Forensic FileSystem

User Guide

version 1.08

Introduction

The Forensic File System is a full 24 x 7 continuous network capture technology with an industry standard file system based upon the Intel EFI Disk Storage Specification which can perform lossless packet capture of all network data on a target network at multi-gigabit speeds.

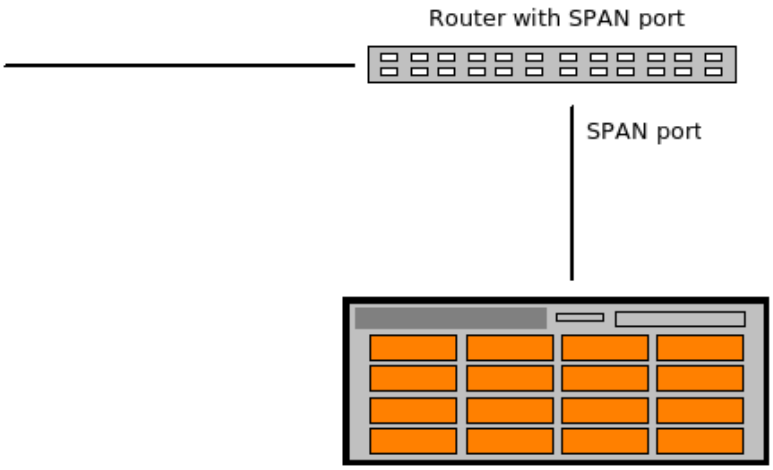
The technology is optimized for continuous capture at line rates up to 10 gigabits per second on specific hardware models. The central component of the platform is the forensic file system, which exposes the captured data as pcap formatted files (libpcap, tcpdump) and as virtual interface drivers through industry standard network and file system interfaces. Because the Forensic File System exports its data via these industry standard interfaces, it supports thousands of industry standard and open source network management, network analysis, network forensics, network monitoring, and network profiling applications out of the box. The Forensic File System supports real time monitoring for a wide variety of intrusion detection systems and network monitoring applications, enabling lossless packet capture and improved performance.

The Forensic File System can support very large storage architectures, and the Forensic File system platform can be configured to support up to 9,007 Terabytes of disk storage in a single cluster configuration (9.007 Petabytes) via Fiber Channel attached storage.

The Forensic File System supports OpenCalea and is pre-installed with a full suite of CALEA applications for law enforcement. The Forensic File system is also provided in a virtual platform software-only configuration and provides remote workgroup collector capability for workgroups and home-based use with a fully distributed storage architecture.

The Forensic File System provides capabilities that range from a laptop based system running as a virtual platform up to multi-pedabyte storage clusters supporting speeds in excess of 10 gigabits per second stream-to-disk.

Supported Topologies

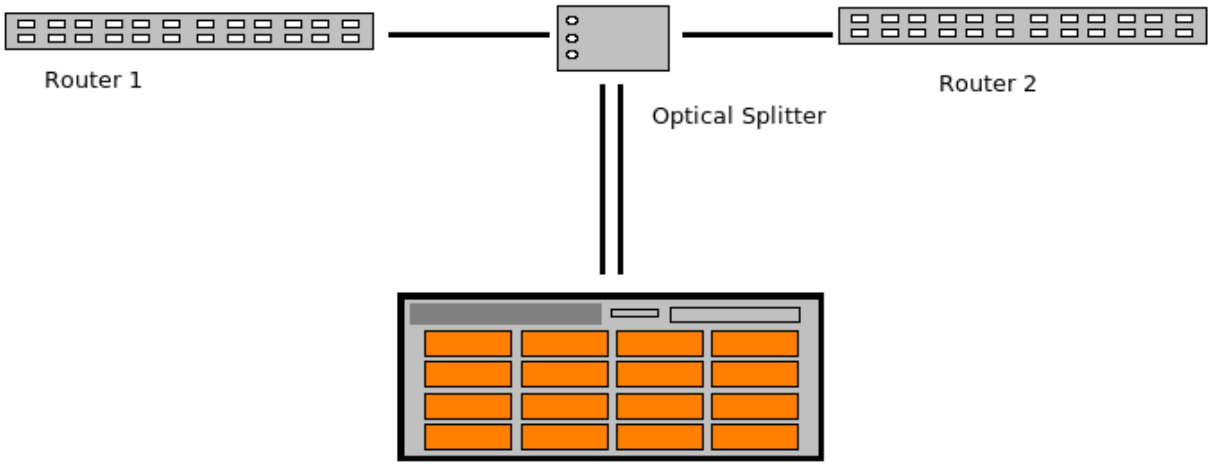


A SPAN (Switched Port Analyzer) configuration. In this example, the appliance is connected to a specific port on a router, and the router is configured to mirror the traffic from all ports and forward it to the spanned port. This model allows a series of routers to forward traffic to the appliance.

The Forensic File System can be configured to capture all network data for a campus-wide networking environment with port mirroring at the router (SPAN configuration) and can be deployed with optical splitter devices for in line monitoring.

In a SPAN (Switched port analyzer) configuration, the platform is attached to a port on a router which has been configured via the router software to mirror all packets received by the router to the spanned port. Packet mirroring is the process of making a copy of every pack the router receives or sends and forwarding the copy to the SPAN port. This configuration allows a network of routers to be setup to forward traffic to specific ports on a selected target

router port for capture and analysis.



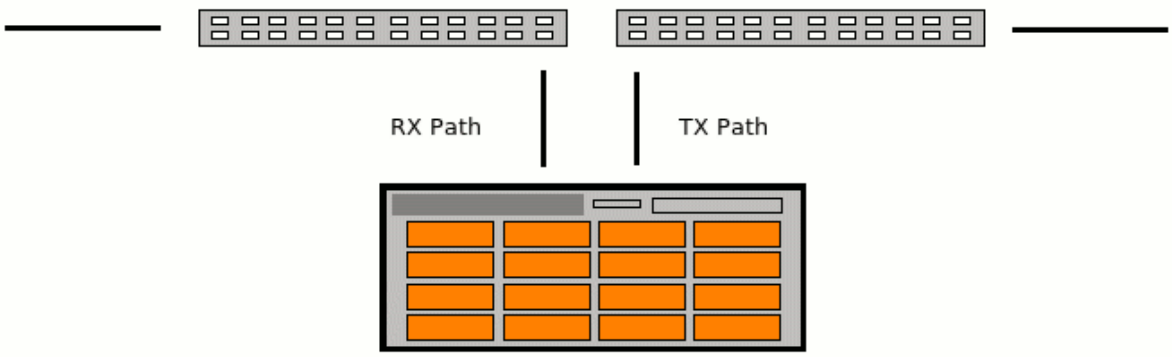
The software supports Optical Splitter configurations which split out the receive RX channel and Transmit TX channel into seperate capture streams. The appliance recombines the channels and presents the captured data to analysis applications as a consolidated stream of network packets.

The Forensic File System also supports capture of Asymmetrically Routed network traffic and allows captured RX/TX data channels to be reassembled in the platform and presented to applications as a consolidated virtual network interface.

Asymmetric routing is typically employed on large networks which require increased bandwidth or fault tolerance. In the asymmetrical routed model, network routers are configured to divide the transmission (TX) and receive (RX) traffic into distinct channels and transmit them via parallel router fabrics.

The advantage of asymmetric routing in large networks is the channels can be mirrored and traverse dedicated router segments with improved bandwidth. Another advantage is the ability to mirror entire network segments in parallel

to avoid network failures in the event one of the router fabrics should fail.



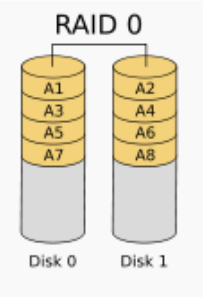
The software supports assymetrically routed traffic where the RX and TX channels of large networks are trasmitted on separate RX/TX router fabrics for fault tolerance or performance. The software Appliance can capture and reassemble the RX/TX channels and combine them into a consolidated stream for presentation to network analysis appliacations. Assymetric routed networks can employ SPAN ports configured to mirror the TX and RX channels and forward packets to the software Appliance for capture and later anaysis.

The Forensic File System supports all common communications topologies including wide area and local area network technologies.

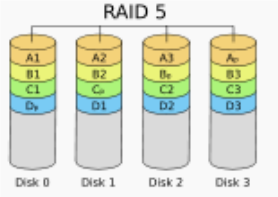
The virtual platform can be installed on a laptop computer or home networking system running Windows or Unix and provide support for network forensics and applications running on lower bandwidth networks or wide area or wireless networks.

Storage Architectures and Features

The Forensic File System provides support for hardware and software RAID0, RAID1, RAID5, RAID10, 64K Block Striping, Segment Striping and Mirroring, an industry standard Intel EFI based Segment Journal, and Native Clustering and can address up to 9,007 Terabytes (9.007 Petabytes) of capture storage and supports fiber channel storage in clustered configurations.

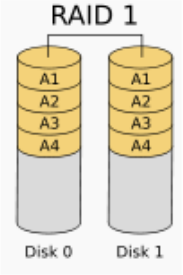


RAID 1 provides the highest level of fault tolerance but the poorest use of available disk space. RAID 1 mirrors identical copies of the data on a one or more drives. RAID 1 provides the highest levels of read performance and can round robin read requests.



RAID 0 stripes 64K segments accross multiple drives and combines all the drives into a continguous unit of storage.

RAID 0 provides very high levels of performance but provides no fault tolerance. If a drive fails, then the data on the drive array is lost.



RAID 5 provides fault tolerance by using xor checksum parity to to recreate data if a drive fails. RAID 5 provides excellent disk performance with fault tolerance since parity blocks are striped accross the array. RAID 5 is faster than mirroring but does sacrifice a drive in each array to store the parity data. RAID5 is about 75% of RAID0 performance

This diagram depicts RAID 0, RAID 1, and RAID 5 features and comparison. The Interceptor Appliance supports these RAID levels at a hardware level and at very high levels of performance.

The platform supports high-speed hardware RAID striping and provides the ability to configure hard drive arrays of 1 to 12 disk devices per array controller. The arrays can be configured to support RAID levels 0 (striping), 1 (mirroring), 5 (parity striping), and 10 (1+0 = striped mirrors). RAID levels 0 and 5 are optimal for stream-to-disk applications. RAID 0 provides

the highest write and throughput performance and RAID 5 provides excellent fault tolerance in the event of hard disk failures.

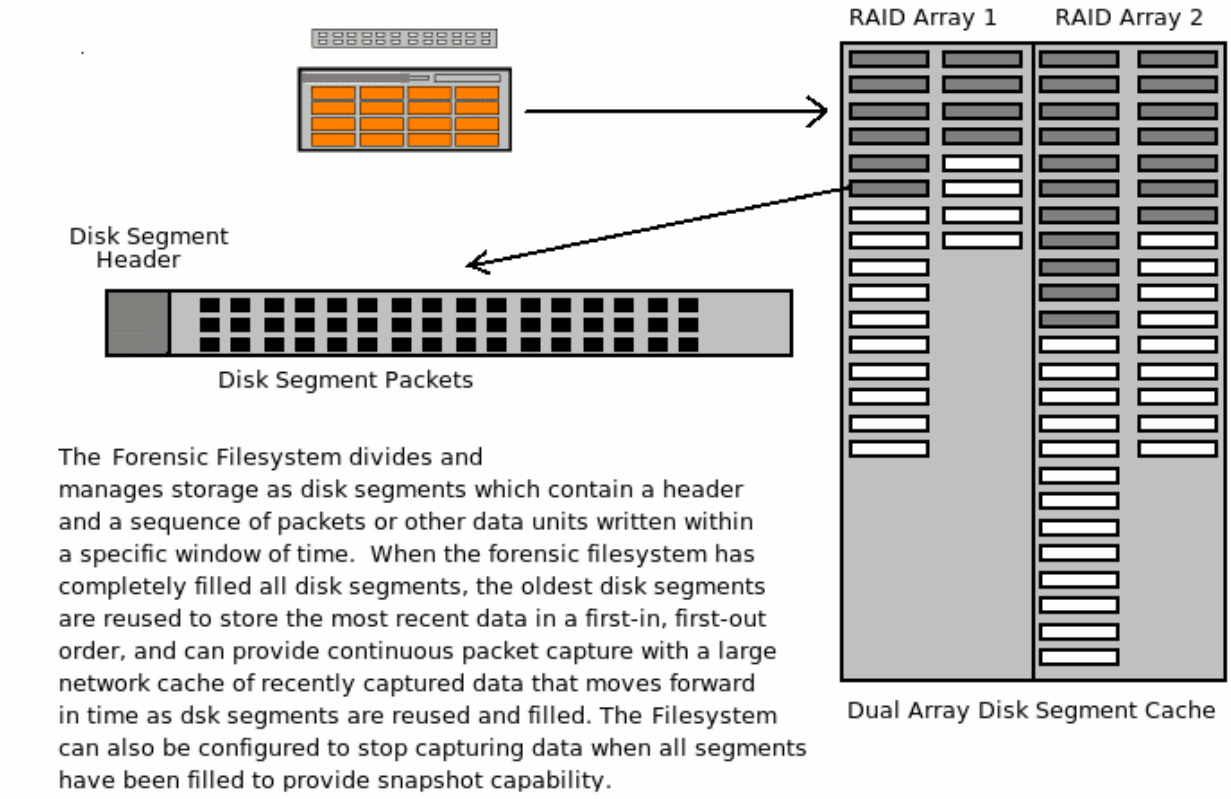
RAID 5 systems typically perform at 75% of throughput when compared to RAID 0 configurations under very heavy traffic loads. Raid 10 (also called RAID 1+0) is an extremely fault tolerance model designed for high performance in reading data, but is less than ideal with write intensive applications such as packet capture. Raid 1+0 creates RAID 0 arrays of mirrored pairs of drives, and will consume 50% of the platform hard drive capacity as mirrored storage.

For deployment of an Forensic File System where fault tolerance is desired, the disk arrays in the platform should be configured to RAID 5 by default. For those applications which require lossless packet capture in environments employing 10 Gb/s Ethernet or capture rates greater than 4 Gb/s, it is recommended to use RAID 0 in order to achieve the highest disk throughput possible. Fault tolerance can be achieved via RAID 0 by deploying multiple platforms running RAID level 0 on the same monitored network segments.

Forensic File system Storage Features

Description

The forensic file system is a non-volatile (permanent, persistent) Intel EFI industry standard caching system which is comprised of variable units of disk storage called “disk segments”.



Disk segments and the underlying methods for managing their organization are based upon the simple concept of time. Segments represent a series of data events which occurred and were recorded over time, such as a stream of network packets.

Perhaps the best description of the Forensic File system is to view segments as “bundles” of packets collected and bound together based upon windows of

time. Disk segments contain variable numbers of packets and each disk segment has a header which describes the content of the segment, the number of data elements within the disk segment, and the starting and ending time of the events which are contained within the segment.

Disk segments are units of time, and the Forensic File system storage architecture is comprised of industry standard disk caches which manage collections of segments by policy on a first-in, first-out basis. This architecture allows the older segments to be reused when the system storage has become full, with incoming data overwriting older disk cache segments. This architecture creates a large time based buffer of network activity which slowly moves forward in time, which depending upon the size of storage on the platform and the network traffic loading, can span several days or even weeks of network activity. This data can be reviewed and analyzed by hundreds of available applications.

A Forensic File System platform can be configured to stop capturing when the disk cache becomes filled rather than reuse the oldest disk segments, essentially making a snapshot of the captured network data in addition to supporting dynamic reallocation of the oldest disk segments in the platform.

Disk Segments are typically configured as variable sized blocks and are the primary unit of caching and IO used by the Forensic File system and can range from 16 megabytes in size up to 256 megabytes in size, based upon the storage configuration. The larger the size of disk segments, the more physical storage than can be addressed.

Storage Management Options

The Forensic File System storage model (logical volumes) can be configured to support the following storage management configurations.

Basic Data Storage Models

64K Block Striping (RAID 0 model)
Segment Striping
Segmented (Dynamic Segments)
Work Group Collector

Distributed Storage (Multiple Readers, Fault Tolerant)

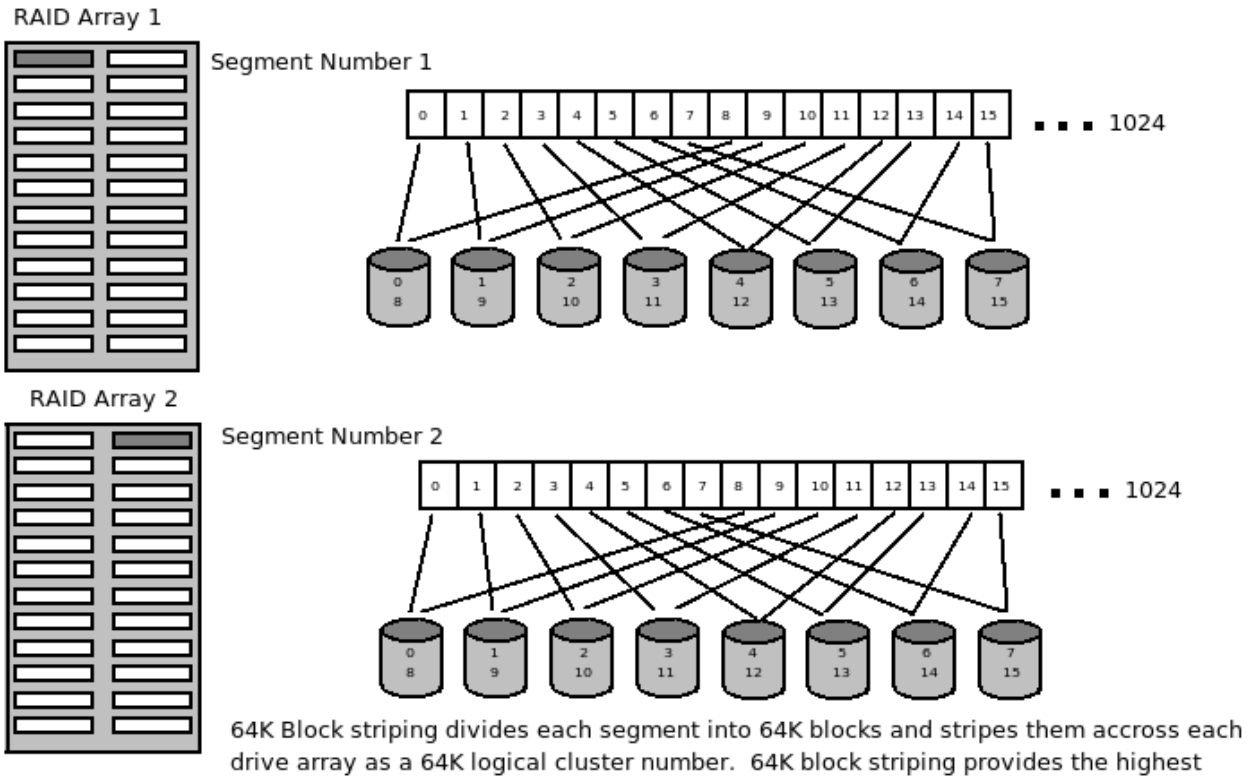
Clustered Dynamic Segment Archive (future release)

Data Collection and Reduction

Work Group Collector
Work Group Collector Mirroring (future release)

64K Block Striping

The Forensic File system 64K Block striping option provides the highest level of disk performance if the underlying arrays are configured as RAID 0.



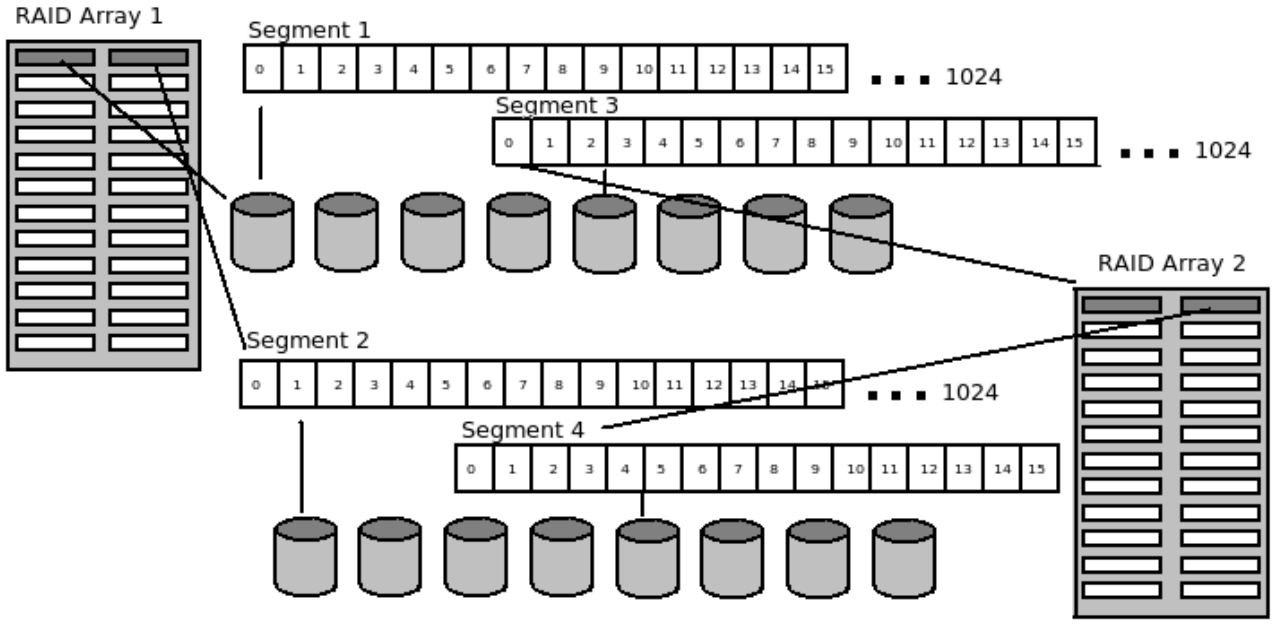
64K Block striping divides each segment into 64K blocks and stripes them accross each drive array as a 64K logical cluster number. 64K block striping provides the highest level of performane and IO scaling but is not fault tolerant unless the arrays are setup as RAID 5.

However, 64K striping also provides the lowest level of fault tolerance unless the underlying arrays are configured in RAID 5 (which can reduce 64K striping performance to less than expected or ideal levels). With 64K Striping enabled and the platform arrays configured as RAID 0, a failure of any drive will result in the loss of all captured data. 64K striping with RAID 0 is ideal where the highest disk throughput is required and a duplicate platform is present to monitor a specific network segment for fault tolerance (if desired). 64K striping requires that all arrays be of the same logical size (but not physical size). If additive storage is desired (the ability to dynamically

increase the storage size of your platform during operations) 64K striping is not the correct option – a dynamic segmentation option is a better choice. If the system is merely being deployed for high speed capture and event analysis without the requirement for fault tolerance or if a duplicate or mirrored platform if present, 64K striping with RAID 0 is an excellent choice for setups requiring the highest level of disk throughput possible.

Segment Striping

Segment Striping provides the performance advantages of 64K block striping with an added level of fault tolerance with RAID 0 configurations.



Segment Striping changes the striping block size from 64K to the size of a disk segment. In this model, each allocated disk segment is "round robin" allocated from total number of arrays with even number disk segments allocated from even numbered arrays and odd numbered segments allocated from off numbered arrays. This model requires that each array be the same logical size (based on segments) not physical size. This model provides near equivalent performance to 64K block striping under heavy loads in terms of disk throughput, and also provides some fault tolerance.

Segment Striping employs similar methods to 64K block striping, but provides the ability to recover at least half of the captured network segments when configured in RAID 0 mode should a hard disk fail. This model allows very

high levels of disk throughput and equals and in some cases exceeds the disk throughput of 64K block striping under extremely heavy loads with multiple interface adapters since disk segments are coalesced contiguous runs of disk sectors and this model forces parallel writes of multiple segments across the disk arrays. Running at RAID 0, this option will allow the recovery of at least half of the stored segments in the event of a disk failure. This option can be configured to use RAID 5 on the underlying disk arrays and provide resilient fault tolerance and high levels of performance.

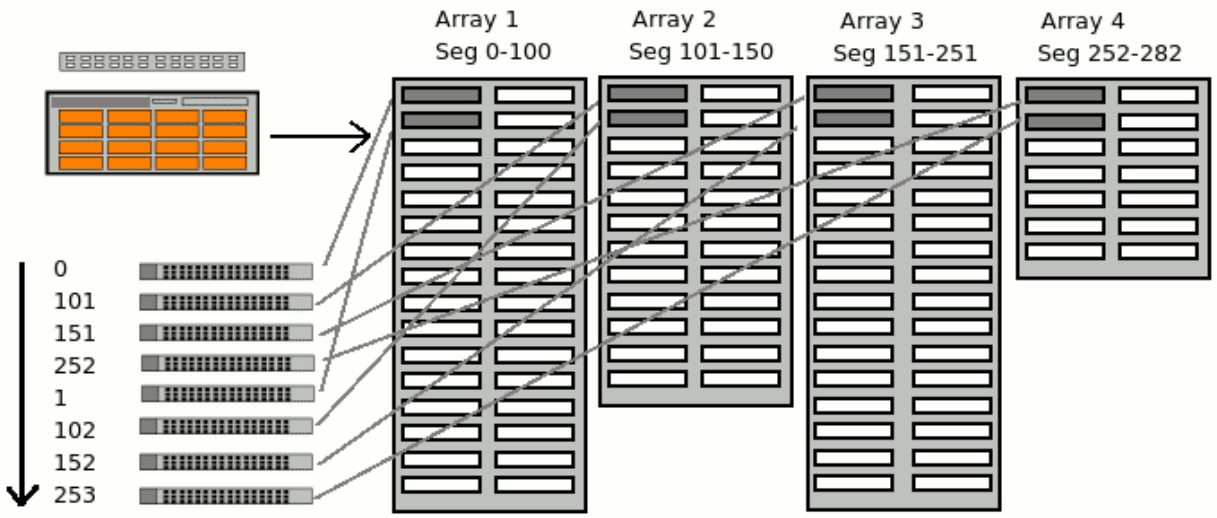
This model is best suited for capture platforms which are archiving segments at high data rates to a back end Fiber Channel cluster of storage platforms for later analysis by clustered Forensic File Systems used a "readers" of the data. It is not the most ideal configuration option for a stand-alone Forensic File System, and is better suited to clustering and segment mirroring configurations, however, this mode can be used standalone and provides extremely high levels of performance and disk throughput when the underlying arrays are configured as RAID 0.

Segmented (Dynamic Segments)

Segmented volumes are the most flexible option and provide the ability to add or remove storage nodes or arrays from a running system real time, and unlike 64K block striping or Segment Striping models, does not require that all the arrays be of the same logical size. Segmented volumes can be configured to support arrays or individual disk drives of variable sizes.

Segmented volumes are more complex to setup and administer, but provide a high degree of flexibility and fault tolerance, as well as high performance. In a segmented volume, each partition or array is managed as a self-contained unit of storage along with any disk segments it contains.

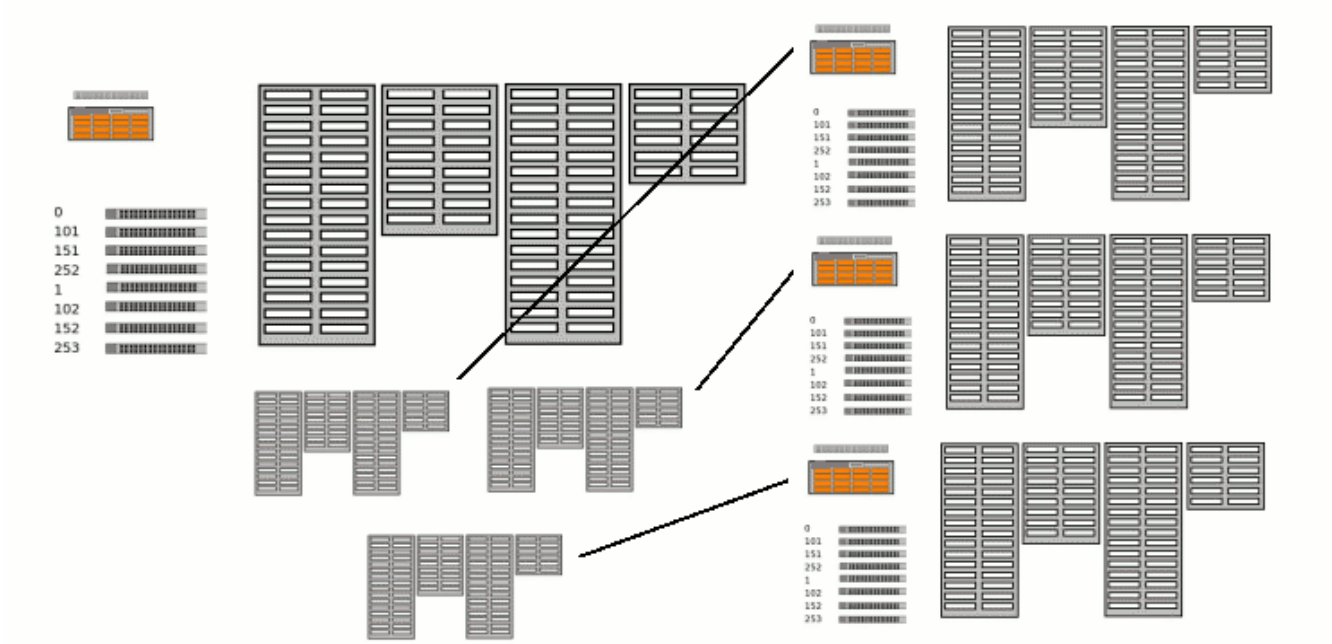
Segment volumes also are more robust if a drive should fail in an array. With a segmented architecture, only segments stored on the affected array or drive are lost, the remaining segments can not only be recovered, but the system can continue to operate and capture data to disk.



Segmented volumes allow arrays and storage partitions of different sizes to be combined and addressed as self contained discrete caches of disk segments. Segmented volumes perform logical striping by allocating segments into logical streams of captured packets "round robin". 64K and segment striping perform similar striping methods but at the physical layer. Segmented volumes perform the striping at the logical layer (which means it is visible to the file system directly). This model provides very high levels of disk throughput and performance, with the added ability to dynamically add or remove arrays, nodes, or partitions to a running system. This model lends itself well to clustering over fiber channel fabrics.

Segmented volumes provide high performance with heavy network loads since they still employ a logical variant of segment striping. Segments are loosely allocated “round robin” between partitions or arrays and the forensic file system attempts to load balance segment writes across arrays by striping at the logical level of segment number instead of at the physical disk layer as is performed in 64K and segment striping models. RAID 0 is the best performance option for Segmented Volumes for performance, but RAID 5 combined with Segmented volumes provides the highest level of fault tolerance with the Forensic File System.

Another advantage of segmented volumes is they support distributed clustering over fiber channel fabrics. Since each drive array or logical partition set is managed as a discrete cache of individual segments, these partition sets are self contained. Segmented volumes support allocation zoning, which allows clustered nodes to be managed as time domains, allowing specific windows of capture to reside within zoned areas of a cluster based upon pre-configured time windows.

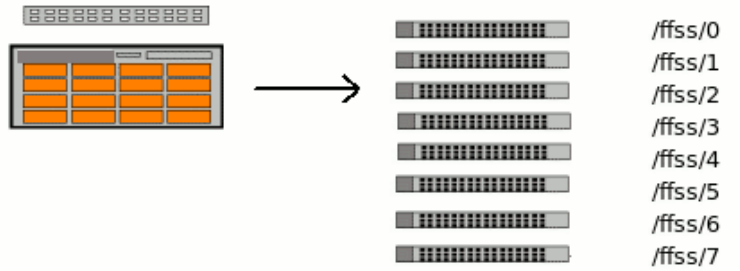


Segments volumes can be combined and addressed by a single capture appliance over a fiber channel storage fabric and with the a Clustered dynamic segment manger installed in each appliance node, allow remote readers all access to data while capture is in process. This architecture also allows specific arrays and appliances to be zoned and managed as remote disk caches for supporting large storage and archival needs of captured data.

Clustered dynamic segment archive support is provided for configurations clustered over fiber channel networks. This option creates and maintains a distributed library table of all segments which exist within the cluster in master/shadow tables maintained on all cluster nodes which describes the total time based segments contained within a library of disk caches.

Workgroup Collector

The Workgroup Collector configuration is best supported on a virtual platform configuration, 1U platform or smaller, or workgroup collector.



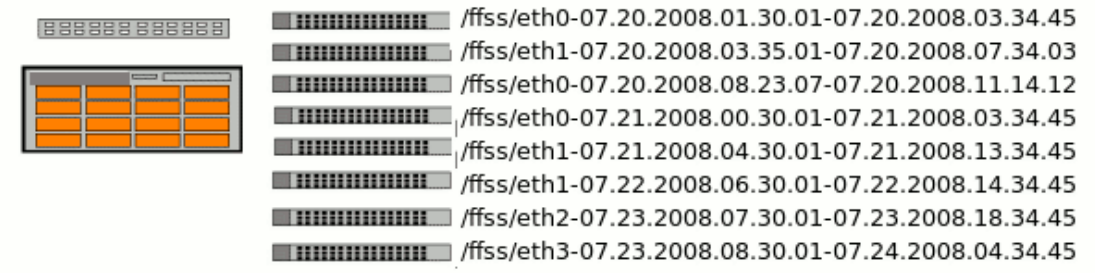
In Workgroup Collector Mode, the Forensic Filesystem can be configured to use the linux native filesystem or a remote Windows Server to store disk segments files.

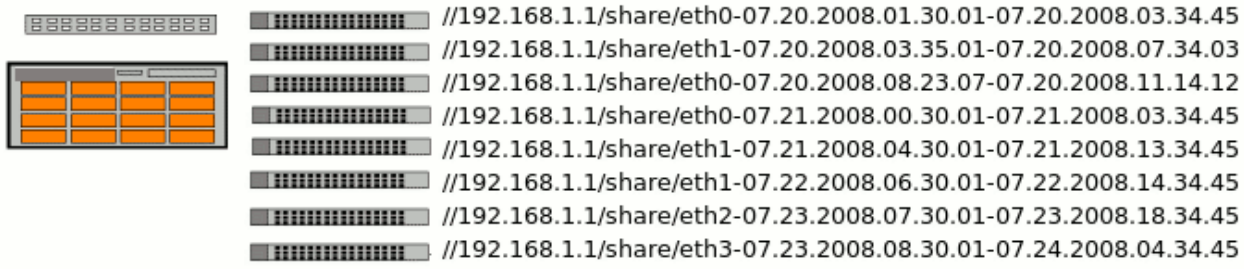
Collector mode allow allows segment files to be redirected to a remote Windows, Unix, Netware, or other local server for storage. This mode also supports continuous infinite capture by creating write once disk segment files with unique names and date and time stamps.

With this option enabled, the Forensic File system will use simple files and the local file system of the device rather than use the native storage architectures previously described. The advantage of this model is in its support for remote file system protocols such as CIFS and SMBFS (Microsoft Windows Server), NCPFS (Novell Netware), AFS, NFS, CODA, and other remote file system protocols. In this mode, the collector is configured to write segments as files by segment number into a local directory. This option can be configured to behave just like the Native Storage model with disk segments overwriting the oldest files and the size of the file window (maximum number of disk segment

files) is configurable.

This option can also be enabled to redirect the directory used for storing these files to a remote server, allowing the collector to spool captured data to a central server for consolidation of captured traffic or archival. The Collector function can be configured to run in a FIFO capture mode, creating unique disk segment files for each segment captured and writing them sequentially to a central archive server or stored locally rather than overwriting the oldest disk segments files when the file limit has been reached.



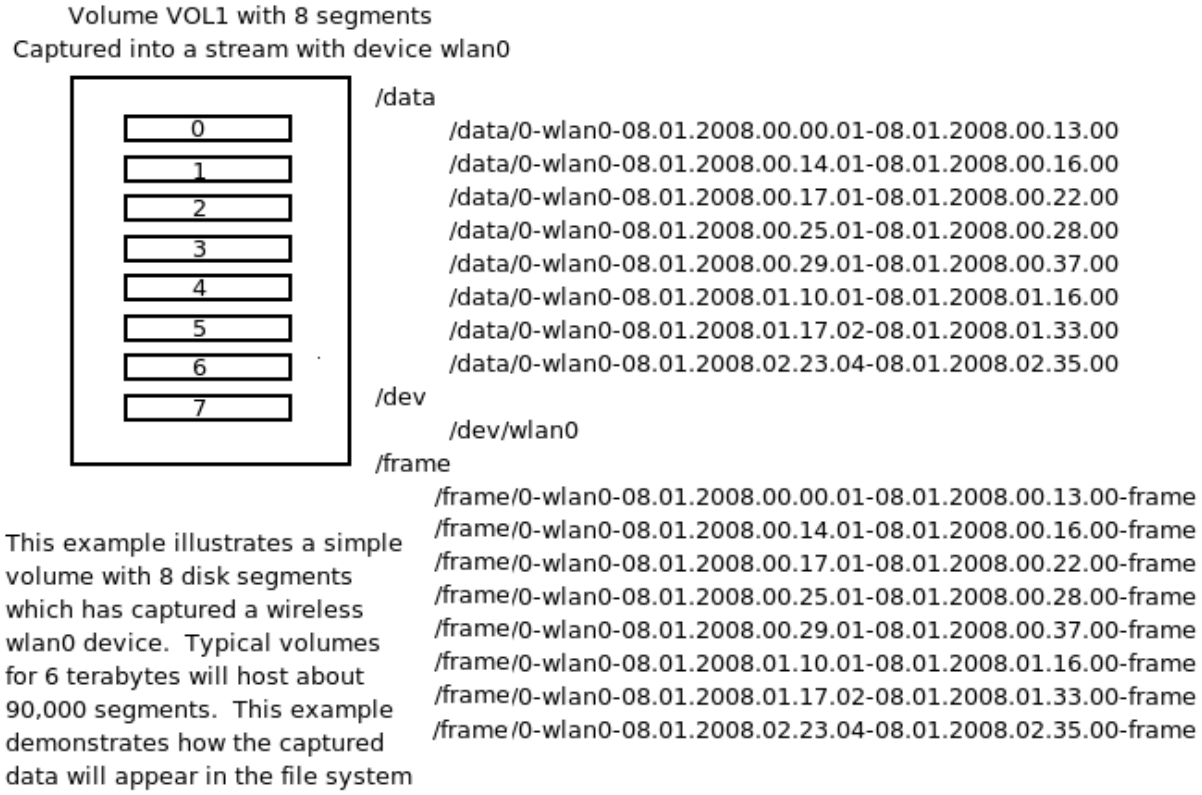


When configured in continuous mode running as a collector appliance, the forensic file system can be directed to create unique date and time indexed files, either in raw disk segment or LIBPCAP format, and redirected to local storage or forward to a remote server of a network. Capture data can aso be routed via traffic regeneration to specifc downstream capture nodes or analysis appliances.

The Workgroup collector can sustain up to 100 megabytes/second stream-to-disk on a native Unix file system without packet loss on the 1U platform.

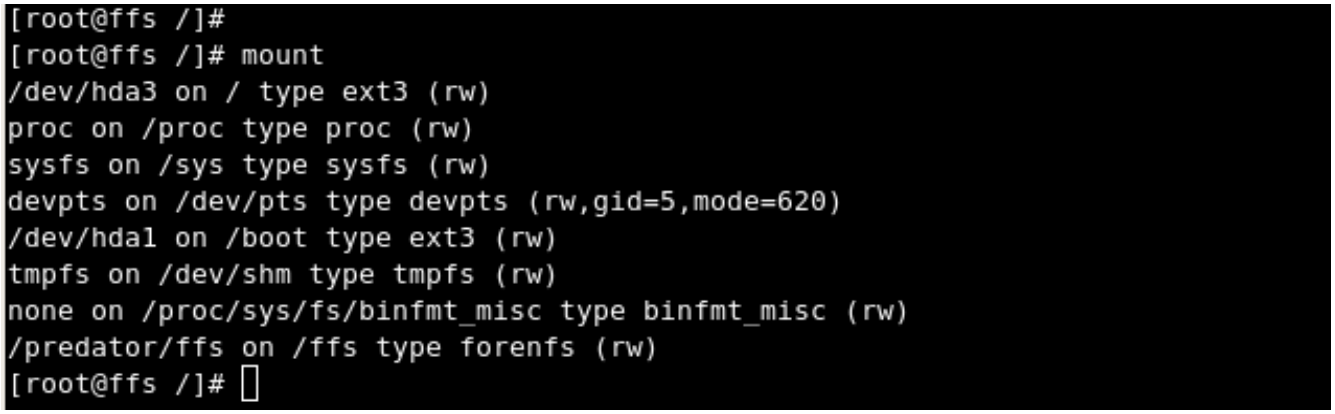
Forensic File System Organization

The Forensic File System as designed is essentially a collection of disk segments managed and linked together to form logical capture streams which are exposed as libpcap files. At the lowest level of organization, disk storage is divided into disk segments which can range in size from 16 megabytes up to 256 megabytes. The optimal size due to the mathematics dealing with 64K buffering and hardware ring buffer architecture of modern Ethernet adapters is 67 megabytes. A 67 MB disk segment typically will contained between 40,000 and 350,000 network packets, depending upon the size of the packets captured.



Disk segments are allocated and filled with captured packets then logically linked into a network packet capture stream based upon the logical interface name of the specified network adapter. These capture streams are exported

through a Unix virtual file system interface and are readable by standard applications through this interface.



The Forensic File System when mounted appears under standard Unix as “forenfs” as the file system type.

The data stored within disk segments employs an industry standard Intel EFI disk layout which is optimized for very high speed reading and writing. When the data is read through the virtual file system interface by a user space application, it is dynamically converted into LIPPCAP format (tcpdump), an industry standard packet capture file format used by thousands of commercial and open source applications.

This allows the captured network packet data to be accessible and readable by the widest possible range of network monitoring, analysis, forensics, and management applications. The exported files can be accessed and read while packet capture is active.

Individual disk segments can also be accessed and are exported and annotated by interface name with the date and time of capture through two special directories which allow scanning of the entire captured volume.

In addition, the virtual file system allows the creation of dynamic “touch” files – files which specify unique time and date ranges and which can then be

read by applications which require the ability to create data extracts for specific time periods.

Above this layer, collections of disk segments which have been grouped into streams are contained within logical volumes. By default, two system volumes are present, an internal SYS volume and VOL1. Volume Names can be automatically created by the File System during volume creation.

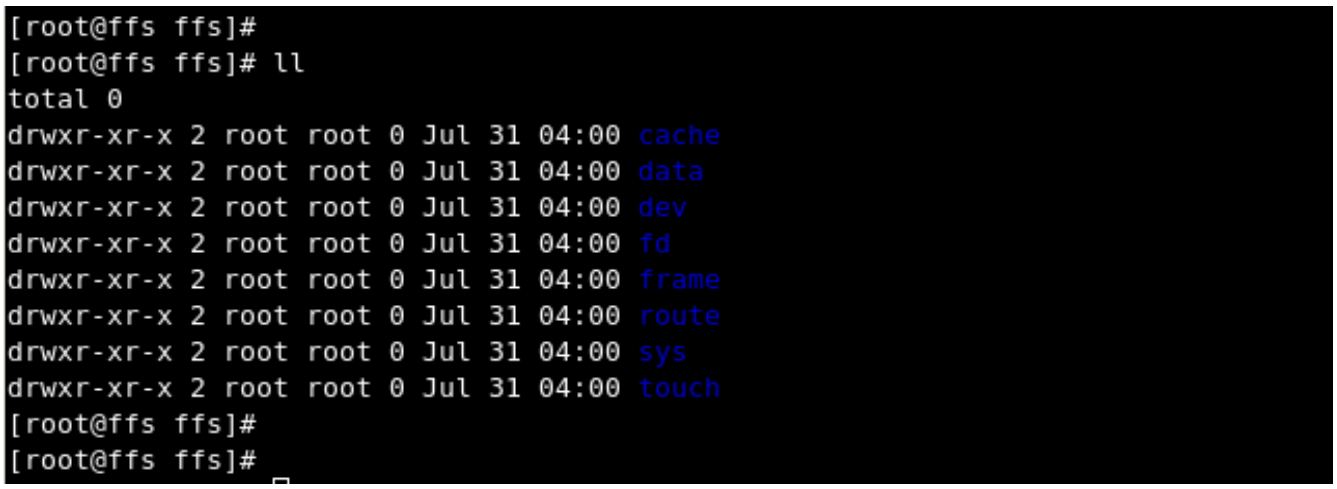
The SYS volume is a logical file system used for cache mirroring and collector operations. The SYS volume is internal and not typically accessible by user applications. The default VOL1 volume is the first storage volume created by the Forensic File system when using native storage methods and is the default volume used for stream-to-disk capture.

The Forensic File system can create and host hundreds or even thousands of distinct volumes. For most applications, a single volume suffices. Each Forensic File system volume can address up to 9,007 Terabytes (9.007 Petabytes) of storage.

The Virtual file system also exports a special system directory for monitoring system performance and statistics, and contains dynamically generated HTML and text files which are updated and recreated every second allowing remote access to file system performance and usage statistics.

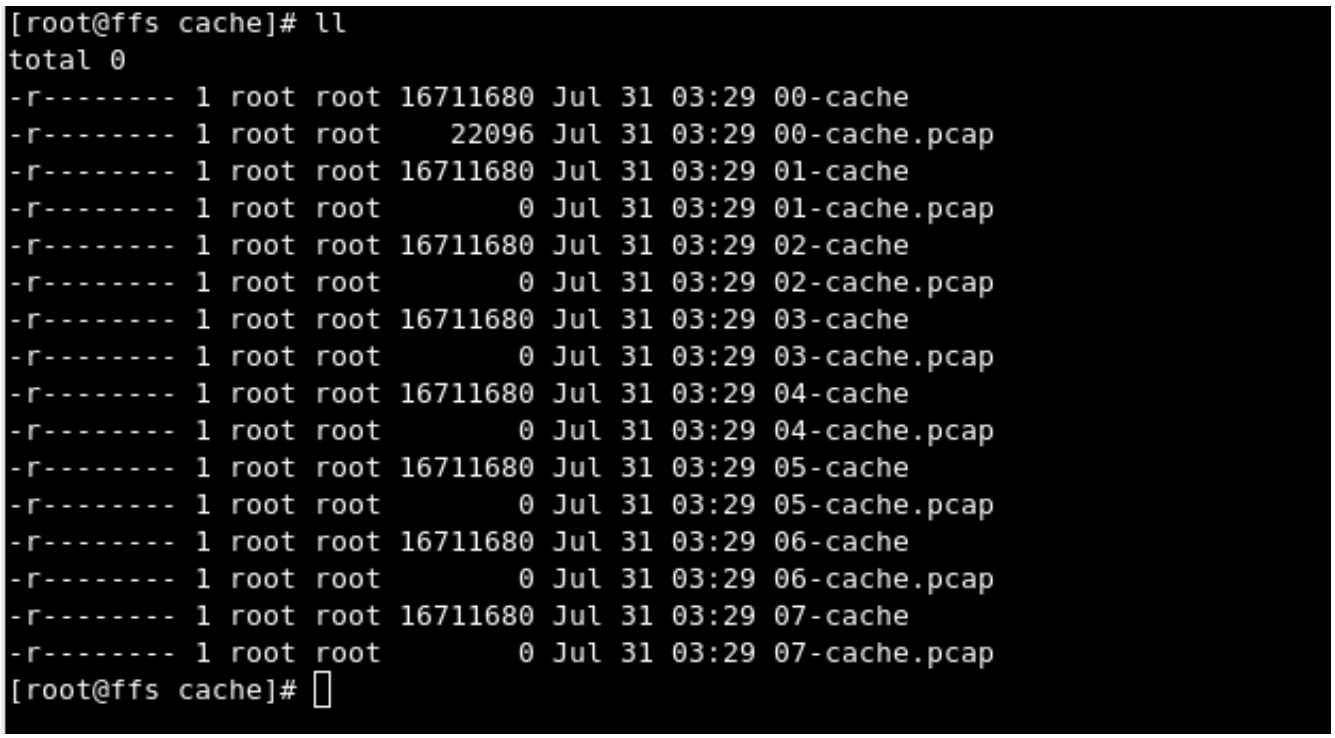
This directory contains special files used to configure the platform in collector mode for the platform, and is used to specify such settings as local or remote directory settings, data type, and cache mirroring settings.

Each Forensic File system Volume can be mounted to a specified mount point and exports the following directory layout:



Basic Directory layout for a mounted instance of the Forensic File System

Cache Directory - The cache directory contains files which are mapped to the in memory disk segment cache blocks. These are volatile files (in-memory) which can be access through the /cache directory. There is a file alias for each raw cache segment with a .pcap file extension. These files are dynamically generated LIBPCAP formatted files which can be read by forensic applications.



The cache directory is exposed for primarily external mirroring software to snapshot disk segment cache blocks or for cursory review of data while still in memory. Most applications will not access captured data through this interface, but the interface is provided to support cache mirroring features for remote collector probes and access to the forensic file system disk segment volatile (in-memory) cache memory system.

As illustrated by the example below, disk segments in cache can be used to dynamically create LIBPCAP files for review of captured data.

```
-r-------- 1 root root 16711680 Jul 31 03:29 00-cache
-r-------- 1 root root    22096 Jul 31 03:29 00-cache.pcap
-r-------- 1 root root 16711680 Jul 31 03:29 01-cache
-r-------- 1 root root        0 Jul 31 03:29 01-cache.pcap
-r-------- 1 root root 16711680 Jul 31 03:29 02-cache
-r-------- 1 root root        0 Jul 31 03:29 02-cache.pcap
-r-------- 1 root root 16711680 Jul 31 03:29 03-cache
-r-------- 1 root root        0 Jul 31 03:29 03-cache.pcap
-r-------- 1 root root 16711680 Jul 31 03:29 04-cache
-r-------- 1 root root        0 Jul 31 03:29 04-cache.pcap
-r-------- 1 root root 16711680 Jul 31 03:29 05-cache
-r-------- 1 root root        0 Jul 31 03:29 05-cache.pcap
-r-------- 1 root root 16711680 Jul 31 03:29 06-cache
-r-------- 1 root root        0 Jul 31 03:29 06-cache.pcap
-r-------- 1 root root 16711680 Jul 31 03:29 07-cache
-r-------- 1 root root        0 Jul 31 03:29 07-cache.pcap
[root@ffs cache]# tcpdump -n -r 00-cache.pcap | more
reading from file 00-cache.pcap, link-type EN10MB (Ethernet)
03:38:36.000213 IP6 :: > ff02::16: HBH ICMP6, multicast listener report v2, 1 gr
oup record(s), length 28
03:38:36.000412 IP 192.168.10.94.50155 > 69.2.248.211.https: P 1244312400:124431
2441(41) ack 3441084892 win 4369
03:38:36.000542 IP 69.2.248.211.https > 192.168.10.94.50155: . ack 41 win 64469
03:38:36.000762 IP6 :: > ff02::1:ff93:fcd6: ICMP6, neighbor solicitation, who ha
```

Data and Frame Directories – Both the /data and /frame directories of a mounted forensic file system volume contain individual disk segments which are either cached or written to disk. When an entry exists in either of these directories, the disk segment with its packet payload is cached and is also stored on disk.

The /frame directory is simply an alias of the contents of the /data directory. The purpose of having two directories is to provide applications which required frame sliced data a series of LIBPCAP files which are generated with frame slicing enabled. Dynamic LIBPCAP files in the /frame directory only return the first 96 bytes as the captured network packets. Many applications

only analyze the frame headers and ignore the data payload. The frame sliced directory is provided for those applications which only need to analyze the packet headers.

The files presented by both the /data and /frame directories are exported in the following format:

<segment number>-<interface>-start_date.time.nsecs-end_data.time.nsecs

```
[root@ffs data]# ls
0-wlan0-07.31.2008.03.38.18:0CB9B501-07.31.2008.03.52.02:2A76E371
1-wlan0-07.31.2008.03.58.16:33491349-07.31.2008.03.59.13:1D1A79E9
2-wlan0-07.31.2008.04.00.23:2ADE9221-07.31.2008.04.02.02:02881C39
3-wlan0-07.31.2008.04.03.18:2A94AB69-07.31.2008.04.31.27:05535011
4-wlan0-07.31.2008.04.32.08:3259A0E9-07.31.2008.04.32.13:1855DEF1
[root@ffs data]#
[root@ffs data]#
```

The filename for each file contains the starting and ending date and time and the nanosecond offset for the time domain of packets stored within a particular disk segment. Each disk segment in a logical forensic file system volume is assigned a logical number 0-<total volume segments> and this number is pre-pended to the beginning of each file entry in the /data and /frame directories. The Forensic File System captures and internally annotates packets with a time granularity based on nanoseconds.

```
[root@ffs data]#
[root@ffs data]# ll
total 0
-r-------- 1 root root 121370 Jul 31 03:40 0-wlan0-07.31.2008.03.38.18:0CB9B501-
07.31.2008.03.40.20:1BEC1011
[root@ffs data]#
[root@ffs data]#
```

When accessing these files, it is only necessary to specify the disk segment

number and not the entire file name.

```
[root@ffs data]# ls
0-wlan0-07.31.2008.03.38.18:0CB9B501-07.31.2008.03.52.02:2A76E371
1-wlan0-07.31.2008.03.58.16:33491349-07.31.2008.03.59.13:1D1A79E9
2-wlan0-07.31.2008.04.00.23:2ADE9221-07.31.2008.04.02.02:02881C39
3-wlan0-07.31.2008.04.03.18:2A94AB69-07.31.2008.04.31.27:05535011
4-wlan0-07.31.2008.04.32.08:3259A0E9-07.31.2008.04.32.13:1855DEF1
[root@ffs data]#
[root@ffs data]#
```

In many instances, the ending data.time.nsec pair will continue to advance during active capture on a file which is currently opened, and this can make it difficult to open the file since the file name will be changing several times a second as the ending data.time.nsec tag is updated with the capture time stamp of newly arriving packets.

The Forensic file system is designed to only require the user to specify the disk segment number for any open attempts for files in either the /data or /frame directories and will allow the file to be opened if you only specify the disk segment number.

```
[root@ffs data]# ls
0-wlan0-07.31.2008.03.38.18:0CB9B501-07.31.2008.03.52.02:2A76E371
1-wlan0-07.31.2008.03.58.16:33491349-07.31.2008.03.59.13:1D1A79E9
2-wlan0-07.31.2008.04.00.23:2ADE9221-07.31.2008.04.02.02:02881C39
3-wlan0-07.31.2008.04.03.18:2A94AB69-07.31.2008.04.31.27:05535011
4-wlan0-07.31.2008.04.32.08:3259A0E9-07.31.2008.04.32.13:1855DEF1
[root@ffs data]#
[root@ffs data]#
[root@ffs data]# tcpdump -n -r 4 | more
reading from file 4, link-type EN10MB (Ethernet)
04:32:17.000844 arp who-has 192.168.10.125 tell 192.168.10.137
04:32:17.000883 IP 192.168.10.87.netbios-ns > 192.168.10.255.netbios-ns: NBT UDP
 PACKET(137): QUERY; REQUEST; BROADCAST
04:32:18.000012 IP 192.168.10.87.netbios-ns > 192.168.10.255.netbios-ns: NBT UDP
 PACKET(137): QUERY; REQUEST; BROADCAST
04:32:18.000044 arp who-has 192.168.10.96 (00:1b:77:1e:8c:42) tell 192.168.10.2
04:32:18.000836 arp reply 192.168.10.113 is-at 00:40:0d:8a:74:17
04:32:18.000934 IP 192.168.10.94.netbios-ns > 192.168.10.2.netbios-ns: NBT UDP P
ACKET(137): REFRESH(8); REQUEST; UNICAST
```

Device Directory (/dev) – The device directory exports the master file streams for all capture data for a given device. These alias file names created in this directory correspond to an active device for which data is being captured or has been previously captured to the Forensic file system.

```
[root@ffs dev]#
[root@ffs dev]#
[root@ffs dev]#
[root@ffs dev]# ll
total 0
-r-------- 1 root root       0 Jul 31 05:11 eth0
-r-------- 1 root root 4401157 Jul 31 05:11 wlan0
[root@ffs dev]#
[root@ffs dev]#
[root@ffs dev]#
[root@ffs dev]#
[root@ffs dev]#
[root@ffs dev]#
```

The dynamically created LIBPCAP files in this directory can be read and accessed by simply typing the file name of the particular device. In the example used above, a wireless network using IP version 6 is being captured by the Forensic File System and streamed to disk with a device name of "wlan0". This file can be opened with tcpdump or any other application which can read LIBPCAP formatted data.

```
[root@ffs dev]#
[root@ffs dev]# ll
total 0
-r-------- 1 root root      0 Jul 31 03:41 eth0
-r-------- 1 root root 132232 Jul 31 03:41 wlan0
[root@ffs dev]#
[root@ffs dev]#
[root@ffs dev]# tcpdump -n -r wlan0 | more
reading from file wlan0, link-type EN10MB (Ethernet)
03:38:36.000213 IP6 :: > ff02::16: HBH ICMP6, multicast listener report v2, 1 gr
oup record(s), length 28
03:38:36.000412 IP 192.168.10.94.50155 > 69.2.248.211.https: P 1244312400:124431
2441(41) ack 3441084892 win 4369
03:38:36.000542 IP 69.2.248.211.https > 192.168.10.94.50155: . ack 41 win 64469
03:38:36.000762 IP6 :: > ff02::1:ff93:fcd6: ICMP6, neighbor solicitation, who ha
s fe80::290:4bff:fe93:fcd6, length 24
```

File Descriptor (/fd) Directory – The file descriptor (/fd) directory allows visible monitoring of all active stream sessions opened on the Forensic File system volume, including sessions which use Virtual Network Interfaces which have been mapped to a Forensic File System system volume.

```
[root@ffs fd]#
[root@ffs fd]# ll
total 0
-r-------- 1 root root 0 Jul 31 05:32 1-wlan0-C0802A0
-r-------- 1 root root 0 Jul 31 05:32 2-eth0-C0802800
[root@ffs fd]# cat 1-wlan0-C0802A0
start time :  0x4891A260:3A3E8C11 (07.31.2008.05.30.20:3A3E8C11)
end time   :  0xFFFFFFFF:FFFFFFFF (02.06.2106.00.28.07:FFFFFFFF)
flags      :  2
count      :  2
hdrsize    :  24
pkthsize   :  16
bin:       :  7
x          :  0
y          :  27200
block      :  220
offset     :  21860
position   :  0
bposition  :  0
device     :  C0802E00
devicename :  wlan0
[root@ffs fd]#
```

These descriptors detail which sessions are opened on which devices and the current file pointer position within the captured stream. This information useful for tracking active virtual device routing sessions and active users of the File System.

Route Directory (/route) – The route directory contains file handles for active virtual routing (traffic regeneration) sessions which are regenerating captured network traffic to downstream nodes.

```
[root@ffs route]#
[root@ffs route]# ll
total 0
-r-------- 1 root root 364 Jul 31 10:11 1-ffs0-lo-D37E1800
-r-------- 1 root root 356 Jul 31 10:11 2-ffs1-lo-C5A7EC00
[root@ffs route]#
[root@ffs route]#
[root@ffs route]#
```

Regeneration sessions or “virtual routing sessions” allow captured network traffic to be regenerated or “re-transmitted” over a physical or virtual network segment for analysis by downstream platforms, such as intrusion detection systems, or other analysis or profiling applications.

```
[root@ffs route]# ll
total 0
-r-------- 1 root root 364 Jul 31 10:12 1-ffs0-lo-D37E1800
-r-------- 1 root root 356 Jul 31 10:12 2-ffs1-lo-C5A7EC00
[root@ffs route]# cat 1-ffs0-lo-D37E1800
start time :  0x00000000:00000000 (12.32.1969.00.00.00:00000000)
xmit time  :  0x00000000:00000000 (12.32.1969.00.00.00:00000000)
flags      :  0
pid        :  1
virtual if :  ffs0
physical if:  lo
ifindex    :  0
rate(mb/s) :  0
task       :  dccb8b70
file       :  c0cf5a00
no buffers :  0
errors     :  0
retries    :  0
sent       :  1827
bytes sent :  147983
[root@ffs route]#
```

The route directory displays active sessions along with packets sent, packet errors, and other useful statistics information more fully described in the section of this manual which discusses packet regeneration and virtual routes.

Touch Directory (/touch) – The touch directory is used to create "touch files" with the touch command. Touch files are a very useful feature for creating data extracts of the captured data store. Touch files also allow capture streams obtained from various hardware adapters to be merged together and presented as a single contiguous stream of network packets LIBPCAP formatted file for archival or analysis by network management or network forensics applications.

Touch files are alias meta pointers which point into the stored data and allow the creation of a dynamically generated LIBPCAP file from the captured network data.

Files created in the touch directory adhere to the following syntax:

if1:if2:if3:-month.day.year.hour.min.sec-month.day.year.hour.min.sec

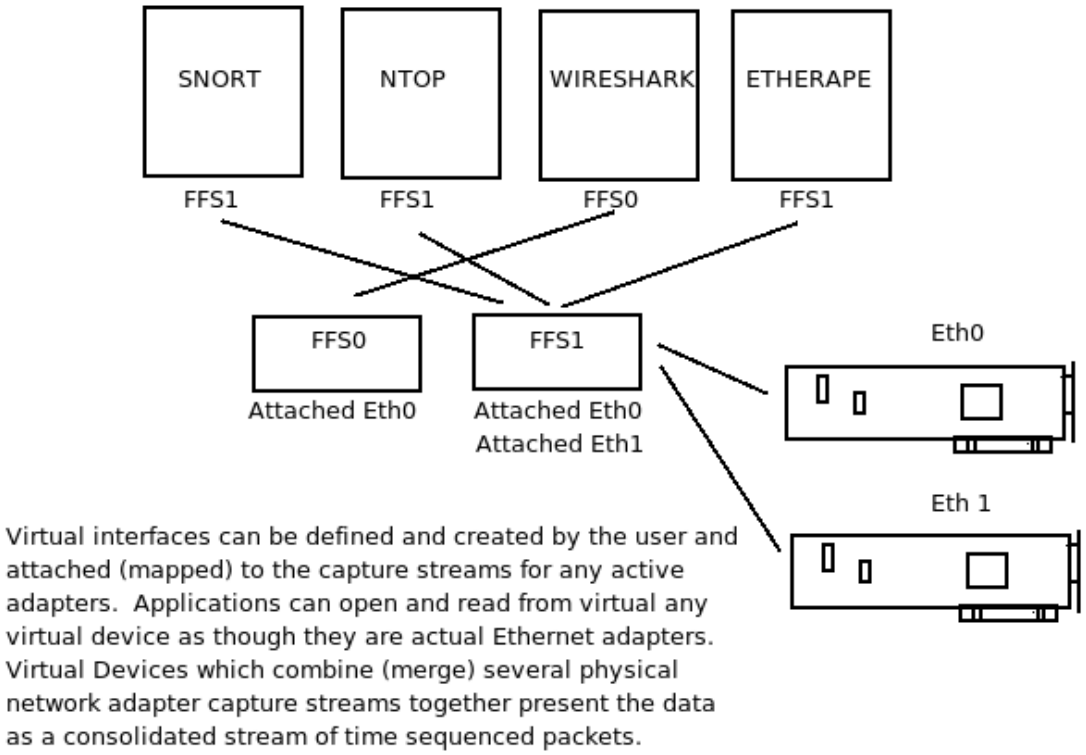
By way of example, if the platform is configured with three Ethernet devices eth0, eth1, and eth2, and you wished to create a touch file which combined all of these devices into a single logical file and extracted all data from the system for the data of August 1, 2008, you would enter the following command to create a touch file:

touch eth0:eth1:eth2-08.01.2008.00.00.00-08.02.2008.00.00.00 <enter>

The file will be created and remain persistent so long as the data range specified in the touch file still exists somewhere within the captured data streams on the platform storage arrays.

Virtual Interfaces

The Forensic File system in addition to supporting a native file system interface also exports user data through virtual Ethernet devices.

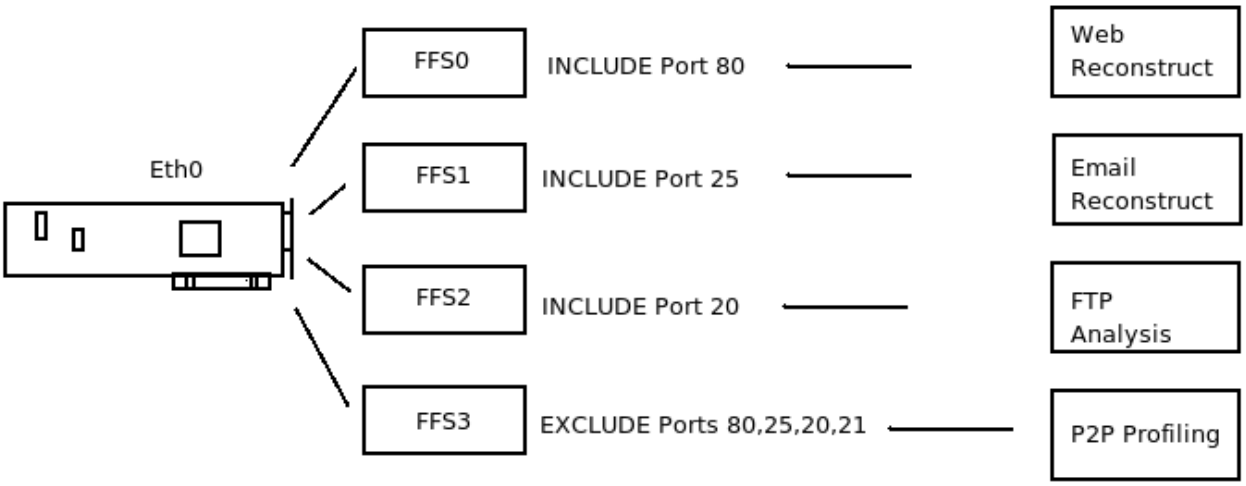


The Forensic File system allows users to create virtual Ethernet devices and assign these devices to specific capture streams stored on the platform. Since these devices are virtual and are mapped to an underlying file system, they function as lossless network taps since the virtual device is reading from previously captured data.

In most instances, the capture threshold and the pointer of a virtual device into the captured data stream operate at near real-time performance levels, with the additional benefit of virtualizing the network adapter to avoid packet loss typically observed when running real time applications on an actual hardware

interface. Another advantage to this model is in the number of concurrent applications which can be supported.

With hardware based interface adapters, many applications cannot share the device for capture without very high overhead and excessive packet copying. With virtual devices, this capability is streamlined and enhanced without the additional memory and resource requirements and poor performance associated with running multiple application instances on a single hardware based device.



Multiple virtual interfaces can be attached to a single physical device capture stream and filters can also be associated with each virtual interface. This allows only specific types of network traffic to be forwarded to applications which either analyze the specific traffic or reconstruct the traffic.

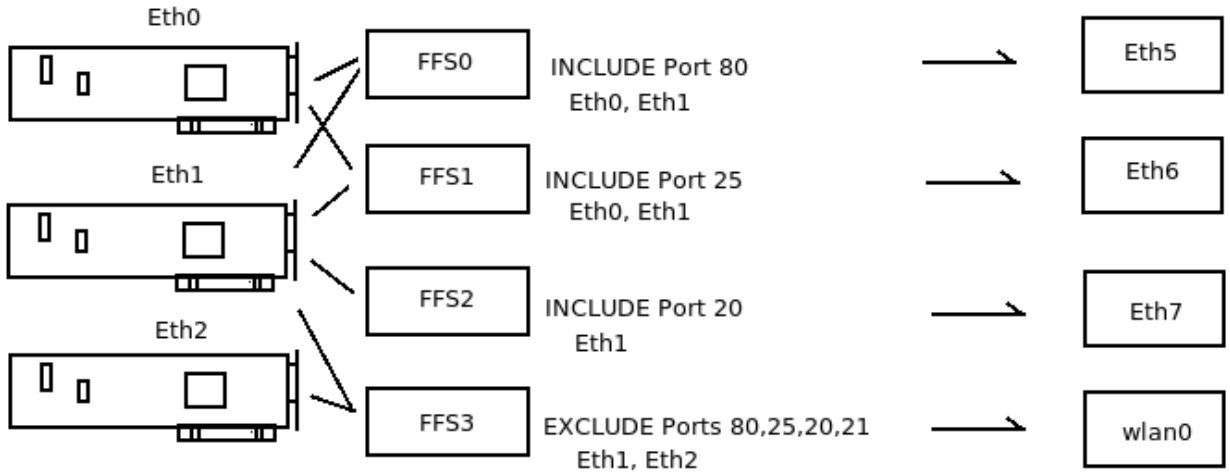
The advantage to this model is obvious. Network traffic can be categorized and forward only to those application which specifically handle targeted protocols.

Virtual devices eliminate this overhead, and allow hundreds or even thousands of application instances access to the capture stream for a specific hardware based network interface with lossless packet analysis and dramatically improved application performance, while still providing real time event analysis of the captured data.

Virtual interfaces also allow capture streams from various captured devices to be merged together and presented as a single network adapter in a time sequenced packet delivery order. Virtual interfaces can also be configured to support time based replay of captured data based upon the original timing observed when the network data was originally captured and can be associated with specific filtering criteria allowing a predefined set of virtual interfaces to split out specific network protocols for application analysis or regeneration (virtual routing) to a series of downstream devices.

Traffic Regeneration and Dynamic Routing

An important combined feature of the virtual interface capability of the Forensic File System is traffic regeneration and virtual dynamic routing capability.

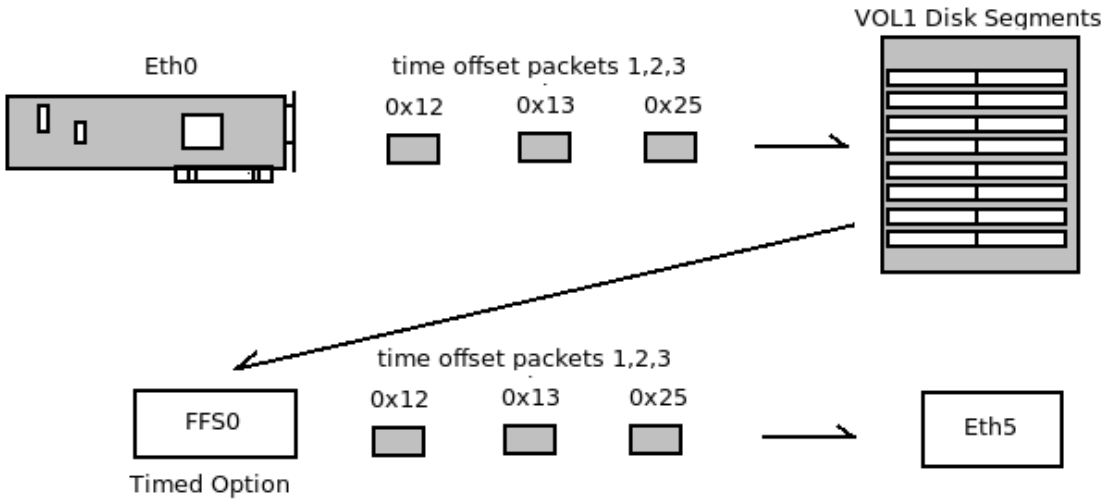


Regeneration (Virtual Routing) of captured data is an enhanced capability enabled by virtual interfaces. Virtual interfaces can be attached and associated with specific filters and the captured traffic can be regenerated onto target network segments for mirroring of the data to downstream capture appliances or for protocol analysis by intrusion detection systems (IDS) or other downstream applications or network monitoring or management systems.

Regeneration can create virtual routes from merged interfaces and can also replay traffic with the same packet gap timing contained in the original capture stream.

Traffic regeneration is the ability to read captured data from a physical device and retransmit this data over a network segment or to a local virtual network device. The advantage of this capability is that it enables protocol splitting of captured traffic for analysis by downstream devices such as third party intrusion detection systems and other network management or forensics applications. This capability also provides the ability to mirror identical captured data across an array of platforms by mirroring all data seen by a primary capture device and regenerating the data to downstream platforms or applications.

Traffic regeneration employs a predefined virtual interface which has been attached to a physical device capture stream. This virtual device is then opened and associated with a virtual route to an external network. As in the case of virtual interfaces, merged physical streams which are attached to multiple adapter capture streams and time indexed streams with filtering can be regenerated onto a target network segment.



Virtual interface can be configured to support a time indexing of packet traffic retrieved from a virtual device. When combined with traffic regeneration (virtual routing) this enables the ability to effectvely reproduce the identical timing network packets were originally received from the network during packet capture and retransmit the packets with the same timing to downstream devices or applications. This capability is extremely useful for replaying certain types of network attacks where timing and windows used in the attacks are critical for analysis purposes.

One obvious advantage to this approach is in support of an array of forensic or network management or monitoring platforms which may be constrained by bandwidth limitations. With this model, these devices will only receive the data that need to analyze without being inundated with excessive network traffic which is typically not analyzed by these devices and is wasteful of network and application bandwidth.

Intrusion detection systems or platforms which perform email reconstruction, web reconstruction, or other memory and processor intensive tasks will only wish to analyze traffic for specific ports, and not the entire network payload. Regeneration of traffic and protocol splitting reduces excessive network bandwidth utilization and allows these types of applications to experience better performance.

As was discussed in the section on file system organization, the route directory contains file handles for active virtual routing (traffic regeneration) sessions which are regenerating captured network traffic to downstream nodes.

```
[root@ffs route]#
[root@ffs route]# ll
total 0
-r-------- 1 root root 364 Jul 31 10:11 1-ffs0-lo-D37E1800
-r-------- 1 root root 356 Jul 31 10:11 2-ffs1-lo-C5A7EC00
[root@ffs route]#
[root@ffs route]#
[root@ffs route]#
```

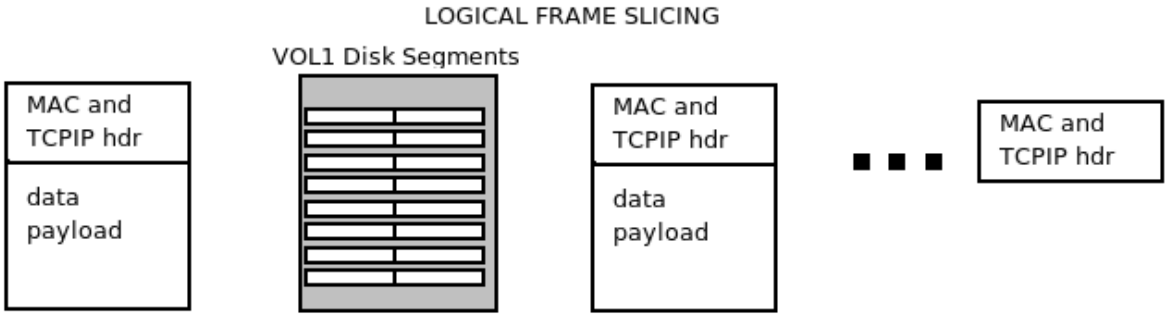
Regeneration sessions or “virtual routing sessions” allow captured network traffic to be regenerated or “re-transmitted” over a physical or virtual network segment for analysis by downstream platforms, such as intrusion detection systems, or other analysis or profiling applications.

```
[root@ffs route]# ll
total 0
-r-------- 1 root root 364 Jul 31 10:12 1-ffs0-lo-D37E1800
-r-------- 1 root root 356 Jul 31 10:12 2-ffs1-lo-C5A7EC00
[root@ffs route]# cat 1-ffs0-lo-D37E1800
start time :  0x00000000:00000000 (12.32.1969.00.00.00:00000000)
xmit time  :  0x00000000:00000000 (12.32.1969.00.00.00:00000000)
flags      :  0
pid        :  1
virtual if :  ffs0
physical if:  lo
ifindex    :  0
rate(mb/s) :  0
task       :  dccb8b70
file       :  c0cf5a00
no buffers :  0
errors     :  0
retries    :  0
sent       :  1827
bytes sent :  147983
[root@ffs route]#
```

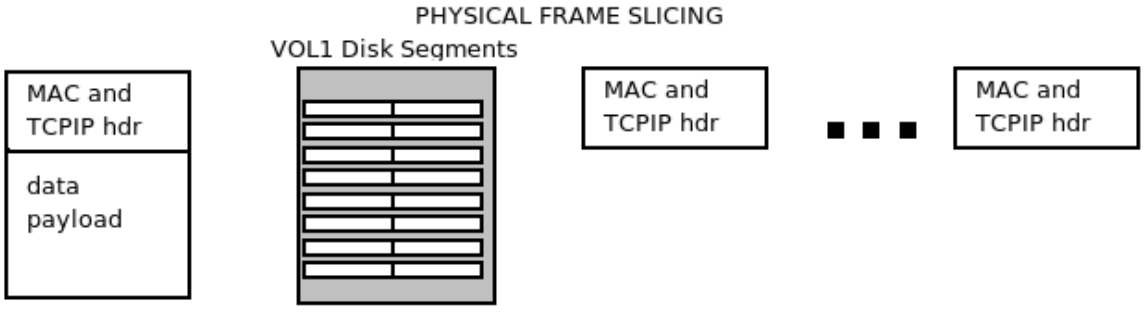
The route directory displays active sessions along with packets sent, packet errors, and other useful statistics.

Filtering and Frame Slicing

Frame slicing is the ability to specify how many bytes of captured network traffic are exposed to applications which read the data. Many network monitoring and profiling applications are unconcerned with the data payload contained within a network packet apart from the network headers, and only have a requirement to read the data contained within the TCPIP or MAC network headers of packets in order to produce useful analysis and network statistics, and do not need to read the entire packet payload. Forcing these applications to read the entire contents of all network packets is wasteful and unnecessary and can negatively impact the performance of these applications.



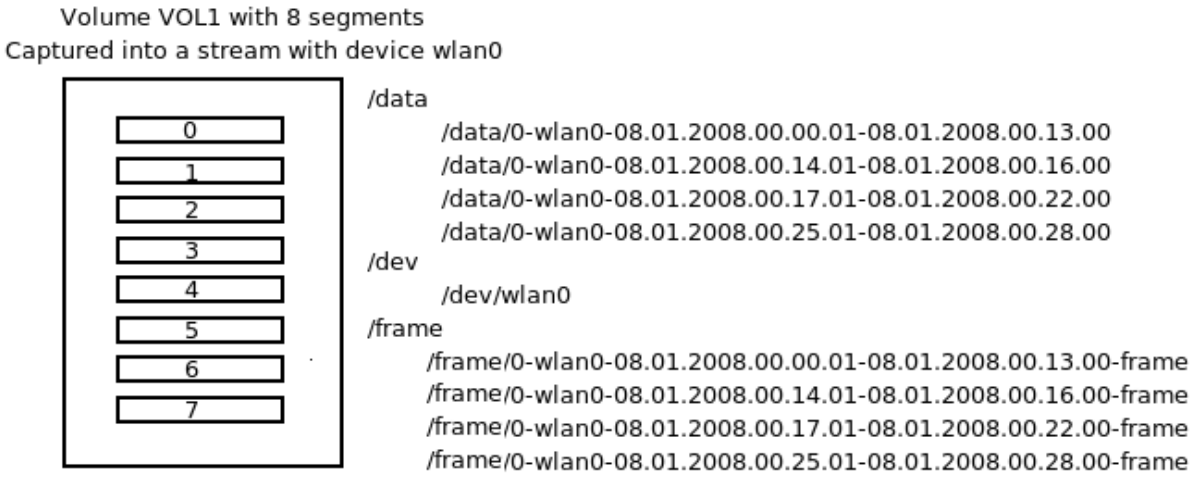
Logical Frame Slicing captures the entire packet including packet payload then exports frame sliced file data through the /frame directory n the forensic filesystem. Data payload is preserved in the volume



Physical Frame Slicing discards the data payload of each packet and prevents it from being written into the volume storage at the physical layer. Only network headers and mac headers are preserved.

Additionally, many government projects and facilities as well as many corporate facilities as a matter of policy do not allow the contents of network

packets to be captured due to confidentiality or security concerns, such as exist on computer networks which may transmit classified materials. These types of installations may have requirements to analyze network traffic without allowing permanent archival or retention of the actual data contained within any given network sessions or traffic.



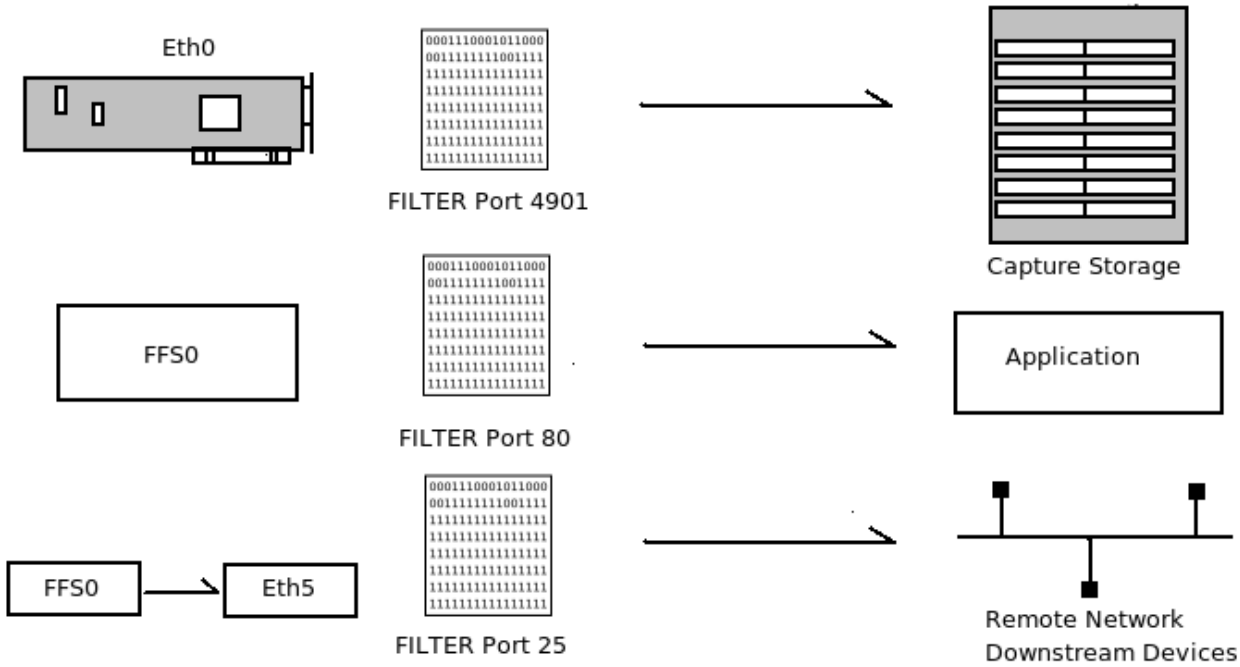
Logical Frame Slicing preserves the entire network packet payload in the capture volume for a particlular physical interface adapter, but exports the data through the virtual file system as frame sliced files. Files read from the /frame directory will only return the first 96 bytes of the network packet and discard the remaining payload. The full packet payload can still be read from the files in the /data directory. If physical frame slicing has been enabled, only the first 96 bytes of the data for any packet is available for the particular capture stream configured with physical frame slicing, since the packet data was discarded at the time of capture and is not present in the capture storage volume.

The forensic file system addresses both of these issues by allowing users to specify physical size windows for any network data captured and written to disk from the network, and can be configured to only capture network headers of packets of a predetermined packet length. This technique of clipping off the data payload portion of network packets is referred to in the networking industry as “frame slicing”.

Frame slicing within the forensic file system is supported at both the virtual file system layer for network analysis applications and at the physical capture

layer for those situations where capture of the packet payload is not allowed based upon a particular organizations security policies. The default minimum size of a frame slice is 96 bytes, which is large enough to capture all MAC and TCPIP headers, but not preserve any packet payload data for standard Ipv4 and Ipv6 traffic.

Filtering – Filtering with the Forensic File System is base upon simple include/exclude bit tables by port or with specified port ranges which are specified on the command line when a virtual routing (regeneration) session is created, when capture is initiated for a physical adapter, or when a virtual interface is attached to a physical capture stream.



Filtering bitmaps can be loaded at the physical layer and discard data from the capture storage, can be loaded at the virtual interface level and be used to split out various protocols for analysis by various application, or can be loaded at the virtual routing (regeneration) layer to specify retransmission of targeted protocols to specific downstream appliances or for data mirroring to other capture appliances.

Physical layer filtering is used to exclude or include specific port ranges for capture to the platform at the physical adapter layer. Typically, the by default will be configured to continuously capture all network traffic. However, there are certain classes of network traffic which can use excessive storage space in

the platform disk volumes and which is undesirable.

One such example would be nightly network based backups of an organizations computers with automated back software that uses the network to connect to each computer. This backup data could potentially consume hundreds of gigabytes of space on the platform volume since it would traverse the network as any other type of network traffic, and would potentially be uninteresting data which would not ever be reviewed by the organization. In such a situation, it would be desirable to create a filter on the to prevent this type of traffic from being captured along with more useful network data, such as external Internet accesses by computers within this organization or external network attacks.

Logical layer filtering at the virtual interface layer or the virtual routing layer (regeneration) was previously described and works essentially in the same manner as physical layer filtering, with the exception that the filtering operation occurs at the virtual interface level and the full packet payload will be present in the packet capture volume if the device was configured by default to capture all network traffic with full packet payload.

Forensic File System Installation and Configuration

Overview

The Forensic File System is currently supported on FreeBSD, VMWare, Linux, and Windows. Dozens of Network Forensics and Network Management applications can be run on the Forensic File system platform out of the box.

The system by default will create a pre-configured set of 16 virtual network devices FFS0-FFS15. The Forensic File System also uses a customized PARTED GNU partition editor required to create disk partitions which are larger than 2 Terabytes.

Unix does not support partitions larger than 2 Terabytes in size with the legacy FDISK utility. GPT EFI partitions (Intel Extended Partition Format) are required when using the Forensic File system on devices which are larger than 2 Terabytes.

Licensing

By default, the Forensic File system uses an encrypted licensing scheme which ties the software to various hardware components within each platform. Virtual platform versions use a similar licensing scheme. If the software is copied to a non-licensed hardware device, or if your platform requires an motherboard replacement or replacement of any of the hard drives or network adapters, a new license file will be required.

The license key is created with an algorithm based upon various hardware information contained within each platform and is tied to the hard drives within each unit. The unit will create a seed file which contains the meta data necessary to create a new license key for each system (but this file does not contain the license key itself). The seed file is created each time the system reboots and is located at /etc/ffs.seed.

If you require an additional license file, you will need to email this file or copy it to a CDROM and send it to an reseller in order to generate a license file for your system. The license file is named /etc/ffs.license and is placed in the /etc directory.

The Forensic File System WILL NOT OPERATE WITHOUT A VALID LICENSE. The system will retain the ability to read captured data should the license file be lost or deleted. But will not allow virtual interfaces, regeneration, virtual routing, or network packet capture to function unless a valid license file is present. It is highly recommended to backup the file /etc/ffs.license onto a CDROM and keep the file in a safe location for disaster recovery purposes. By default, a CDROM containing this license file is shipped with each licensed version or is provided by electronic fulfillment.

Getting Started

Partitioning of Storage

You must have first configured any hardware based RAID arrays prior to attempting to partition storage arrays, By default, the Forensic File system uses partitions which are set to fdisk(MSDOS) partition type 0x67. The Forensic File System also uses the Intel EFI GPT disk partitioning layout which supports partitions larger than 2 Terabytes in size.

The assigned GUID GPT Identifier for the Forensic File system is 83968a41-5fe6-4f9e-9111-52cf828c510a when viewed from utilities that expose EFI GUID values. Under the Unix PARTED program and the Unix Operating System, the forensic file system is identified as “forenfs” by the tools used with the file system.

To determine which drives host the forensic file system, type 'cat /etc/partitions ' to display the current partition layout for the system. System drives which host the Unix operating system will be listed at least 3 distinct device handles to represent the /boot, /, and SWAP partitions. Devices which show only a single partition or partitions of the same size are typically forensic file system disk partitions, depending on how the platform was configured. To determine which types of partitions were selected, you must open the device with either parted or fdisk.

```
[root@ffs /]#
[root@ffs /]# cat /proc/partitions
major minor  #blocks  name

   3     0   78150744 hda
   3     1     208813 hda1
   3     2    5124735 hda2
   3     3   40965750 hda3
   3     4   31848862 hda4
   8     0  156290904 sda
[root@ffs /]#
[root@ffs /]#
```

Output of cat /proc/partitions. Note that hda1, hda2, and hda3 are most likely the Unix system partitions based upon their sizes. Unix as installed on the platform will typically occupy the first three partition entries on a particular hard drive. Device sda reports it has no partitions.

Using PARTED

To create a forensic file system partition larger than 2TB on the sda device, enter 'parted /dev/sda' to invoke the GNU parted disk editor. Parted is a better tool to use if the devices are larger than 2TB. For disk drives smaller than 2TB, fdisk can be used with the legacy MSDOS disk partitioning method still used on most intel based computers.

```
[root@ffs /]#
[root@ffs /]# parted /dev/sda
GNU Parted 1.8.8
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted)
```

Create an Intel EFI GPT label on the disk. This operation is destructive and will erase all data stored on the device.

```
[root@ffs /]# parted /dev/sda
GNU Parted 1.8.8
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted)

(parted) mklabel gpt
mklabel gpt
Warning: The existing disk label on /dev/sda will be destroyed and all data on
this disk will be lost. Do you want to continue?
Yes/No? y
y
New disk label type?  [gpt]?

(parted)
```

The final step is creation of the forensic file system GPT partition with the mkpart command.

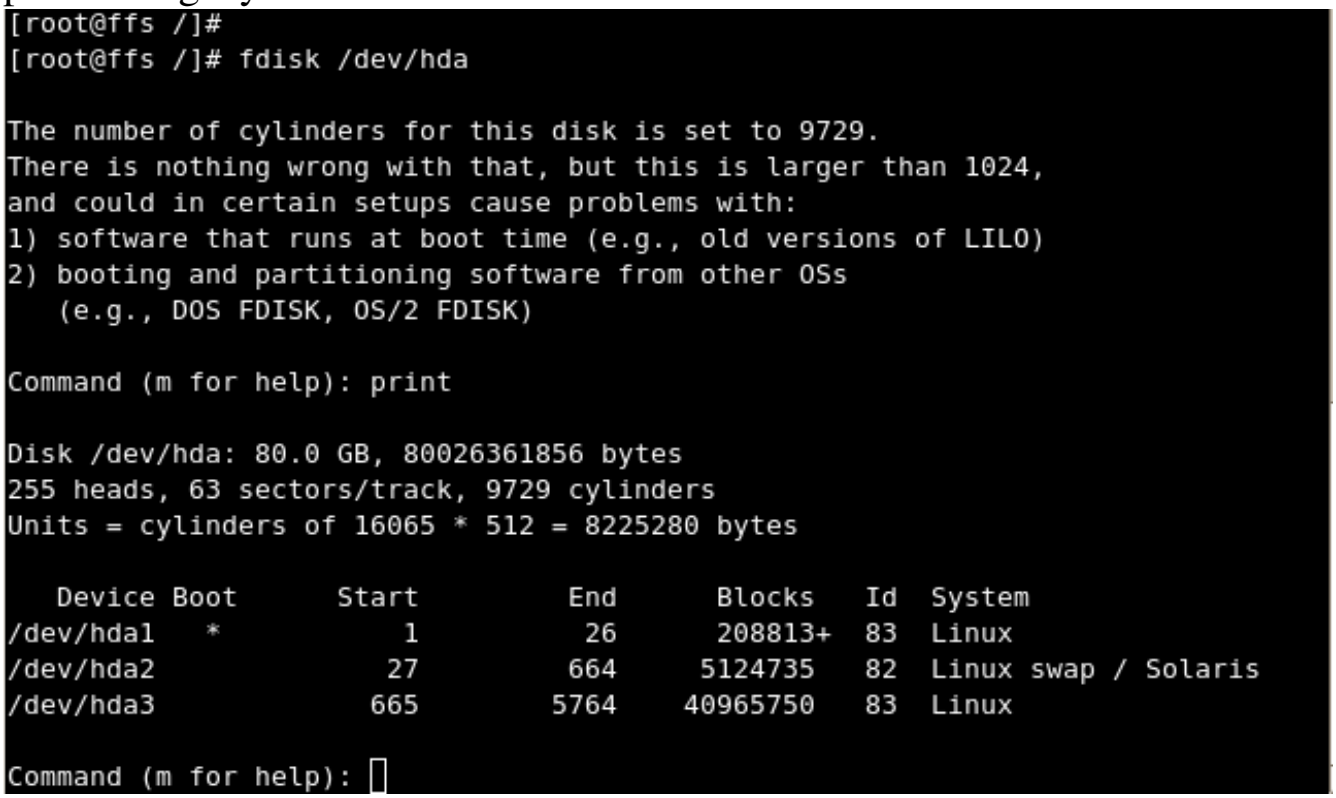
```
[root@ffs /]# parted /dev/sda
GNU Parted 1.8.8
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) mklabel gpt
mklabel gpt
Warning: The existing disk label on /dev/sda will be destroyed and all data on
this disk will be lost. Do you want to continue?
Yes/No? y
y
New disk label type?  [gpt]?

(parted) mkpart primary forenfs 0% 100%
mkpart primary forenfs 0% 100%
(parted)

(parted)
```

0% to 100% tells Parted to use the entire disk drive. The print command with parted will not always correctly identify newly created partitions as the 'forenfs' type until the drives have been formatted by the file system.

Using FDISK

Fdisk may also be used to create type 0x67 (Forensic File System) partitions on drives smaller than 2TB in size. This example creates a forensic file system

partition on device /dev/hda as partition 4 using the legacy MSDOS partitioning layout.



Please note from the panel above the three disk partitions already exists in this example and these appear to contain the base Unix operating system and the swap partitions.

Use the 'n' command to create a forensic file system partition entry and select its size when prompted to do so. Typically, you will want to claim all available free drive space for a forensic file system partition.

```
[root@ffs /]# fdisk /dev/hda

The number of cylinders for this disk is set to 9729.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Selected partition 4
First cylinder (5765-9729, default 5765):
Using default value 5765
Last cylinder or +size or +sizeM or +sizeK (5765-9729, default 9729):
Using default value 9729

Command (m for help):
```

Set the partition type to type 0x67 with the 't' command to flag this as a forensic file system partition.

```
Selected partition 4
First cylinder (5765-9729, default 5765):
Using default value 5765
Last cylinder or +size or +sizeM or +sizeK (5765-9729, default 9729):
Using default value 9729

Command (m for help): t
Partition number (1-4): 4
Hex code (type L to list codes): 67
Changed system type of partition 4 to 67 (Unknown)

Command (m for help): print

Disk /dev/hda: 80.0 GB, 80026361856 bytes
255 heads, 63 sectors/track, 9729 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1          26      208813+  83  Linux
/dev/hda2              27         664     5124735   82  Linux swap / Solaris
/dev/hda3             665        5764    40965750   83  Linux
/dev/hda4            5765        9729    31848862+  67  Unknown

Command (m for help):
```

Use the 'w' or write command to save the changes to the partition table you just made.

```
Command (m for help): p

Disk /dev/hda: 80.0 GB, 80026361856 bytes
255 heads, 63 sectors/track, 9729 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1          26      208813+  83  Linux
/dev/hda2              27         664     5124735   82  Linux swap / Solaris
/dev/hda3             665        5764    40965750   83  Linux
/dev/hda4            5765        9729    31848862+  67  Unknown

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 16: Device or resource
 busy.
The kernel still uses the old table.
The new table will be used at the next reboot.
Syncing disks.
[root@ffs ~]#
```

Formatting the Disk Arrays

After all arrays/drives have been partitioned or re-partitioned, they must now be formatted or reinitialized by the forensic file system before they can be used for data capture.

The forensic file system driver has the ability to scan the system for newly partitioned devices and add them to the existing configuration without requiring reformat of the volume provide the volume have been configured as a SEGMENTED volume during volume creation.

64K Block striping is the default configuration for most systems, and the fastest for stream to disk where performance is critical, but does not enabled dynamic volume support with additive storage capability. To enable segmented volumes, the file system must be erased and reinitialized with the segmented settings. This will result in the loss of all captured data if you choose to perform this step.

The first step is to dismount any active mount points on the device and unload the file system driver. Enter the following commands from the console logged in as the 'root' superuser. Invoke the 'mount' command to first determine where the forensic file system is mounted in the system. Dismount the file system by typing 'umount <name of mount point>. '

In this example, the mount point is /ffs identified as file system type 'forenfs'.

```
[root@ffs /]# mount
/dev/hda3 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/hda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
/predator/ffs on /ffs type forenfs (rw)
[root@ffs /]#
[root@ffs /]# umount /ffs
[root@ffs /]#
```

The next step is to unload the forensic file system driver then reload it with specific parameters directing it to reformat (erase) or repartition the platform storage. Use the 'rmmod' command to unload the file system driver. The command to mount or remount a mount point to a forensic file system volume is:

#
 # mount ffs <mount point> -t forenffs
#

The typical default mount point in an Forensic File System is /ffs.

```
[root@ffs predator]#
[root@ffs predator]# umount /ffs
[root@ffs predator]#
[root@ffs predator]# rmmod ffs
[root@ffs predator]#
[root@ffs predator]# tail /var/log/messages
Aug  6 05:19:38 ffs kernel: ffs:  adding device [wlan0] (4)
Aug  6 05:19:38 ffs kernel: ffs:  adding device [eth0] (2)
Aug  6 05:19:38 ffs kernel: ffs:  reading netdev table (2)
Aug  6 05:19:38 ffs kernel: ffs:  reading netdev map (65536)
Aug  6 05:19:38 ffs kernel: ffs:  volume VOL1 mounted
Aug  6 05:19:38 ffs kernel: ffs:  device wlan0 bound to ffs0
Aug  6 05:19:38 ffs kernel: ffs:  device eth0 bound to ffs1
Aug  6 05:20:20 ffs kernel: ffs:  dismounting volume VOL1
Aug  6 05:20:20 ffs kernel: ffs:  writing netdev table (2)
Aug  6 05:20:20 ffs kernel: ffs:  writing netdev map (65536)
[root@ffs predator]#
```

The 'tail /var/log/messages' command will dump to the screen the 'tail' or the last few lines of the system log. You should see messages similar to those displayed in the panel above which indicate the file system driver has unloaded successfully.

Device Driver Method for Partitioning the Drives

There are three file system commands which are passed to the forensic file system driver which enabled the ability to format and setup your storage. You must first tell the driver to erase all data on all partitions in the system (repair_level=3). You must next specify to the driver which volume type you wish to create with the stripe_option parameter (stripe_option=[1,2,3,4]).

Type 1 – Segmented Volume (dynamic storage)
Type 2 – 64K Block Striping (Striping – does not support dynamic storage)
Type 3 – Clustered (dynamic and distributed storage)
Type 4 - Striped Segments (Striping – does not support dynamic storage)

The driver is loaded with the 'modprobe' utility, which loads the forensic file system driver and passes the command line parameters to the driver directing it how to configure the storage.

Following are several examples of how to invoke the driver and direct it to create and manage volumes based upon different stripe options. It is useful to dump the contents of the /var/log/messages file with the 'tail' utility after you perform the operation to verify it was successfully completed.

```
[root@ffs predator]#
[root@ffs predator]# modprobe ffs repair_level=3 stripe_option=1 gather=1
[root@ffs predator]# tail /var/log/messages
Aug  6 05:27:13 ffs kernel: CacheMemory: 133693440, CacheBinSize: 16711680, Tota
lBins: 8
Aug  6 05:27:13 ffs kernel: ffs:  reinit device 03:04 (32613235200 bytes) hda4
Aug  6 05:27:15 ffs kernel: ffs:  reinit device 08:01 (160041851392 bytes) sda1
Aug  6 05:27:19 ffs kernel: [VOL1            ] [segmented] active segments: 1147
1
Aug  6 05:27:19 ffs kernel:    hda4 03:04 segments: 1938 start 0/0 (32613235200
bytes)
Aug  6 05:27:19 ffs kernel:    sda1 08:01 segments: 9533 start 1938/1938 (160041
851392 bytes)
Aug  6 05:27:19 ffs kernel:    stripe size: 1938
Aug  6 05:27:19 ffs kernel:    logical segments: 11471 logical sectors 375937031
Aug  6 05:27:19 ffs kernel:    stripe segments: 3876 stripe sectors 127008768
Aug  6 05:27:19 ffs kernel: MemTotal:   481728 kB, MemFree:    13028 kB, Buffers
:     6316 kB
[root@ffs predator]#
[root@ffs predator]#
[root@ffs predator]#
```

This example creates a segmented volume with dynamic additive storage. Note the command line parameters passed to modprobe.

```
[root@ffs predator]#
[root@ffs predator]# modprobe ffs repair_level=3 stripe_option=2 gather=1
[root@ffs predator]# tail /var/log/messages
Aug  6 05:29:04 ffs kernel: CacheMemory: 133693440, CacheBinSize: 16711680, Tota
lBins: 8
Aug  6 05:29:04 ffs kernel: ffs:  reinit device 03:04 (32613235200 bytes) hda4
Aug  6 05:29:06 ffs kernel: ffs:  reinit device 08:01 (160041851392 bytes) sda1
Aug  6 05:29:09 ffs kernel: [VOL1            ] [64K striping] active segments: 3
876
Aug  6 05:29:09 ffs kernel:    hda4 03:04 segments: 1938 start 0/0 (32613235200
bytes)
Aug  6 05:29:09 ffs kernel:    sda1 08:01 segments: 9533 start 1938/1938 (160041
851392 bytes)
Aug  6 05:29:09 ffs kernel:    stripe size: 1938
Aug  6 05:29:09 ffs kernel:    logical segments: 11471 logical sectors 375937031
Aug  6 05:29:09 ffs kernel:    stripe segments: 3876 stripe sectors 127008768
Aug  6 05:29:09 ffs kernel: MemTotal:   481728 kB, MemFree:     8092 kB, Buffers
:     6712 kB
[root@ffs predator]#
[root@ffs predator]#
```

The example above illustrates 64K striping storage setup. The example below

illustrates clustered volume setup.

```
[root@ffs predator]#
[root@ffs predator]# modprobe ffs repair_level=3 stripe_option=3 gather=1
[root@ffs predator]# tail /var/log/messages
Aug  6 05:30:11 ffs kernel: CacheMemory: 133693440, CacheBinSize: 16711680, Tota
lBins: 8
Aug  6 05:30:11 ffs kernel: ffs:  reinit device 03:04 (32613235200 bytes) hda4
Aug  6 05:30:13 ffs kernel: ffs:  reinit device 08:01 (160041851392 bytes) sda1
Aug  6 05:30:16 ffs kernel: [VOL1            ] [clustered] active segments: 3876
Aug  6 05:30:16 ffs kernel:    hda4 03:04 segments: 1938 start 0/0 (32613235200
bytes)
Aug  6 05:30:16 ffs kernel:    sda1 08:01 segments: 9533 start 1938/1938 (160041
851392 bytes)
Aug  6 05:30:16 ffs kernel:    stripe size: 1938
Aug  6 05:30:16 ffs kernel:    logical segments: 11471 logical sectors 375937031
Aug  6 05:30:16 ffs kernel:    stripe segments: 3876 stripe sectors 127008768
Aug  6 05:30:16 ffs kernel: MemTotal:   481728 kB, MemFree:     7868 kB, Buffers
:     6892 kB
[root@ffs predator]#
[root@ffs predator]#
```

The example below illustrates a striped segmented volume. Note that in this configuration setup the volume stripe size is logically the same across both volume segments in a similar manner to 64K striping except it is aligned on the disk with segment size boundaries.

```
[root@ffs predator]#
[root@ffs predator]# modprobe ffs repair_level=3 stripe_option=4 gather=1
[root@ffs predator]# tail /var/log/messages
Aug  6 05:31:22 ffs kernel: CacheMemory: 133693440, CacheBinSize: 16711680, Tota
lBins: 8
Aug  6 05:31:22 ffs kernel: ffs:  reinit device 03:04 (32613235200 bytes) hda4
Aug  6 05:31:24 ffs kernel: ffs:  reinit device 08:01 (160041851392 bytes) sda1
Aug  6 05:31:27 ffs kernel: [VOL1            ] [striped segments] active segment
s: 3876
Aug  6 05:31:27 ffs kernel:    hda4 03:04 segments: 1938 start 0/0 (32613235200
bytes)
Aug  6 05:31:27 ffs kernel:    sda1 08:01 segments: 9533 start 1938/1938 (160041
851392 bytes)
Aug  6 05:31:27 ffs kernel:    stripe size: 1938
Aug  6 05:31:27 ffs kernel:    logical segments: 11471 logical sectors 375937031
Aug  6 05:31:27 ffs kernel:    stripe segments: 3876 stripe sectors 127008768
Aug  6 05:31:27 ffs kernel: MemTotal:   481728 kB, MemFree:     6988 kB, Buffers
:     7088 kB
[root@ffs predator]#
[root@ffs predator]#
```

After reinitializing your storage, you will need to reboot the Forensic File System in order to allow it to complete the storage reinitialization. The reboot process will reload the forensic file system driver and update the storage arrays with the new configuration.

Reinitializing the platform storage does not wipe out or invalidate the unit license, since this file is stored as /etc/ffs.license in the system directory volume.

Forensic File System Command Utility - FFS

The Forensic File System can be configured to automatically capture data 7x24 and to retain any previous network capture configurations for a specified set of network interfaces. The platform configuration can be retained across reboot or power cycle events by saving the ffs utility commands used to configure the forensic file system configuration into the /etc/rc.d/rc.local startup script.

This script is configured to default settings for the Forensic File System, and can be modified using the 'vi' text editor. These commands can be easily customized.

The system loads and executes the commands in this script after all other

system initialization has been performed and completed successfully. Additional commands for controlling capture streams, creating and mapping virtual interfaces, and setting up virtual routing sessions can be entered in this file and will be activated each time the platform performs boot up.

Starting and Stopping Capture

Starting and stopping capture sessions on the platform uses the following command syntax with the ffs utility:

```
[root@ffs predator]#
[root@ffs predator]#
[root@ffs predator]# ./ffs acquire eth0
ffs:  acquire eth0 returned (0)
[root@ffs predator]# ./ffs acquire wlan0
ffs:  acquire wlan0 returned (0)
[root@ffs predator]#
[root@ffs predator]#
[root@ffs predator]#
```

This example starts capture on wlan0 and eth0

```
[root@ffs predator]#
[root@ffs predator]# ./ffs release eth0
ffs:  release eth0 returned (0)
[root@ffs predator]# ./ffs release wlan0
ffs:  release wlan0 returned (0)
[root@ffs predator]#
[root@ffs predator]#
[root@ffs predator]#
```

This example stops the capture.
Capture sessions can be directed to filter traffic at the physical level and exclude it from the capture storage and to only include those packets which meet specific port criteria. Capture sessions can also be directed to exclude

traffic from specific ports during packet capture. Excluded data is discarded and not written to the packet storage of the platform if filtering at the physical layer is implemented.

FFS Capture Session Commands

```
ffs slice <physical interface><size>
ffs acquire <physical interface><filter>
ffs release <physical interface>

example 1 (start eth0 capture of all traffic EXCEPT the backup data on port 4901)

ffs acquire eth0 --exclude=4901

example 2 (you work in a classified government lab, and only the IP headers are
           allowed to be stored -- the data must be discarded and not stored on
           the appliance on eth0)

ffs slice eth0 96 (96 bytes -- the size of an IP header)
ffs acquire eth0

example 3 (capture only email traffic on eth1 and store it to disk)

ffs acquire eth1 --include=25
```

In the above examples, the '–include=' and "--exclude=' command line parameters will direct ffs to create a capture session with include/exclude bitmap tables and subsequently filter packets based upon specific port filtering criteria contained in these bitmap tables.

Attaching or Detaching Virtual Interfaces

Creating and removing virtual interface devices on the platform uses the following command syntax with the ffs utility:

```
[root@ffs predator]#
[root@ffs predator]#
[root@ffs predator]# ./ffs create ffs0
ffs:  create ffs0 returned (0) 0
[root@ffs predator]# ./ffs create ffs1
ffs:  create ffs1 returned (0) 0
[root@ffs predator]#
[root@ffs predator]#
```

This example creates two virtual Ethernet devices, ffs0 and ffs1

```
[root@ffs predator]#
[root@ffs predator]# ./ffs delete ffs0
ffs:  delete ffs0 returned (0) 0
[root@ffs predator]# ./ffs delete ffs1
ffs:  delete ffs1 returned (0) 0
[root@ffs predator]#
[root@ffs predator]#
```

This example deletes two virtual Ethernet devices, ffs0 and ffs1.

If a virtual Ethernet device is deleted, any applications attached to the device or virtual routes regenerating traffic from this device will be signaled and closed.

Virtual interfaces can be mapped to physical adapters using the following command syntax with the ffs utility:

This example demonstrates how to create virtual interfaces, activate capture streams, then bind the capture streams for physical interfaces wlan0 and eth0 and attach them to virtual interfaces ffs0 and ffs1.

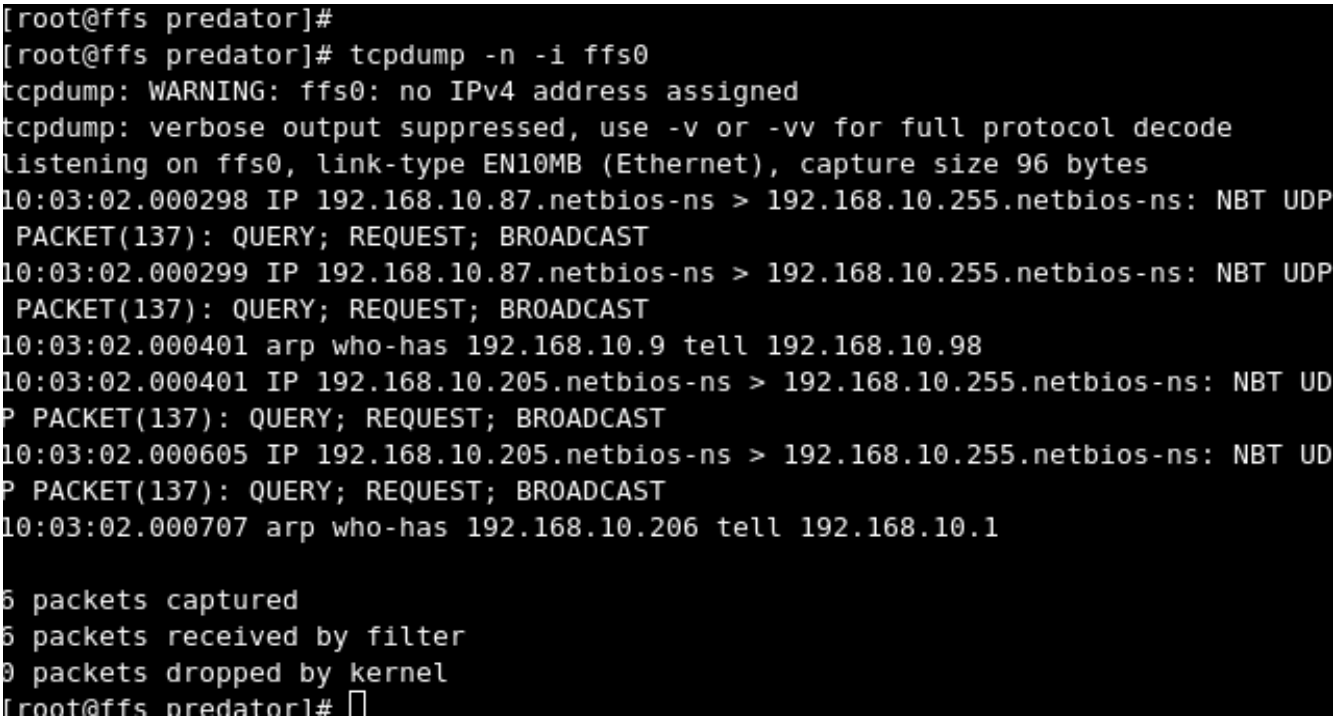
```
[root@ffs predator]#
[root@ffs predator]# ./ffs create ffs0
ffs:  create ffs0 returned (0) 0
[root@ffs predator]# ./ffs create ffs1
ffs:  create ffs1 returned (0) 0
[root@ffs predator]# ./ffs acquire wlan0
ffs:  acquire wlan0 returned (0)
[root@ffs predator]# ./ffs acquire eth0
ffs:  acquire eth0 returned (0)
[root@ffs predator]# ./ffs attach wlan0 ffs1
ffs:  attach wlan0 to ffs1 returned (0) 0
[root@ffs predator]# ./ffs attach eth0 ffs0
ffs:  attach eth0 to ffs0 returned (0) 0
[root@ffs predator]#
[root@ffs predator]#
```

This example demonstrates how to detach a virtual interface mapping from a physical interface capture stream.

```
[root@ffs predator]#
[root@ffs predator]# ./ffs detach eth0 ffs0
ffs:  detach eth0 from ffs0 returned (0) 0
[root@ffs predator]#
[root@ffs predator]# ./ffs detach wlan0 ffs1
ffs:  detach wlan0 from ffs1 returned (0) 0
[root@ffs predator]#
[root@ffs predator]#
```

A virtual interface can be read from in the same manner as any other ethernet device. After the virtual device has been attached to a physical capture stream,

it can be opened and used to read all captured data received from a physical adapter real time. When a virtual device reaches an end of the capture stream, it waits until more traffic is captured from the physical device and proceeds. From the view of the application, the application appears to be capturing from a live network interface since virtual interfaces appear as Ethernet adapters to the host operating system.



Virtual interface FFS0 attached to physical adapter wlan0 capturing packets with TCPDUMP

Virtual devices are essentially routing end points with unique pointers into a capture stream of packets which are routing all packets captured from a physical adapter and stored within the platform. This model allows applications to open and read hundreds of unique instances of a physical capture stream with the same behavior as if they were attached directly to the physical adapter, with the added benefits of improved performance, copyless and lossless packet capture and analysis, and real time performance levels.

Virtual Interface can be directed to filter traffic at the virtual interface level to

only include those packets which meet specific port criteria. Virtual Interface adapters can also be directed to exclude traffic from specific ports during reading of packets from the virtual device. Excluded data is discarded and not returned to the reader of the virtual interface if filtering at the physical layer is implemented.

FFS Virtual Interface Commands

```
ffs create <virtual device name><options><filter>
ffs delete <virtual device name>
ffs attach <physical device><virtual device><filter>
ffs detach <physical device><virtual device>
ffs attach ALL <physical device><options><filter>

example 1 (create a virtual device which merges traffic from all physical adapters)

ffs create ffs0
ffs attach ALL ffs0

example 2 (create virtual device and receive all eth0 and eth1 traffic for port 80)

ffs create ffs1
ffs attach eth0 ffs1 --include=80
ffs attach eth1 ffs1 --include=80
```

In the above examples, the '–include=' and "--exclude=' command line parameters will direct ffs to create a virtual device and create and bind include/exclude bitmap tables and subsequently filter packets based upon specific port filtering criteria contained in these bitmap tables each time a packet is read from the virtual device.

Creating or Removing Virtual Regeneration Routes

Starting and stopping virtual routing sessions on the platform uses the following command syntax with the ffs utility:

This example creates a virtual route from virtual device ffs0 (attached to wlan0) and forwards all captured traffic from ffs0 to the network attached to eth0. This example also creates a virtual route from ffs1 (mapped to wlan0) and forwards the traffic to the local loopback device (lo).

```
[root@ffs predator]#
[root@ffs predator]# ./ffs route ffs0 eth0
ffs:  route ffs0 -> eth0 pid: 0 ret (4850) 0
[root@ffs predator]#
[root@ffs predator]# ./ffs route ffs1 lo
ffs:  route ffs1 -> lo pid: 0 ret (4852) 0
[root@ffs predator]#
[root@ffs predator]#
```

This example clears the virtual routes created in the previous example.

```
[root@ffs predator]#
[root@ffs predator]# ./ffs uroute ffs1 lo
ffs:  clear route ffs1 -> lo ret (0) 0
[root@ffs predator]# ./ffs uroute ffs0 eth0
ffs:  clear route ffs0 -> eth0 ret (0) 0
[root@ffs predator]#
[root@ffs predator]#
```

A value in megabits per second can be added as a final argument to the the 'ffs route' command to limit the traffic regeneration rate for each virtual route. The value is specified in megabits per second. When this value is added as the final command when creating a virtual route, the interface will monitor the packets per second rate and only regenerate traffic at this rate.

FFS Virtual Route Command Examples

ffs route <virtual interface> <physical interface> <megabits/second rate>

megabit/second rate can be 1 to 10000 or greater, 0 means full line rate:

```
      1  =  1 megabit/second   (0.125 megabytes/second)
     10  =  10 mb Ethernet      (1.25 megabytes/second)
    100  =  100 mb Ethernet    (12.5 megabytes/second)
   1000  =  1 Gb Ethernet       (125.0 megabytes/second
  10000  =  10 Gb Ethernet     (1.25 gigabytes/second)
```

i.e.

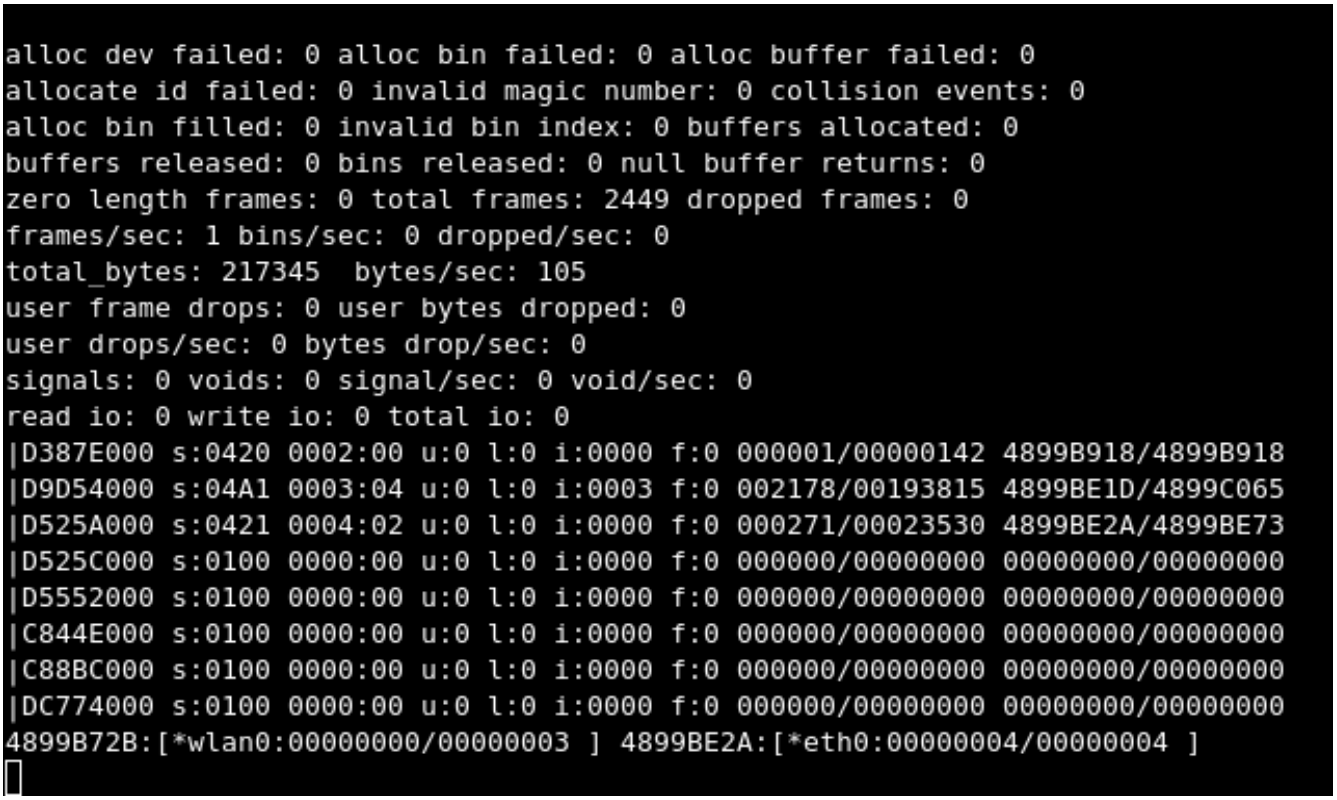
```
ffs route ffs0 eth1 100 (route packets from ffs0 out eth1 at 100 mb/s rate)
ffs route ffs1 eth1 100 --include=80-90,25,22 (route as before but only include
                                                traffic for these ports)
ffs route ffs2 eth1 1000 --exclude=25 (route at 1Gb/s all traffic except
                                       port 25)
```

Virtual routing sessions can also be directed to filter regenerated traffic to include only those packets which meet specific port criteria or can be directed to exclude traffic from specific ports during regeneration.

In the above examples, the '–include=' and "--exclude=' command line parameters will direct ffs to create a virtual route and create include/exclude bitmap tables and subsequently filter packets based upon specific port filtering criteria contained in these bitmap tables. With this example, these filters are only bound to the virtual routing process and are released if the virtual routing session is closed, but will filter this unique set of filtering criteria for a specified virtual routing session .

FFSMON Utility

A very simple text based statistics monitoring program is provided which will monitor and report various statistics in the system including packets per second, dropped packets, and bytes per second written to disk can received from the network. The command line switches and options are more fully described in the command reference at Appendix A.



This is an example of FFSMON output produced from the command ./ffmon -s -c -d (-s statstistics -c cache -d device information).

APPENDIX A.

Utilities and Command Options

File System driver and Storage Initialization

/sbin/modprobe ffs [options]

where options are:

cache_mode = [1-8x16MB(laptop/collector), 2-4x67MB(1U), 3-12x67MB(1U), 4-28x67MB(3U)] – size of cache memory used by the Forensic File System.

logical_size = cache logical address space size (collector only) - number of files created and managed in recycling mode for a file based collector

lru_interval = lru callback interval (default is 15 sec) – aging interval to write cached disk segments to disk when they have become partially filled but not completely full.

repair_level = repair level 0-none,1-repair,2-reinit,3-erase – options for repairing, erasing, or reinitializing storage. Reinit mode(2) destroys and rebuilds the storage tables but repairs and/or preserves captured data and disk segments, erase(3) destroys all captured data. Repair (1) will attempt to repair the file system.

stripe_option = 1-segment,2-64K stripe,3-cluster,4-striped-segments - this option must be used with repair_level=3 in order to erase then reinitialize the storage arrays.

collector = 0-File System,1-collector,2-continuous – configures the unit as a collector or continuous collector. If mode 2 is selected (continuous) the collector will write until all disk space is filled with unique files annotated by interface, date, and time.

collector_output_mode = 0-raw,1-pcap,2-regeneration – specify file output type when running as a collector. Raw mode writes disk segments in native mode. Mode 2 creates LIBPCAP formatted files. Mode 3 will regenerate the captured traffic to a pre-configed downstream platform.

check_level = 0-system default,1-always check – force auto-repair of storage even if no errors are detected. By default, the file system will not attempt to repair itself unless it detects it was improperly

shutdown. This flag will force the file system to run a mode (1) repair_level check.

enable_snapshot = snapshot capture 0-disable,1-enable – when the storage fills, stop capturing and do not reuse or recycle the storage, and enter a read-only mode of operation.

gather = auto-create volumes 0-nogather,1-gather,2-dynamic = auto-detect new storage partitions and combine them into new volumes. If mode (2) is selected, gather the newly detected storage into the existing default volume dynamically, even if the system is active and capturing.

ramdisk_enabled = disk-paged ramdisk support 0-off,1-on – enable the ability to use the backing store (virtual memory disk file) to behave as a ramdisk and allow temporary files to read and write to any mounted ffs mount points.

unloading the file system driver:

/sbin/rmmod ffs

mounting the file system:

mount ffs /ffs -t forenfs

dismounting the file systems:

umount /ffs

invoking GNU parted to create or change partitions:

parted <device> (i.e. /dev/dsa, /dev/sdb, /dev/sdc, /dev/sdd, etc.)

File System Utilities and Commands

ffs – Forensic File system command interface

ffs acquire <physical device> <filter> (i.e. eth0, eth1, etc.) - enable capture of hardware device.<filter> is specified as –include=port,port,port-range or --exclude=port,port,port-range.

ffs release <physical device> - stop capture of hardware device.

ffs create <virtual device><options><filter> – create virtual device (i.e. ffs0, ffs1, etc.) options = “time”

for time based virtual interfaces. <filter> is specified as –include=port,port,port-range or --exclude=port,port,port-range.

ffs delete <virtual device> - delete a virtual device and remove all processes, routes, and capture streams associated with it.

ffs attach <virtual device> <physical device><filter> - map a virtual device onto a physical device capture stream. Multiple physical capture streams can be attached to a single virtual device. <filter> is specified as –include=port,port,port-range or --exclude=port,port,port-range.

ffs detach <virtual device><physical device> - unmap a virtual device from a physical device.

ffs route <virtual device><physical device> <rate><filter>- create a virtual routing process and forward all network traffic received for a virtual device out of a physical device onto a live network a virtual network. Multiple virtual devices can have virtual routes to a single physical device. A rate can be specified in increments of megabits per second (1=1 megabit/second, 10=10 megabit/second, 100=100 megabit/second, etc). <filter> is specified as –incude=port,port,port-range or --exclude=port,port,port-range.

ffs uroute <virtual device><physical device>[option - id]- remove a virtual route by id or by name from a physical device.

ffs mount <volume name> - mount an FFS volume (default volume name is VOL1).

ffs umount <volume name> - dismount an FFS volume

ffs scan – scan the system for newly added storage and combine the storage dynamically with the default FFS VOL1 volume.

ffs slice <physical device><size><filter> - set a defined frame slice size for a physical capture device. Any data received greater than this size will be discarded from the storage.

ffsmon – display Forensic File system statistics

ffsmon [options] = -acdlsfr

-a – show ALL information
-c – show active disk segment cache information

-d – show device information
-l – show disk segment cache lru
-s – show live statistics and counters
-f – show free disk segment caches