to understand the structure of the network that is to be monitored. This is essential for determining placement of the solution. One of the most important locations to monitor will be the ingress and egress points of the network where the true Internet destination and the true internal source IP are captured. In some networks, for example one where a proxy server is in use, this may mean deploying two sensors. Another important location will be those where remote access users come into the environment, such as from a VPN or business partners from a dedicated connection or gateway device (Bejtlich, 2012).

Once placement on the network is determined the network traffic will need to be sent to the packet capture system. Typically this is done through a SPAN port (also known as port mirroring). For each segment of the network that needs to be captured the network equipment will need to have the ability to mirror the traffic to the capture system (Bejtlich, 2005) Another option to gaining access to the network traffic is through the use of aggregating taps. Device such as those offered by NetOptics will allow for traffic to be tapped then aggregated to the packet capture device.

The pros and cons of using a SPAN port or network taps will vary from environment to environment. SPAN ports may be susceptible to dropping traffic. This can be difficult to detect and alert on. Network taps are less likely to drop traffic, but are in-line devices that may have to be part of troubleshooting procedures if there is a production network issue.

It is also essential to understand the threats that the organization faces. Careful analysis of the threats an organization faces should drive the system design. For example, is spear phishing a concern? If the answer is yes, then the solution will need to have a way to extract out important information with regard to email traffic. What about covert data channels – is there a risk in the environment for this? Some thought should be given to what kind of application information should be extracted and monitored by the analysts using the system. Having terabytes of packet captures is not worth much unless the data can be acted upon (Heins 2011).

The next item to consider is the volume of traffic to be captured and the required retention time for the data. This will determine the amount of disk space that the system needs. Full saturation on a 1Gbps link will yield approximately 6 terabytes of traffic a day. Chances are that a given 1Gbps link is not entirely saturated throughout an entire day and the throughput will fluctuate. A good number to work with for a gigabit link is 1TB per day (Heins 2011).

Another obstacle to overcome is the time sink involved with searching for data. Being able to search for a data point such as an attacker IP address or domain name must occur within a reasonable time frame. Searching through the large raw packet data is the least efficient way to search. To reduce the amount of time for any given search, indexing and extraction of application data (such as SMTP or DNS) must be a part of the system. (Heins 2011).

2. Overview of capturing traffic

While a custom solution can be created on a Windows Server platform, Linux provides better native tools and scripting capabilities. The most commonly used packet capture library is libpcap and one of the most popular applications for capturing traffic on either server platform is tcpdump (Bejtlich, 2005).

The core of the design revolves around tcpdump, usage of command switches for tcpdump, and some bash scripting for post capture processing.   Tcpdump will be used to write a packet capture (pcap) file to disk for a specific interval of time in seconds and then name this file based on the date and time. Note that it may be a good idea to exclude certain types of traffic from the capture as well, for example encrypted traffic that may not be useful in the capture and that can potentially be monitored by another layer of defense. Once the traffic is captured, a post processing script runs in parallel to the capture - this is where indexing, extraction and certain analysis commands take place.

Before starting with capturing traffic, make sure that the server OS is loaded and a partitioning scheme is chosen that will isolate the packet capture files to a partition. The reason for this is to ensure that there are not issues with the operation of the OS due to the capture filling up disk space. An example of a portioning scheme is:

Table 1.1 – example partitioning scheme
/boot -500MB
/home – 10GB
/ - 10GB
/var – 10GB
/usr/local -10GB
/var/pcap – remainder of the disk space

Also note that since the sole purpose of this server should be capturing traffic, a minimal OS install should be performed and only packages necessary to the functionality of the system should be added. Alternatively a system hardening guide should be followed to make sure services that are not needed are turned off and removed. There are many resources online that provide guidance on Linux system hardening.

2.1. TCPDUMP Command Details

Assuming that the base of the system is in place such as the server OS has been installed, disks are partitioned correctly (capture area is contained in its own mount point, etc), and tcpdump is functioning, the first step will be to create a script that will run tcpdump with the arguments necessary to capture data. Making the tcpdump command and arguments part of a script will allow for changes to be a bit easier and will also allow it to be called at start up (either via /etc/rc.local or as an init script). Table 1.1 contains an example of the syntax of the tcpdump command that would capture all traffic on the given interface and a brief description of what each switch does. Appendix A contains a sample tcpdump script.

Table 1.2 – tcpdump command	
tcpdump -s0 -nn $EXCLUDED -G60 -w "%Y-%m-%d-%H%M.pcap" -i $IFACE -z pcap_parsing_script.sh"	
Switch explanation:	
-s0	Will capture the entire packet
-nn	Do not resolve host names or port names
-G60	Rollover to new capture file every 60 seconds
-w	Write capture to file, in this case the date in year, month, day, Hour, minute with .pcap extension (for consistency and to aid in clean up)
-i	Interface that tcpdump listens on, in this case, a script variable is used
-e	Interface the capture listens on (variable in script or network device)
-z	Post rotate command – for operation on the packet after capture

The –s switch captures "snaplen" bytes of data from each packet. Giving it a value of 0 makes it capture the default value of 65535. This effectively means that the entire packet will be captured (Jacobson, 2009).

The –nn switch will prevent resolution of host names and port names (Meissler, n.d.). Preventing the resolution of domain names is important as it will reduce network traffic on the network. Also, in many circumstances it is not desirable to look up external host names for Operations Security (OPSEC) reasons. Not resolving the port name keeps the source and destination ports the actual port number than what tcpdump matches to the most likely protocol on that port. This is important because you do not want an attacker to know that you are monitoring because of DNS lookups to a server in their control.

It may be useful to exclude certain types of traffic from being captured, especially if there is little chance of it being useful in an investigation and it is something that may be monitored and controlled in some other way. For example, if the organization allows outbound ssh to certain locations and alerting of unauthorized traffic is handled by some other device then it may not be necessary to capture port 22 traffic. Multiple exclusions can be handled by calling a script variable and using a file with the appropriate contents. An example exclusion file is included in Appendix B; however, care should be taken in how it is used. Exclusions that make sense in one network environment may not make sense in another.

The –G switch rotates the capture file specified by the –w switch. The number value is the amount of seconds for the capture to go before a new file is written. So a value of 60 will result in a new file every sixty seconds. The –w switch writes the captured packets to file and should include in the file name statement syntax for the time and date conforming to strftime, otherwise, the capture file will be overwritten (Jacobson, 2009). The can of course be adjusted according to preference as long as it conforms to the format.

The –z switch is used to make tcpdump run a command on the file it just finished writing to disk. This is commonly used to compress pcap files after capture, but can run any program on the system (Jacobson, 2009).

2.2. Post Capture Processing

Rather than calling a compression utility or some other application, the –z switch will call a bash script as its argument that contains multiple commands that will process the pcap file in parallel to the next capture file starting. In other words, when tcpdump rolls over to the next file, the –z switch will be used to process the command given, in this case a bash script, and as it starts a new capture, the commands in the bash script begin executing on that file.

Using a bash script as the command that is run via the –z switch has some advantages. It allows for one location to change the processing on the pcap file. It also allows for easier addition of new functionality in the future. An example of a post processing script is located in Appendix C.

Rarely should an analyst start investigating by looking at raw packet data. There should be some indicator of interest elsewhere that spurs investigation into the packet data (Bejtlich, 2012). These indicators can be from other logging sources or from eventual alerting set up on the full packet capture system, but at some point the analyst will have to search for data. This is one of the main functionalities of post capture processing – to create an index of IP address and application data to make searching much more efficient.

2.2.1. IP Address Indexing

One of the most important post processing routines is to abstract each pcap file into a corresponding index of IP addresses that communicate during the time frame of the packet capture. This index file should contain data that makes it easier to search for communication in the raw packet capture rather than searching through the large raw files themselves. It can be netflow type information or simply just an IP address seen as a destination depending on preferences. The index will act as metadata of sorts for the raw packet captures. The approach is to simply retain the file name of the pcap for the index file and write them to a separate directory. This makes a one for one relationship for pcap to index file.

The main concept here is that when there is a call for data – for example was there communication with a website at IP address 1.2.3.4 on December 21, 2012 – that searching for the results does not take an inordinate amount of time. Searching through a full day's worth of raw packet captures could take six hours or more (Heins 2011). A more efficient approach is to search through the index files because they will be much smaller than the raw packet data and therefore much quicker to search through.

The index files will be smaller in size and compress well since they will be text data. This means that they can be stored for much longer than the raw pcap files and a record of communication can be retained for many months or years. So even if the pcap data has been removed months after the data call comes in, at least there was record that something occurred.

The index files can be created in many ways and this is where one of the more complicated aspects of this system will exist for those inexperienced in writing code. There are perl scripts available on the Internet for parsing packet captures that can be relatively easily modified to accomplish the task (see appendix D). If there is in-house talent for C or python coding, a program written in either of those two languages should be faster in processing time when compared to a perl script.

2.2.2. Application Data Indexing

Application data (DNS, SMTP, HTTP, etc) extracted from the packet capture files can also prove to be useful. The idea here is the same as the IP address indexing in that the time needed to perform any search for data is done outside of the raw pcap files in much smaller abstracted files.

For example, if certain fields of email traffic are being extracted into application index files, like senders and receivers, and there was an indication that a malicious mail was sent to someone in the organization, it can be quickly verified if the mail had been processed by simply searching through the indexes. If there is a match, then the full content data can be extracted from the raw pcap files.

There are various ways that creating an index of application data can be accomplished. One method for accomplishing this is to use ngrep and extract out the data into a separate directory from the packet captures as text files (just like the IP address index above). The sample post processing script in Appendix C demonstrates a method for extracting SMTP data through the use of ngrep. That method can be adapted for other types of application data as well.

2.2.3. RAM Disk

It may be necessary to use a RAM disk to initially capture the pcap files to if packets are being dropped, then offload them to disk storage. This will depend on the I/O capabilities of the disk in the server and the amount of RAM in the server will need to be adequate. This can be accomplished by setting the initial capture script to write to the RAM disk, and then offload to the disk based storage as part of the roll over processing.

There are various how to articles on the Internet on how to set up a RAM Disk in Linux, in fact, there are already RAM disks in use normally; however they generally default to 16MB. One approach that works in Red Hat and Centos based distributions is to set the initial RAM disk size at boot by passing a kernel parameter of ramdisk_size=<size of RAMDisk> (Emery, n.d)

If it were determined the system needed a 1GB RAM disk to allow for the amount of time allotted to the pcap write process, then the parameter passed to the kernel would be ramdisk_size=1048576. Some trial and error, testing, and observation of network throughput over time may be needed to determine the right size of the RAMDisk.

Once the size is set, there are a few more items that need to be addressed. The RAMDisk will need a file system and need to be mounted to a mount point such as /mnt/ram. It should also happen at boot time, so ideally the creation of the file system and mounting would happen in an init.d script in /etc/init.d. This may vary based on the flavor of Linux that is being used.

2.2.4. Maintenance

Some maintenance of the pcap files will be necessary to avoid filling the volume where the data is being written. One of the most basic ways to accomplish this is to observe the packet capture data and as the disk starts to approach 80-90% full a cron job can be implemented that calls a script that cleans up any raw pcap files after a certain amount of days.

For example, if the storage volume was at 85% capacity and there were 32 days worth of pcap files, the script located in Appendix E could be modified and used to run as a daily cron job and clean up a pcap file older than 35 days.

There may be other maintenance tasks that would be useful such as compressing and offloading index files to longer term storage on the network for archive purposes. Regardless of the requirements however, some thought should be given to what maintenance is necessary on the system and can be automated.

2.3. Basic Analysis

So now that there is a massive archive of raw packet capture files and an index of them to enable efficient retrieval, what can be done with the all of this data? The goal is to take indicators and warnings, either from a mechanism set up on the full packet capture system or from some other source and detect and respond to potential intrusions (Bejtlich, 2012).

One of the most basic and recurring tasks will be to find communication to an external website from an internal host. Assuming that http traffic is being indexed, using native Linux commands will enable the searching to take place. Recall that every pcap will have a corresponding text based index – in this case http traffic – therefore the index can be searched via grep in a relatively short amount of time. Table 2.1 shows a grep command that would list every file match for the given file name range.

Table 2.1 – example grep search for indexed data	
grep –l www.evil.com 2012-12-21*	
Switch explanation:	
-l	List all files that contain matches
Sample output:	
2012-12-21-1440.pcap	

Once the file names where the traffic is known the analyst must make a determination of what internal IP address initiated the communication. For simplicities sake, assume that the traffic in question is from one internal IP address and is contained in one pcap file. Once the internal IP address is known, traffic for just that host can be extracted from the larger pcap with tcpdump. Using 192.168.1.15 as an example internal host, table 2.2 demonstrates a tcpdump command that will get a smaller pcap file narrowed down to the internal IP communicating to the external site.

Table 2.2 – example tcpdump extraction	
tcpdump –r 2012-12-21-1440.pcap –w /tmp/web.pcap host 192.168.1.15 and port 80	
Switch explanation:	
-r	Read pcap from file
- w	Write pcap to file

As this type of investigation into the traffic will be recurring, the process can and should be scripted so that an analyst can spend less time performing manual investigation steps.

Having historical network traffic data opens up many interesting possibilities for analysts. One of possibilities is to use the archive of network traffic when new IDS signatures are available to determine if an attack has occurred in the past and potentially an ongoing compromise that had not yet been detected.

Assume that a new attack has been discovered and there is a snort signature or some other indicator of compromise such as malware command and control traffic or hostnames or IP addresses potentially involved in ongoing attacks. Software such as Snort or Yara can be used to search historically through the packet captures to aid in discovering if compromise has occurred.

Snort is very flexible and has many options for reading existing packet capture files. It has options to read from a single file, a directory of packet captures, from a list of pcap files, and even shell style filters (Esler, 2012). If a Snort signature is available or can be written by the analyst, the historical traffic can be searched for specific attacks that may have occurred in the past.

Snort does have some overhead associated with it and can potentially take some time to run. Another option to parse through the pcap files for indicators of compromise is Yara.

Yara is a tool aimed primarily for malware classification and can parse file contents based on descriptions in textual or binary form. Yara uses a configuration file that analyst will have to create (Alvarez, 2011). For example, if www.evilbadness.com was an indicator of compromise, the contents of table 2.3 demonstrates what the corresponding Yara rule would look like.

Table 2.2 – Example Yara Rule

```
rule evilbadness : evilbadness
{
   strings:
      $a = " www.evilbadness.com "
   condition:
      $a
}
```

If the directory that stores the pcap files is /var/pcap and the Yara configuration file is in /home/user, then the syntax for running Yara is:

```
yara /home/user/yaraconfigfile /var/pcap
```

Multiple rules can be contained in a Yara configuration file. The output from matches to rules will be the name of the rule and the path to the file that hits. Table 2.3 shows what a match to the above rule would look like.

Table 2.3 – Example Yara Rule
evilbadness /var/pcap/2012-12-21-1028.pcap

3. Conclusion

Full packet capture systems are no longer too expensive for most organizations to deploy. Using a moderately powered server and storage system and a specialized network capture card combined with open source tools and some creativity allows information security analysts to be able to collect invaluable full content data.

Creating a custom system to collect full content data has many advantages over using a commercial solution. One of the largest advantages is the use of open source tools that are commonplace. This allows the analyst to be flexible in that the processes and tools used to create the system can be adapted to many different network environments and not reliant on a specific vendor's tools and methods. Using a custom platform means that any tool needed to accomplish the task at hand can be used and forces the analyst to obtain a greater understanding of the operation of the network and how to detect and respond to potential compromise.

4. References

Heins, Randy. (2011). The Full Packet Capture Pocket Fisherman. Retrieved from http://www.cert.org/flocon/2011/presentations/Heins_Indexing.pdf

Bejtlich, Richard. (2005). The Tao of Network Security Monitoring: Beyond Intrusion Detection: New York, NY: Addison-Wesley

Bejtlich, Richard. (2012). Why Collect Full Content Data?. Retrieved from http://taosecurity.blogspot.com

Emery, Van. (n.d). Linux RAM Disk mini-HOWTO. Retrieved from http://www.vanemery.com/Linux/Ramdisk/ramdisk.html

Jacobson, L. (2009, November 19). *Tcpdump*. Retrieved from http://www.tcpdump.org/tcpdump_man.html

Meissler, D. (n.d.). *A tcpdump tutorial and primer*. Retrieved from http://danielmiessler.com/study/tcpdump

Esler, Joel. (2012, December 4). *Reading Pcaps*. Retrieved from http://manual.snort.org/node20.html

Alvarez, Victor Manuel. (2011, April 8) Yara in a Nutshell. Retrieved from http://code.google.com/p/yara-project/

5. Appendices

Appendix A – Sample TCPDUMP Script

```
#! /bin/bash

IFACE="eth1"
HOMEDIR="/usr/local/bin"
# where the data is output
DATADIR="/var/pcap"

cd $DATADIR

# read the rules from the file for the port/protocols you don't want in the dump
EXCLUDED=`cat $HOMEDIR/tcpdump-exclude`

# create a variable for the program to use
TCPDUMPRUN="/usr/local/src/tcpdump/sbin/tcpdump -s0 -nn  $EXCLUDED -G60 -w
"%Y-%m-%d-%H%M.pcap" -i $IFACE -z /var/pcap_scripts/pcap_parsing_script.sh"

# run the tcpdump process
`$TCPDUMPRUN`

#done
```

Appendix B– Sample Excluded File Contents

```
not (port 22)

and not (port 123)

and not (ip proto 29)

and not (ip proto 47)

and not (ip proto 50)

and not (ip proto 51)

and not (ip proto 55)
```

Appendix C– Sample Post Processing Script

```
#! /bin/bash

pcapfile=$1
basename=`basename $pcapfile`
tooldir='/usr/local/bin'
datadir='/var/pcap'
indexdir='/var/pcap/index'

#cd $indexdir
echo "Processing $pcapfile"

# Extract all IPS's into an index file
`/usr/local/bin/parse_pcap_script.pl "$datadir/$pcapfile"  > “$indexdir/$basename”’

# Extract SMTP information from PCAP file
/usr/local/bin/ngrep -Wsingle -qtiI "$datadir/$pcapfile" "Subject:|filename=|RCPT
TO|MAIL FROM:|X-mailer:|To:|From:" port 25 | grep -i "Subject:\|filename=\|RCPT
TO\|MAIL FROM\|To:\|From:\|X-mailer" > "$data
dir/smtp/$basename"

#Extract http information from PCAP file
/usr/local/bin/ngrep -Wsingle -qtiI "$datadir/$pcapfile” tcp and dst port 80 or 443 |  grep
–I “http” > "$datadir/http/$basename"

#Run Snort on PCAP file

/usr/local/bin/snort -q -r "$datadir/$pcapfile" -c /etc/snort/snort.conf
```

Appendix D – Sample Pcap Parsing Script

Adapted from http://hype-free.blogspot.com/2010/03/parsing-pcap-files-with-perl.html

```
#!/usr/bin/perl

use Net::TcpDumpLog;
use NetPacket::Ethernet;
use NetPacket::IP;
use NetPacket::TCP;
use strict;
use warnings;

my $log = Net::TcpDumpLog->new();
$log->read("$ARGV[0]");

foreach my $index ($log->indexes) {
  my ($length_orig, $length_incl, $drops, $secs, $msecs) = $log->header($index);
  my $data = $log->data($index);

  my $eth_obj = NetPacket::Ethernet->decode($data);
  next unless $eth_obj->{type} == NetPacket::Ethernet::ETH_TYPE_IP;

  my $ip_obj = NetPacket::IP->decode($eth_obj->{data});
  next unless $ip_obj->{proto} == NetPacket::IP::IP_PROTO_TCP;

  my $tcp_obj = NetPacket::TCP->decode($ip_obj->{data});
  my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime($secs +
$msecs/1000);

print sprintf("%02d-%02d %02d:%02d:%02d.%d",
    $mon, $mday, $hour, $min, $sec, $msecs),
    " ", $ip_obj->{src_ip}, ":", $tcp_obj->{src_port},
    " -> ", $ip_obj->{dest_ip}, ":", $tcp_obj->{dest_port}, "\n";
}

close STDOUT
```

Appendix E – Sample Maintenance Script

```
#!/bin/bash
```

```
#this script will remove pcap files that are older than
#60 days from the data directory

DATADIR=/var/pcap
DAYSOLD="60"
cd $DATADIR
find $DATADIR -name "2012-*" -mtime +$DAYSOLD -exec /usr/bin/nice -n 10 rm -f {} \;
```