The OSU Flow-tools Package and Cisco NetFlow Logs

Mark Fullmer – OARnet
Steve Romig – The Ohio State University

ABSTRACT

Many Cisco routers and switches support NetFlow services which provides a detailed source of data about network traffic. The Office of Information Technology Enterprise Networking Services group (OIT/ENS) at The Ohio State University (OSU) has written a suite of tools called flow-tools to record, filter, print and analyze flow logs derived from exports of NetFlow accounting records. We use the flow logs for general network planning, performance monitoring, usage based billing, and many security related tasks including incident response and intrusion detection. This paper describes what the flow logs contain, the tools we have written to store and process these logs, and discusses how we have used the logs and the tools to perform network management and security functions at OSU. We also discuss some related projects and our future plans at the end of the paper.

NetFlow Accounting Records

We should start with a more complete description of what the flows are. Quoting from Cisco:

> A network flow is defined as a unidirectional sequence of packets between given source and destination endpoints. Network flows are highly granular; flow endpoints are identified both by IP address as well as by transport layer application port numbers. NetFlow also utilizes the IP Protocol type, Type of Service (ToS) and the input interface identifier to uniquely identify flows [3].

A NetFlow record is created when traffic is first seen by a Cisco router or switch that is configured for NetFlow services. Flows are identified uniquely by characteristics of the traffic that they represent, including the source and destination Internet Protocol (IP) address, IP type, source and destination Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) ports, type of service and a few other items. NetFlow records end and are sent to the logging host on at least the following conditions:

- For flows representing TCP traffic, when the connection is done (after a RST or FIN is seen)
- When no traffic for the flow has been seen in 15 seconds.
- 30 minutes after the start of the flow. This causes long lasting traffic patterns to show up sooner than they might otherwise in the log.
- When the flow table fills.

Each NetFlow record contains data about the packets that are represented in that flow in addition to the unique identifiers listed above. These data include the start and end times for the flow, the number of packets and octets in the flow, the source and destination Autonomous System (AS) numbers, the input and output interface numbers for the device where the NetFlow record was created, the source and destination net masks and, for flows of TCP traffic, a logical 'or' of all of the TCP header flags seen (except for the ACK flag). In the case of Internet Control Message Protocol (ICMP) traffic, the ICMP type and subtype are recorded in the destination port field of the NetFlow records.

For example, suppose that a SSH connection is established from a client on host 128.146.222.233 port 1234 to a server on host 131.187.253.67 port 22, and that the traffic passes through a Cisco device that has NetFlow processing enabled. We will simplify things and identify our flows here by a tuple containing the IP Protocol type, source IP address, source TCP/UDP port, destination IP and destination TCP/UDP port. The initial packet from the client to the server causes the router to create a flow entry for {TCP, 128.146.222.233, 1234, 131.187.253.67, 23}. The response from the server to the client causes the router to create a related flow {TCP, 131.187.253.67, 23, 128.146.222.233, 1234}. Data from subsequent traffic will be aggregated in these two flow records until one of the ending conditions listed above is seen, such as

```
tc4>show ip cache 131.187.253.67 255.255.255.255 flow

SrcIf     SrcIPaddress    DstIf     DstIPaddress    Pr SrcP DstP  Pkts
AT2/0.31  128.146.222.233 AT3/0.1   131.187.253.67  06 03FA 0016     4
AT3/0.1   131.187.253.67  AT2/0.31  128.146.222.233 06 0016 03FA     8
```

Figure 1: Active flows as seen on a Cisco router.

when the TCP session ends, or because there has been no traffic for 15 seconds. Active flows can be viewed in the router command line interface with the command show ip cache <IP Mask> flow. This allows you to view flows that exist on the router whose NetFlow records have not been exported yet (Figure 1).

In the simplest case for a TCP session there will be a single flow representing the traffic from the client to the server, and a single flow representing traffic from the server to the client. The TCP flags field for both flows would typically have both the SYN and FIN bits set, indicating that packets with those flags had been seen traveling in both directions.

This is not typical, however. Traffic for a single TCP connection is frequently represented by multiple flow records, due to timeouts from lulls in the conversation, the flow table filling up, or the 30 minute flow maximum lifetime. This means that one often has to string multiple flow records together to get all of the data corresponding to an entire TCP session. In these cases, the TCP flags field can be used to determine whether a flow represents data from the start, middle or end of the TCP session. Flows from the start of a session will have the SYN (but not FIN or RST) bit set, flows from the middle of the session will typically have no flag bits set, and flows from the end of the session will have the FIN or RST bits set (but not SYN).

Flows for UDP and ICMP traffic behave similarly, although it is important to note that since neither of these are connection oriented protocols flows of UDP and ICMP traffic are just collections of similar packets.

Terminology

A **NetFlow device** is a Cisco router or switch that supports NetFlow services and which is exporting NetFlow records. **NetFlow Protocol Data Units** (PDUs, also called **NetFlow records**) are the accounting records that NetFlow devices emit. We will use the term **flow records** or **flow logs** to refer to flow accounting records in the internal flow-tools format. A **NetFlow collector** is a host running flow-capture to create a flow log from NetFlow records exported from one or more NetFlow devices. Note that **flows** are unidirectional collections of similar packets. We will use the terms **connection** or **session** to refer to all of the packets associated with bidirectional communication between a client and a server. A connection (such as a connection from a telnet client to a telnet server) will consist of at least two and possibly many more unidirectional flows.

The OSU Flow Tools

The Ohio State University has written a suite of tools for collecting, filtering, printing and analyzing Cisco flows. The tools are written to work as UNIX pipelined commands, making it easy to perform data reduction without creating unnecessary intermediate files. The tools are grouped roughly as "capture tools," "general analysis tools," and "security tools" in the following discussion.

Work on flow-tools started in August of 1996 when Cisco had released an EFT image with a new feature called NetFlow switching. At this time OSU had Internet connectivity via CICNet, and a local peering with the state network, OARnet. We needed a way to determine how much of our traffic was staying local to CICNet, and indirectly the other big 10 schools, and how much was transiting CICNet to MCI. Where our traffic was terminating and originating would potentially influence future Internet bandwidth purchasing decisions. An initial release of flow-tools aided by testing and feedback from other CICNet member schools was generating statistics in early September. Since then features, fixes and documentation have been added to the tool set, primarily for OSU's internal use with incident response and traffic analysis. Cisco has improved NetFlow switching over the past few years including features such Border Gateway Protocol (BGP) AS information, aggregated flow exports, and integration with dCEF to provide what we are now using, NetFlow accounting. Much of the work OSU has done has been made publicly available in open source form.

Internal Operation

NetFlow records are exported from Cisco gear in one of several versions. To accommodate the slightly different records, the flow-tools package receives these records in native Cisco format and translates the records to a fixed size record stream format which contains a snapshot of what was available in the Cisco version 5 NetFlow export records. This conversion work is done in the programs that receive flow exports from devices (flow-capture and flow-receive), and most of the rest of the tools work with this internal flow-tools representation (flow-fanout is the exception – this is a generic UDP packet multiplexer). Most of the programs in the flow-tools suite were designed to read and write to stdin and stdout, although some have command line options that allow you to redirect I/O to specific files. The tools also support zlib compression (RFC 1950 [10]) on the fly to conserve space.

Capture Tools

Each NetFlow enabled router or switch has to be configured to export their flow records to a flow collector. Flows are exported through UDP packets sent to a designated host and port, where some sort of network service is expected to receive the data and do something useful with it. The IOS command ip flow-export destination 10.0.0.1 12345 would cause a device to export NetFlow records to the host with IP address 10.0.0.1 at UDP port 12345. Several packages are available for receiving and processing these NetFlow exports, see the related work section at the end of this paper for a brief survey.

In the case of the flow-tools package this collection service is provided through a program called **flow-capture**. Flow-capture listens on the designated UDP port and writes the received flow records to log files. To keep individual logs from growing to an unwieldy size flow-capture rotates to a new file periodically. Flow-capture was designed to facilitate the long-term archival of flow records and has a built-in mechanism to manage the log file space in the output directory, either by restricting the number of separate log files that are maintained or by restricting the total bytes allocated to log files in the output directory. The UDP port, output directory, rotation period, and log count or size limits are all configured through the command line. For instance, flow-capture -E1G -n23 -p9991 -w/var/log/flows -z6 would cause flow-capture to listen on UDP port 9991, write its logs with level 6 compression to /var/log/flows, rotate the file once per hour, and save up to one gigabyte of log files in the output directory.

Flow-capture also allows you to connect to it via TCP to receive a real-time feed of flow records, which you can receive using the **nc** (netcat [8]) program, as in nc capturehost 9991 | flow-print. This can be useful for debugging as well as for applications that require real-time access to NetFlow records such as flow-dscan.

There is also a simple program called **flow-receive** which listens for NetFlow records on a UDP port (like flow-capture), but which writes the resulting flow records to stdout rather than archiving them to files. This is useful for debugging NetFlow exports.

Flow-fanout captures NetFlow records through a UDP port (like flow-capture) and replicates them to multiple destinations, which are specified by host and UDP port on the command line. This is primarily useful for debugging purposes, though it can also be used to replicate flow records to other tools, such as **cflowd**.

Flow-mirror and **flow-rsync** are simple scripts that copy flow logs from the collection hosts to the archive hosts using FTP and rsync, respectively.

Flow-expire implements the same space management features that flow-capture does. This is useful for managing the logs on systems that have copies of the flow logs mirrored through the flow-mirror or flow-rsync scripts but which are not running flow-capture.

Our architecture for flow collecting and processing has grown from a single Sparc 5 equipped with a few Gigabytes of disk space in August of 1996 to eleven Pentium and Pentium II based flow collectors, a dedicated Pentium III file server equipped with two 250 gigabyte RAID5 arrays, and two one gigahertz Athlon processing servers. The flow collectors are typically connected directly to the routers they are

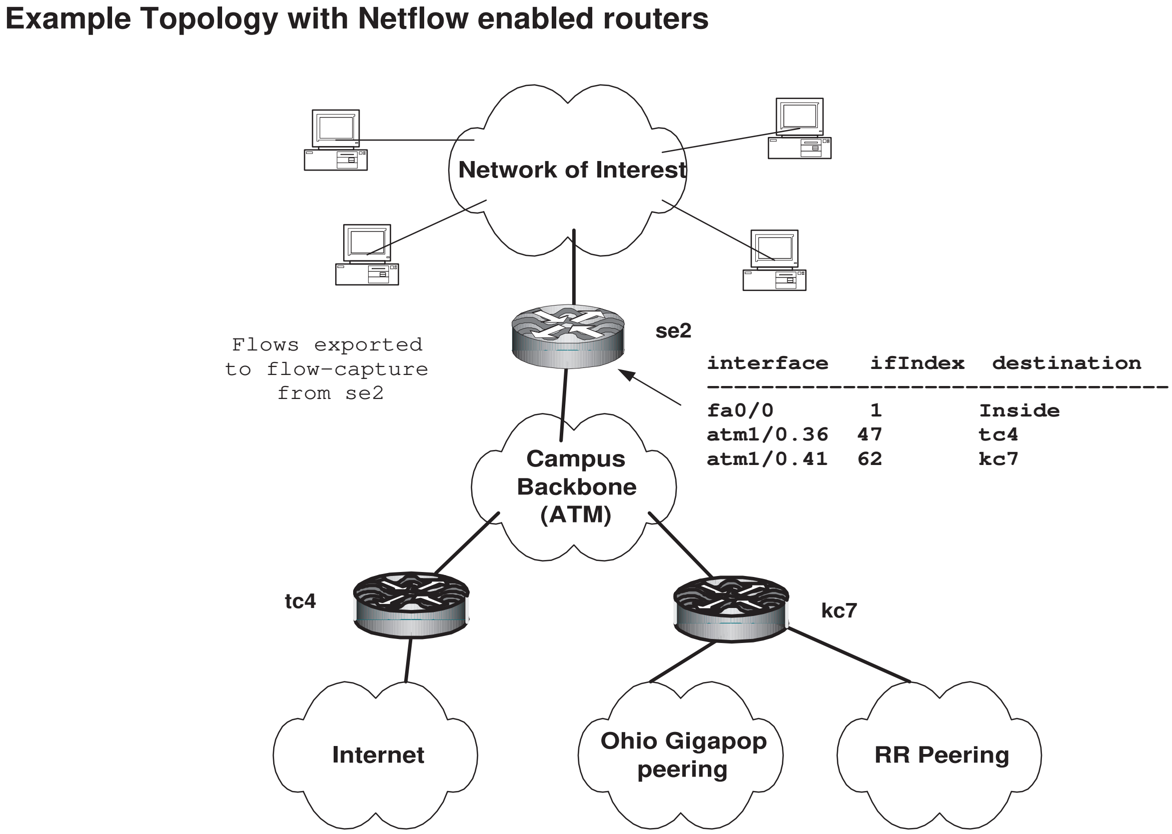

Figure 2: Simplified diagram of the OSU network.

gathering flows from and also can double as reverse terminal servers and consoles for other physically adjacent equipment. These collectors have minimal local disk space, so our primary file server polls the flow collectors once an hour with rsync for completed flow exports. We use flow-capture to receive flows and manage the disk space on the collectors, and flow-expire to manage space on the file server. Figure 2 shows a logical diagram of the OSU backbone and its connections to the Internet. Figure 3 gives an idea of how we manage flow captures.

Flow collector performance depends directly on the availability of enough free CPU cycles to compress the inbound flow exports and move the compressed stream to disk. To verify that the collector is not over subscribed, poll the kernel statistics for dropped UDP datagrams due to full socket buffers. On FreeBSD this can be done with netstat -s. Flow-capture attempts to use larger than default socket buffers to help ensure bursty flow exports are handled without unnecessary packet loss. Flow-capture is started with rtprio which gives it an unfair advantage over other running processes.

Disk IO and CPU both contribute to the performance of processing the flow data. Initially we would collect and process flows from the campus border on a single server. At that time most of our other routers were not capable of generating flow statistics due to hardware constraints or were not able to run the Cisco images that supported the NetFlow feature set. Today, NetFlow exports are archived from all the campus backbone routers to a single large disk store which is shared with two other servers dedicated to processing the data sets and generating reports. Each of the servers where the data is crunched also have a striped two disk temporary storage area to reduce NFS traffic when iterating over the same data set many times, as we typically do. A simple benchmark of using flow-cat to uncompress a days worth of flow exports from a single router and print it in ASCII leads to a processing speed of about 300,000 flows per second on the one gigahertz Athlon boxes using local storage. Typical pipelines such as flow-cat <dataset> | flow-filter | flow-stat <options> can reduce this down to around 245,000 flows per second, which is adequate for the number of reports we run and the size of the data sets currently in use. Our upgrade plans include a faster RAID controller and a newer version of FreeBSD with improved NFS and network performance.

We currently average 67,320 octets per flow and 92 packets per flow. One of our busiest routers handles 397 gigabytes of traffic per day (about 35 megabits per second) in 548 million packets which generates 5.9 million flow records per day. The flow-

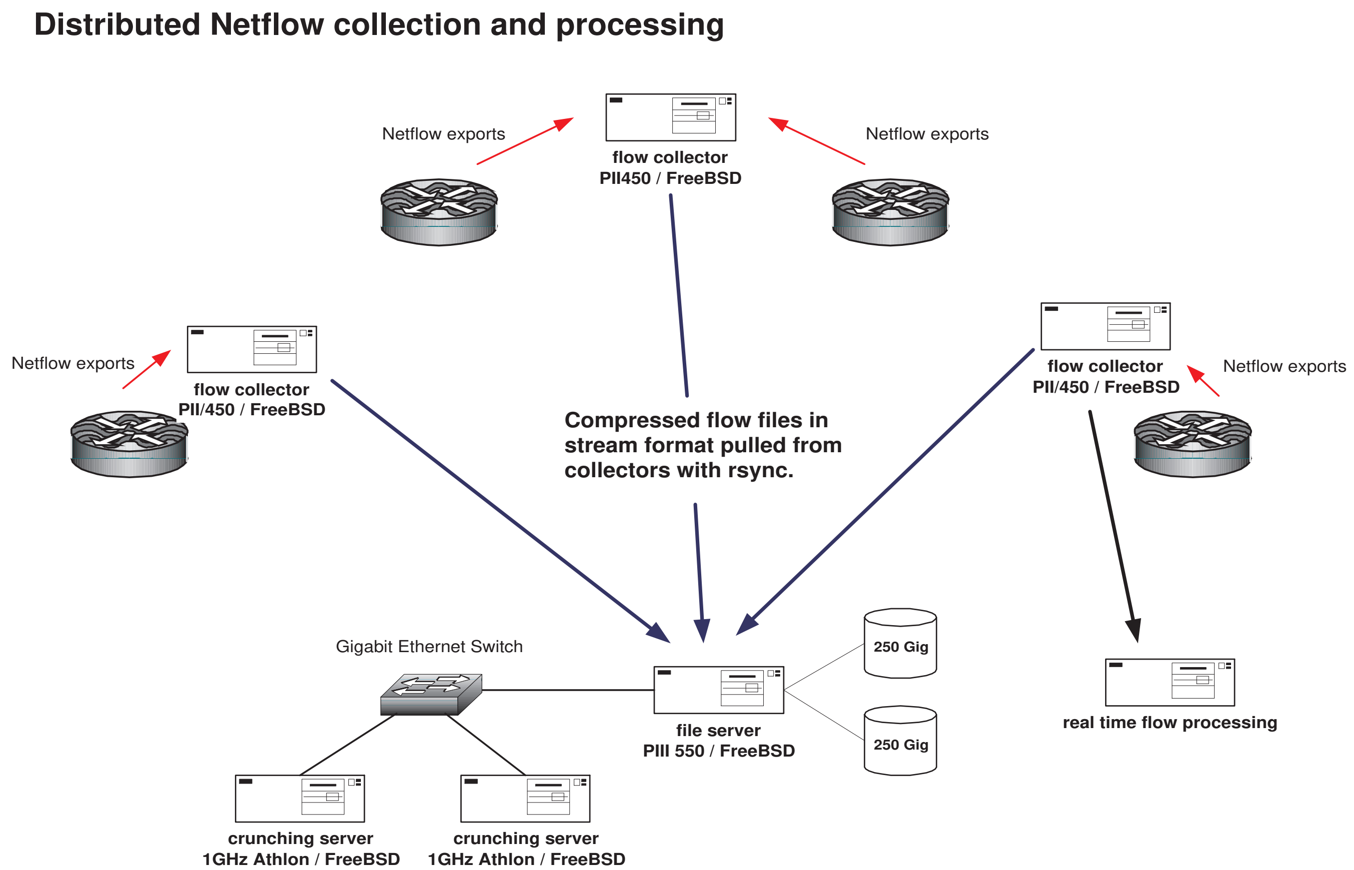

Figure 3: Simplified diagram of flow collectors at OSU.

tools flow record is 60 bytes long, so this works out to about 350 megabytes of flow logs per day at level 0 compression. We use level 6 compression by default, where we get a 4.3:1 compression ratio, so this actually works out to about 82 megabytes of actual disk space. This number can vary considerably – various denial of service attacks (especially SYN floods) can greatly increase the number of flows created, and accordingly, the number of flow records reported.

General Analysis Tools

Since our logs are split into relatively small files that represent network traffic for short time periods, we need a way to catenate these files together prior to processing. We can not simply use the UNIX cat program since each log starts with a general header that describes the contents of that log and this header needs to be removed from within the catenated files. The **flow-cat** program reads one or more flow logs (listed on the command line, though it will also read from stdin) and catenates the contents of these files in turn to stdout (or a designated output file).

Flow logs are written as binary records to a file and are not directly readable. The **flow-print** program reads records from a flow log and prints their contents in one of several output formats. For example, flow-cat * | flow-print -f 5 would print information about each of the flows in all of the logs in the current directory using format 5. Format 5 includes the start and end time of the flow, source and destination IP address and TCP or UDP ports, IP protocol type, source and destination interface numbers, TCP flags, and a count of the number of octets and packets for each flow. In the example in Figure 4 we have removed several of the output fields to make it more readable. The column labeled "p" is the IP protocol type – 6 is TCP, 17 is UDP. The column labeled "f" is the OR of the TCP flags for each flow. The last two columns, labeled "#" and "octets" show the total number of packets and octets for each flow. We also shortened the timestamp format in the "start time" column – normally timestamps are printed as MMDD.HH:MM:SS.SSS, so a timestamp of 0927.18:30:23.562 would represent the time 18:30:23.562 on September 27.

You can use command line options to cause flow-print to translate IP addresses and port numbers to names, where possible. We do not use this option very often, since we are used to working with numeric IP addresses and port numbers. Also, the translation to host names can lead to misleading interpretations due to DNS spoofing (through cache poisoning or other techniques) and due to port overloading (running a web daemon on TCP port 23, which would be reported as a telnet connection).

You can search flow logs and pull out interesting records with **flow-filter**. Flow-filter allows you to match records by the following fields:

- Source or destination autonomous system number (-a and -A options).
- Source or destination port number (-p and -P options).
- IP protocol type (-r option).
- Source or destination IP addresses, using Cisco standard Access Control Lists (ACLs, -S and -D options).
- Input or output interface numbers (-i and -I options).

Most of these options allow you to specify ranges and lists of values to match. For instance, flow-filter -r 6 -P 21,23,25,80 would match TCP flows (IP protocol type 6) with a destination port of 21, 23, 25 or 80 (which are usually FTP, telnet, SMTP and web services). The ability to filter flow records using Cisco standard ACLs allows us to perform powerful searches through our archives as part of incident investigations. The command flow-filter -f flow.acl -S attackers -D victims would read ACL definitions from a file named flow.acl (see Figure 5) and match flow records where the source IP address matches the "attackers" ACL and the destination IP address matches the "victims" ACL. You can also use the command line options to further filter the output by other fields, such as by IP protocol or TCP/UDP port.

We wrote a related script named flow-search to facilitate investigations of computer intrusions using the flow logs. Flow-search allows you to easily apply the same filter to a large number of separate log files. You could easily do this by applying flow-filter to the output of flow-cat, but this procedure disassociates the flow records from the name of the log file that they were found in, which can be inconvenient in incident investigations. Flow-search applies flow-filter to each log file individually and keeps the results in derivative files named after the source logs.

```
start         src               src dst                dst    p f  # octets
time          ip               port ip                 port

00:00:11.380 164.107.1.2       1026 205.188.254.195    4000  17 0  1     56
00:00:11.384 216.65.138.227    1055 164.107.1.3       28001  17 0  1     36
00:00:11.384 164.107.1.3      28001 216.65.138.227     1055  17 0  1     68
00:00:11.392 164.107.1.4      27015 24.93.115.123      1493  17 0  3   1129
00:00:11.392 164.107.1.5       1034 205.188.254.207    4000  17 0  1     48
00:00:11.392 128.146.1.7         53 206.152.182.1        53  17 0  1     61
00:00:11.404 204.202.129.230     80 140.254.1.6        1201   6 3 30  14719
```

Figure 4: Sample output from flow-print, edited to make it more compact.

Since flow records are created when their corresponding flows end the records are recorded in the log file in order of the ending time of these flows. This makes it hard to correctly interpret the contents of the flow logs, For example, you may suspect that an intruder is logging into a host through a backdoor of some sort, and you can see network activity coming from the host (scans, exploit attempts, IRC connections, etc) that lead you to believe that the intruder is active. But the flows associated with the backdoor connection may not show up in the flow logs until 30 minutes after the backdoor traffic actually started, so the other activity may actually occur first in the log. We wrote **flow-sort** to quickly sort the flow logs into chronological order according to the starting time of each flow. Flow-sort is implemented as a rolling heap sort, where flows are sorted into a heap which represents a 35 minute wide window (the maximum duration of a flow is 30 minutes). As flows move out of the rolling 35 minute window they are written out, now sorted by the starting time of the flow. Flow-sort can also save the heap when it reaches the end of the input and can restore from a saved heap when it starts up on a new file, which allows us to correctly sort flows between different log files.

Since flows are unidirectional and do not contain any indication about who initiated a connection, it can be difficult to correctly determine the client/server relationship between the source and destination hosts in each flow,. One can apply heuristics based on the transport level source and destination port numbers, or by maintaining state about previous network activity on each host (“host A created a connection to TCP port 21 on host B”) and using that to make inferences about unknown traffic (“this traffic from {TCP, A, 12345} to {TCP, B, 32145} might be a passive mode FTP data connection”). These are subject to mislabeling (“this specific traffic was a backdoor to host B”) or ambiguity (is traffic between TCP port 2000 and port 6000 is either OpenWindows, X, or something else entirely). The TCP flags field is of no help, since the same flags are usually seen in aggregates of packets in both directions. It would be useful if flows of TCP traffic where a TCP packet contained a SYN but not an ACK were marked in some fashion, but NetFlow services does not provide that information.

We can try to use the starting times of the flows to infer the client/server status of the endpoints – the source of the first flow is the one that initiated the connection. Flow-connect attempts to make these inferences from sorted flow records and aggregates flow records for the same session into a single flow record whose source IP is the client and whose destination IP is the server. At the time of this writing, this is still something of a work in progress, though preliminary results are encouraging.

Network Planning and Performance Tools

We have depended primarily on other tools such as statscout [13], MRTG [14] and a variety of homegrown tools for network planning, performance analysis and monitoring. We have found it useful to generate some common reports from the flow logs. **Flow-stat** summarizes flows into useful reports, some of which are easily viewable as graphs using tools like gnuplot. Flow-stat currently supports almost 20 different report styles, including summaries by source or destination AS number, port, or IP address, and source/destination matrix summaries. See the section on network planning and performance tools for some examples of how we use flow-stat.

Flow-profile is similar to flow-stat but provides the ability to aggregate flows based on groups of hosts as defined in a configuration file. This provides a powerful mechanism for additional data analysis, and is especially useful to generate usage data by group which can be used as a source for usage based billing. Flow-profile reads a configuration file which essentially describes the mapping between billing units and IP address ranges. The data from the flow records passed to flow-profile are aggregated according to these groupings and are summarized in the output, which is designed to serve as input to a billing system. See Figure 6 for an example of a configuration file, and Figure 7 for sample output.

The flow-tools package is a small set of programs for processing NetFlow exports that can be chained together, along with other standard utilities such as awk, grep, perl, sort, etc to produce reports. Leveraging off existing well-known utilities can lead to simple one-liners that produce detailed network activity reports or data sets for depicting long term

```
! permit anything
ip access-list standard all permit any

! match the attackers
ip access-list standard attackers permit 10.0.0.1 0.0.0.0
ip access-list standard attackers permit 128.146.222.0 0.0.0.255
ip access-list standard attackers deny any

! match the victims
ip access-list standard victims permit 140.254.1.1 0.0.0.0
ip access-list standard victims permit 140.254.1.2 0.0.0.0
ip access-list standard victims deny any
```

Figure 5: Example access control list definition file for flow-filter.

usage with utilities such as gnuplot or mtv. The following are examples of some of the types of reports that can be generated.

Top 10 Users

It is easy to create custom reports. Here is how we find the list of the top 10 network bandwidth users in our dorm network, ResNet (see Figure 8). We use flow-cat to catenate the data sets for a day's worth of logs together and pass the resulting output to flow-filter to isolate traffic with an input interface of 1 and an output interface of 47 or 62 (our connections to the outside world, see Figure 2). The output from flow-filter is passed to flow-stat, which we will instruct to generate a usage report based on source IP address (-f9) and to report the totals in percent/total form. Finally, we filter out the comments with grep and sort the results in descending order by the value in the total octets column. If you examine the numbers you will see that the top 10 users are using almost 30% of the total octets counted.

Counting Addresses

First, lets count the number of unique IP addresses on the Internet that have exchanged traffic with hosts at ResNet. We will use flow-filter to pull out just traffic going to the Internet, and flow-stat format 8 (statistics by destination IP address) to generate

```
# define the inside interface
inside 1 3 4

# define outside interface
outside 8

range 128.146.070.001   128.146.070.001 math
range 128.146.024.001   128.146.024.001 physics
range 128.146.225.001   128.146.225.001 economics
range 128.146.216.001   128.146.216.001 math
range 128.146.222.001   128.146.222.001 athletics
...
range 164.107.005.151   164.107.005.244 physics
range 128.146.050.240   128.146.050.244 psychobotany

# define these after ranges to make sure all addresses in a block
# have range definitions
bound 128.146.0.0 128.146.255.255
```

Figure 6: Sample configuration file for flow-profile.

```
#group          octets                  packets
#               in              out     in              out

math            358627478       3423321 3274516064      4518383
physics         7449413327      38512040 32410612445    44213679
psychobotany    2025644653      27020722 14947199604    28384258
english         212751          2283     2108750        2495
...
```

Figure 7: Sample output from flow-profile.

```
flow-cat cf05.2000-09-26.* | flow-filter -i1 -I47,62 | flow-stat -P -f9 \
                                        | grep -v '^#' | sort -n -r +2 -3

IP              flows    octets   packets  duration
--------------------------------------------------------
 164.107.70.189   0.077    7.477    3.669    1.053
 140.254.229.202  0.091    5.036    2.487    0.719
 140.254.233.119  0.116    3.874    1.936    0.824
 164.107.67.66    0.262    2.412    1.240    0.530
 140.254.106.226  0.122    2.358    1.306    0.754
 140.254.229.255  0.045    1.977    0.969    0.264
 164.107.86.143   0.059    1.835    0.988    0.337
 164.107.93.206   0.091    1.602    0.857    0.390
 164.107.90.220   0.238    1.390    0.742    0.352
 140.254.236.103  0.013    1.381    0.714    0.152
```

Figure 8: Using flow-stat to find the top 10 users.

the unique list, which we will count with the wc (Word Count) program: flow-cat cf05.2000-09-26.* | flow-filter -I47,62 | flow-stat -f8| grep -v ' #' | wc -l.

From this, we see that there were 945,189 unique destination IP addresses contacted.

Similarly, this will count unique source IP addresses at ResNet that have sent traffic to the Internet: flow-cat cf05.2000-09-26.* | flow-filter -i1 -I47,62 | flow-stat -f9 | grep -v ' #' | wc -l.

This reports that there are 6,209 unique source IP addresses active.

It might be interesting to count the unique IP addresses at ResNet that have received traffic from the Internet. We will turn the filter around (the destination filter is ResNet's internal network and the source filter is the Internet links) and count the unique destination addresses: flow-cat cf05.2000-09-26.* | flow-filter -I1 -i47,62 | flow-stat -f8| grep -v ' #' | wc -l.

This reports 11,739, which is greater than the actual number of hosts that generated traffic. This implies that the network had been host scanned at least once.

Finally, here is a way to count the Internet hosts that have sent traffic to ResNet:

```
flow-cat cf05.2000-09-26.* | \
     flow-filter -i47,62 |    \
     flow-stat -f9 |          \
     grep -v ' #' | wc -l
```

This reports 879,438, which is less than the number of hosts that we sent traffic to (945,189). This implies that ResNet hosts were involved in host scanning the Internet.

Security Tools

We most commonly use the flow logs in our various security functions, for incident response, intrusion detection, firewall planning, and security assessments and consultation.

Intrusion Detection

NetFlow logs are not useful for any form of content based intrusion detection since the flow records do not contain the data portion of the network traffic. We also do not have a detailed picture of the values in all of the headers for individual packets, and so we cannot use that to detect signatures of intrusions that leave calling signs there. We can use the flow logs to detect network access policy violations, to report on the activities of known suspicious hosts, and to detect some of the more obvious forms of scanning and denial of service attacks.

Flow-dscan is an attempt at detecting and reporting interesting network related events in near real-time. It can be configured to report:

- Excessive octets or packets per flow – typically floods.
- A source IP contacting more than a threshold of destinations – host scanning.
- A source IP contacting more than a threshold of destination ports on a single host. The port list is limited to 0..1023 to keep the memory overhead low.

Flow-dscan must keep a fairly large hash table of {source, destination} pairs in memory to be able to detect slow port and host scanning. Memory is allocated and managed dynamically based on the frequency of flow arrival. The ager and various table sizes can be user configured. Optional source and destination suppress lists are also supported in hash format for fast lookups. Typically on-line multi-player game servers and web-based add servers must be entered in the suppress list to prevent them from being reported as scans. Flow-dscan can either be run on archived data sets or by connecting to flow-capture for

```
>flow-cat /netflow/se2/cf05.2000-09-28.* | flow-filter -r6 -i1 -I47,62 \
                                            | flow-dscan -b -p -O -P

info(6):   port scan: src=140.254.103.62 dst=24.160.184.73 start=966204748

>flow-cat /netflow/se2/cf05.2000-09-28.* | \
        flow-filter -r6 -i1 -I47,62 -f flow.acl -S portscan | flow-print

Sif  SrcIPaddress     Dif  DstIPaddress      Pr SrcP DstP  Pkts       Octets
...
0001 140.254.103.62   003e 24.160.184.32     06 f78  15    1          60
0001 140.254.103.62   003e 24.160.184.32     06 f79  7     1          60
0001 140.254.103.62   003e 24.160.184.32     06 f7a  5d    1          60
0001 140.254.103.62   003e 24.160.184.32     06 f7b  13    1          60
...
0001 140.254.103.62   003e 24.160.184.32     06 e4b  13    1          60
0001 140.254.103.62   003e 24.160.184.32     06 e4c  5d    1          60
0001 140.254.103.62   003e 24.160.184.32     06 e4d  7     1          60
0001 140.254.103.62   003e 24.160.184.32     06 e4e  15    1          60
...
```

Figure 9: Results of running flow-dscan to detect a scan and flow-print to view the associated traffic.

real-time data. Flow-dscan is usually run in the background against a live feed of flow records, and reports its results through syslog. You can also force flow-dscan to run in the foreground and report to stdout, which is useful for interactive use against archived log files (see the section on intrusion detection). Visually inspecting the flows with flow-filter and flow-print can verify the scanning activity.

Normally flow-dscan would be run against a live feed of flow records, but you can also run it against archived logs. In Figure 9 we use flow-filter to pull out TCP traffic from OSU going to the Internet, and then run flow-dscan in the foreground on the resulting flow records. After running flow-dscan we used flow-filter to pull out traffic coming from the source of the scan and print it.

Although it is not practical to respond to every trigger it has been useful to log such activities so that we can refer back to them at a later date. The frequency of external sources scanning our network is so high that an insecure machine will almost certainly end up on someones database of vulnerable hosts within 24 hours of installation. Insecure hosts are frequently compromised within a day of being set up on the network.

A simple script named **flow-scan-report** takes a time range and IP address as arguments and uses flow-filter and flow-print to display flow activity to and from that address in the given time range. This makes it easy to pre-compute brief snapshots of network activity for each of the detects that turn up in IDB (our Incident Database, in the section ''Related Work'').

Incident Response

As we saw with the previous example, flow-filter is an effective tool for pulling interesting traffic out of the haystack. This is invaluable for incident response. For example, if we receive a report that one of our computers was involved in a scanning and intrusion incident at another Internet site, we can use the flow logs to:

- Confirm whether the alleged incident actually involved OSU.
- If it did, we can usually use the flow logs to determine what hosts the OSU host contacted by using flow-filter to search for traffic coming from the OSU host.
- We can also search for traffic going to the OSU host to determine whether it is being controlled from elsewhere.
- Once we identify the hosts used to compromise our hosts, we can search the flow logs for traffic from those hosts to OSU to discover other hosts that might have been compromised.

Iterating over the flow logs with varying options to flow-filter, flow-stat, and flow-print on each pass allows us to quickly determine to source(s) and destination(s) of DoS attacks and potentially the attacker and their arsenal of compromised hosts. Once the compromised hosts, victims, or attackers are isolated the IP addresses can be quickly disabled by use of a black-hole router which injects specially tagged prefixes into our backbone routers which get rewritten using route-maps to point to a non existant destination.

For example, early in the afternoon of July 3, 1999 we were alerted of slow or non-responsive network services on campus. It didn't take long to find that the inbound side of our OC3 connection to OARnet was full. Running the most recent flow logs through flow-filter to isolate inbound traffic and flow-stat to create a summarized traffic report by destination IP isolated the destination to a single host. A second run of the flow logs through flow-filter to isolate flows to that single IP revealed thousands of ICMP echo replies – a Smurf attack. Further analysis of the destination IP flow logs revealed an IRC client session which is a common ingredient on provoking a denial of service attack. Disabling the host with a black-hole route prompted the attackers to end the attack end shortly after.

Unfortunately the Smurf attack was only a precursor to the activity that followed later that day. Shortly after one of the evening fireworks displays our upstream provider sent a page informing us that severe denial of service attacks originating at OSU required them to shut down OSU's Internet connection. Most departments at OSU at this time were connected with 10 megabit per second Ethernet to the campus backbone, so in order to generate a full OC3 of traffic we knew that many hosts on many different LAN segments must be involved. Again, flow-filter was used to isolate outgoing traffic by filtering on the interface fields and flow-stat was used to generate a report based on destination IP, and we discovered the IP address of a single victim. We soon discovered that many hosts on the OSU network were sending high bandwidth UDP streams to the victim. Further analysis of a larger window of the flow logs using flow-filter and flow-stat to create a report of source IP addresses contacting the victim revealed about 43 sources on many different LAN segments participating in the attack. Using those sources in a filter we also found that multiple victims were involved over the course of the day.

Many of the compromised hosts on campus were not very active on the Internet, so it was easy to spot the IP address of the attacker in the flow logs and the TCP port used to start the UDP floods. The attack was controlled by a fairly simple set of Perl and shell scripts that connected to compromised hosts through a shell backdoor on the FTP port, from which it would run a script to download a UDP flooder called milk from another university through the Remote Copy Protocol (RCP). The milk script would then be run against one of several external targets.

Using black-hole routes to disable the compromised hosts, victims, and attacker proved effective in stopping the attacks. Later long term analysis of the flow logs for the compromised hosts revealed the method of break-in and several sets of hosts that were involved in what was apparently a distributed and automated scan and exploit. One last iteration over the archived flow records after the incident revealed that as many as 250 hosts at OSU were compromised in the initial set of breakins on July 2, although only about 50 of them were used for the UDP denial of service attack the next day.

The use of flow logs to home in on compromised hosts and their traffic has shown that it is not uncommon for a site to be scanned for vulnerabilities by one host, compromised at a later date by a second, contact a third site for downloading of denial of service and exploit tools, and then have the installed and waiting remote controlled denial of service programs be triggered at a later date by yet a fourth site to attack a victim.

Firewall Planning and Security Assessments

As we have instrumented the rest of our core routers and switches with NetFlow exports, we have come to increasingly rely on the flow logs for guidance in designing firewalls and in studying the network behavior of systems in site and product security evaluations.

Host Activity Profiling

Flow-host-profile builds a list of the network services running on each host on campus, and allows us to compare activity over a period of time with the existing profile to detect changes. We are especially interested in new hosts and new services that show up on campus. The sample output from flow-host-profile (Figure 10 shows activity for several services that we had not previously seen (e.g., HTTP services on 128.146.1.4). The new activity on port 7440 on 128.146.1.3 might be a new service, but sometimes busy clients have high end port numbers that show up in the report (as they get reused the connection count goes up, passing our activity threshold). We've been experimenting with looking at not just the presence of new services (and hosts) but also with changes in the level of activity of a service. For example, it would be interesting to know if activity on a usually quiet FTP server suddenly increases (possibly due to warez trading through a writable directory, for example). Unfortunately, flow-host-profile is very sensitive to traffic that evades its attempts to winkle out the client/server role of each endpoint, and tends to either create many false positives or becomes insensitive to low levels of intrusion activity if we turn the thresholds up to decrease the false positives.

Common Problems in Interpreting NetFlow Logs

We already discussed the difficulty of determining the client/server role of the hosts at each end of a flow (in the section on general analysis tools) and of interpreting the flows in light of the fact that they are ordered chronologically by the ending time of each flow. There are a number of other issues that you need to keep in mind when you are reading through flow logs.

It is important to recognize that source IP addresses are easily spoofed. This is common in many denial of service attacks or in chaff created by scanning tools like nmap. If you have unicast Reverse-Path Forwarding (RPF) checking enabled on your devices, or use Access Control Lists (ACLs) to drop spoofed traffic this traffic will be recorded in your flow exports with a destination interface of 0 (indicating that the traffic arrived, but was not forwarded). Of course, hosts within a LAN can still spoof the addresses of other hosts on the same LAN.

Note that NetFlow enabled devices only create flow entries for the traffic that actually passes through that device. If you have asymmetric routing conditions where outbound traffic passes through a different router than the analogous inbound traffic and you are only looking at the flows from one router, you will only see part of the flows representing those client/server sessions. To get a complete picture of the traffic in these cases you would have to combine the flow records from the devices that handle all of the traffic (through flow-cat and flow-sort). Note that since flow-sort uses a 35 minute window for sorting, you need to merge the files in chunks that span less than 35 minutes. This presumes that the clocks have been accurately synchronized to a common time source.

One issue that needs to be considered when using flow exports from the routers is the reliability of the data sets. Flows are currently exported via UDP with no ability for the collector to signal retransmission if it detects missing NetFlow PDUs. Each PDU contains a 32 bit sequence number that can be used to detect missing or out of order flows. Out of order flows could be an indication of an attacker trying to inject false PDUs into the collector. Missing flows could either indicate the flow-collector is overloaded, possibly due to an attack or the network segment

```
IP-ADDRESS   PORT SERVICE  PROTO CONNECTIONS DAYS   PERCENT
128.146.1.1    53 domain   17    3           1       100
128.146.1.2   162 snmptrap 17    3           1       3
128.146.1.3  7440 N/A       6    3           1       100
128.146.1.4    80 http      6    3           1       100
128.146.1.5   518 ntalk    17    3           1       0
```

Figure 10: Abbreviated output from flow-host-profile, showing new hosts and services.

connecting the flow collector to the router is over subscribed or under attack. The possibility of spoofed flows can be minimized by deploying unicast RPF or IP spoofing filters appropriately. Overloaded network segments can be avoided by directly connecting the collectors to a dedicated router interface on the router exporting the flows and limiting traffic on that network with access lists. Cryptographic signatures on flow PDUs and a reliable transport mechanism could reduce some of the potential problems, but not without the memory and CPU penalties on the routers and line cards. The potentially unreliable export mechanism and 32 bit sequence number is adequate for our current needs.

Privacy and Legal Concerns

The flow logs do not contain a record of what is usually considered the contents of the packets. This means that although we could determine that a given host accessed a given web server at a certain time, the flow logs would not contain a record of the URL requested or the response received. However, if you can correlate the activity recorded in the flow logs against the data in other logs (such as authentication logs), you might be able to match accounts (and so, to a large degree, people) to IP addresses, IP addresses to their associated network activity, and then match that network activity to specific details such as URLs requested, email addresses for correspondents, newsgroups read and so on. Consequently, the act of recording and archiving NetFlow records raises a number of privacy concerns.

In addition, OSU is a state university and as such is subject to the state public records laws, which indicate that most records created as part of the normal business processes of the university are ‘‘fair game’’ for disclosure requests from the general public. This of course raises a fair degree of concern that someone might request a copy of our NetFlow logs to determine what our employees are using their computers for. On the other hand, we are also subject to FERPA (Family Education Rights and Privacy Act) which indicates that student academic records are exempt from the public records laws. According to our lawyers, it is not clear whether the flow records for students would be considered protected by FERPA, or whether they are part of the public record. Since the flow logs cover a mixed population of students and non-students, and since we have no easy way to separate them, they enjoy a sort of murky, though dubious legal protection.

We try to protect the privacy of our customer base by storing the logs on secure servers, with limited access by staff members, and with clear access and use policies in place. We do not archive the raw logs to tape, although we do retain a fairly long window (currently about four months worth) on our central file server.

Our rationale is that the logs are invaluable for security, performance and network monitoring and usage based billing. We could aggregate the data and use that for some of these functions, which would solve most of the privacy concerns. However, having a long (2 to 3 month) window of past logs is invaluable for incident response, and we expect that it may prove invaluable for bill dispute resolution as well. We think that the level of detail present in the flow logs represents an acceptable balance between utility and privacy for our environment. On the positive side, we have found that we have had to do content based sniffing (e.g., with tcpdump [15]) far less often, since we have a ready source of information about network activity.

Related Work

Other groups have also been working on tools for collecting Cisco flow logs. In particular, you might be interested in looking at the CAIDA tool collection [16]. In particular, the cflowd [12] system provides a mechanism to collect, save and aggregate NetFlow exports and view a variety of summaries and graphical views of the collected data. There are also a number of tools that can be used with the cflowd package to create reports and graphs.

Cisco has also released a set of tools for collecting NetFlow logs [2]. These tools provide fairly sophisticated data aggregation, graphing and billing features, and as you might expect, support all of the versions of the NetFlow exports, including version 8 aggregated NetFlows.

OSU has integrated intrusions detected through flow-dscan into its Intrusion Database system (IDB). IDB is a web front end to a list of intrusion detects from a variety of sources, including host based scan detectors, snort and now flow-dscan. Our incident response staff reads through the detects in IDB and responds to them in a variety of ways. A hook from IDB allows the operator to easily view a summary of network traffic matching the detect, which is derived from the archived flow logs. Incidents that turn into full-fledged investigations are tracked through our incident tracking system, SITAR [9], which also has hooks that allow us to view the results of previous searches through the flow logs or to initiate new searches.

David Brumley at Stanford University has written a program that converts **argus** [1] logs into flow-tools format, allowing you to use this as an additional source of data for flow-tools [5]. Dave has also written a small perl script for detecting scans through threshold violations [6].

Larry Lidz at the University of Chicago modified the **extract** program from the netlog package [4] package to read the logs that flow-tools creates [7]. The resulting program, **flowextract**, allows you to easily create watch lists to report network activity for known hostile hosts, or for critical hosts in your environment, or to identify traffic that might indicate hostile activity (for instance, traffic to TCP port 31337). Larry has also written a program called **flow-merge** which merge sorts flow logs from multiple routers into a single log. This is helpful in cases where multiple routing paths lead to asymmetric routing conditions, or where you need to search all of your connections to the Internet for intrusion activity.

Simon Leinen has put together a great summary of flow related references and tools which is well worth looking at [11].

Future Plans

We are often asked why we have not merged our work with the cflowd package. There are several reasons, chief among them being motivation and philosophy. We have not had a need to use the presentation and analysis tools that cflowd provides since we have other tools to do that work. Cflowd appears to have been designed to facilitate the aggregation of data directly from NetFlow exports, and most of the tools written to process flow data from cflowd work with these aggregations rather than with raw flow data. The NetFlow capture mechanism in the flow-tools was designed to efficiently store and manipulate flow records in compressed form, and data aggregation is done by tools operating on these raw logs (if at all). Its design also allows us to take a highly distributed approach to data collection and analysis. It is possible to either write something to export the flow-tools logs into whatever format the cflowd package uses internally, or to duplicate the incoming UDP NetFlow records into parallel incarnations of flow-capture and cflowd.

The principle problems with using the flow logs for intrusion detection are that it is difficult to correctly determine the client/server role of the two endpoints of a flow and that the flows do not contain packet contents. We have not converted our intrusion detection systems to use the sorted flows from flow-sort – we expect that this would allow us to both increase the sensitivity of flow-host-profile and flow-dscan, and also greatly reduce the frequency of false positives.

Two papers by Yin Zhang and Vern Paxson describe their work with detecting backdoors [17] and detecting stepping stones [18] using packet size, packet timing characteristics, and correlations between flows of traffic. Aggregating packets into flows obscures some of the detailed information that they used in their algorithms (individual packet sizes, delays between packets), but it might be possible to adapt their work to flow based data using average packet sizes from the flows and calculated average delays, or using flow sizes and delays between the flows. We hope to start working on this soon.

We will soon be adding support for the NetFlow version 7 PDU (used on some Cisco switches). Cisco also has a version 8 PDU which is used for exports of aggregated flow data [3]. We plan to add this at some point, but doing so will require development of a different set of analysis tools, since the current tools are designed to work with flows, and not with the aggregated data in tables that are supplied in the version 8 PDU.

We also plan to add support to flow-capture to use the sequence numbers in the version 5 and version 7 NetFlow export headers to detect and report missing NetFlow exports. This addresses some of our concerns about using the flow logs in incident investigations since we will at least be able to tell when critical records are possibly missing, or note that there are no missing records.

The current filtering mechanisms in flow-filter are sufficient for a wide range of tasks, but it would be nice to provide more powerful filters. We plan to fully implement Cisco extended access control lists.

We are also working toward building more powerful interactive front ends for browsing our flow logs.

Conclusion

Cisco NetFlow exports and the flow-tools package contribute to our ever growing toolbox of network management resources at Ohio State University. We are collecting and archiving flow exports from twenty routers with eleven flow collectors, one file server and two high performance data crunching servers. Our campus network has over 500 LAN interfaces, 15 remote sites and multiple external connections including OARnet, Abilene, and a local peering with the Columbus cable modem service provider. An online archive of 500 Gigabytes of compressed flow exports allows turning back the clock on network disruptions and security incidents to provide a post-mortem birds eye view of events leading up to and surrounding an incident. Real-time processing of flow data can shed light on difficult to pinpoint activities by allowing us to view traffic patterns without having to deploy sniffers or LAN probe devices on every LAN segment or WAN link. Departments traffic reports can be generated for potential future cost recovery of network services, and over all network usage reports are used to determine the impacts of popular new applications such as Napster.

The OSU flow tools are available at http:/www.net.ohio-state.edu/software/flow-tools.shtml. We also have a mailing list (flow-tools@net.ohio-state.edu) – you can subscribe by sending an empty message to flow-tools-subscribe@net.ohio-state.edu.

Mark Fullmer wrote the bulk of the OSU flow tools collection. Steve Romig mostly just uses the tools and makes suggestions for further development, though he has recently been seen writing documentation, fixing various bugs, and finding students to write tools like flow-sort, flow-host-profile and flow-connect. Ron Luman wrote flow-sort and flow-connect.

Suresh Ramachandran wrote the flow-host-profile program.

Biographies

Steve Romig in charge of the Ohio State University Incident Response Team, which provides incident response assistance, training, consulting, and security auditing service for The Ohio State University community. In years past Steve has worked as lead UNIX system administrator at one site with 40,000 users and 12 hosts and another site with 3,000 users and over 500 hosts. You can reach him by phone at 1-614-688-3412 (GMT-0400/0500) or by email at romig@net.ohio-state.edu.

Mark Fullmer is a recovering system and network administrator from a large.edu, currently working on his degree at The Ohio State University and is employed part time in the OARnet engineering group.

Bibliography

[1] Bullard, Carter <chellyaz@aol.com>, *Audit Record Generation and Utilization System (Argus)*, ftp://ftp.andrew.cmu.edu/pub/argus .

[2] Cisco, *Cisco Netflow Flowcollector*, http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/nfc .

[3] Cisco, *Netflow services and applications white paper*, http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm .

[4] Schales, Douglas Lee, David R. Safford, and David K. Hess, "The TAMUSecurity Package: An Ongoing Response to Internet Intruders in an Academic Environment," *Proceedings of Fourth USENIX UNIX Security Conference*, anon ftp at coast.cs.purdue.edu in /pub/tools/unix/logutils/netlog, 1993.

[5] Brumley, David <dbrumley@theorygroup.com>, *Argus Export Scripts*, contact Dave by email for copies.

[6] Brumley, David <dbrumley@theorygroup.com>, *How to detect network scans,* http://www.theorygroup.com/Theory/scans.html .

[7] Lidz, E. Larry <ellidz@eridu.uchicago.edu>, *flowextract*, http://security.uchicago.edu/tools/net-forensics .

[8] Hobbit <hobbit@avian.org>, *netcat*, http://www.l0pht.com/weld/netcat .

[9] Assor, Mowgli <mowgli@net.ohio-state.edu>, *Security and Incident Tracking And Response (Sitar)*, http://www.net.ohio-state.edu/software/sitar.shtml .

[10] Gailly, J-L. and P. Deutsch, *Zlib compressed data format specification version 3.3*, http://www.ietf.org/rfc/rfc1950.txt .

[11] Leinen, Simon <simon@switch.ch>, *Flow based monitoring and analysis*, http://www.switch.ch/tf-tant/floma .

[12] The CAIDA Web Site, *cflowd: Traffic Flow Analysis Tool*, http://www.caida.org/tools/measurement/cflowd .

[13] Statscout, *Statscount network performance monitor,* http://www.statscout.com .

[14] Rand, Dave <dlr@bungi.com> and Tobias Oetiker <oetiker@ee.ethz.ch>, *Mrtg: Multi Router Traffic Grapher*, http://mrtg.hdl.com/mrtg.html .

[15] Leres, Craig, Van Jacobson, and Steven McCanne, *The tcpdump software package*, anon ftp from coast.cs.purdue.edu in /pub/tools/unix/tcpdump .

[16] Various, *The Caida Web Site*, http://www.caida.org .

[17] Zhang, Yin and Vern Paxson, "Detecting Backdoors," *Proceedings of Ninth USENIX Security Symposium*, 2000.

[18] Zhang, Yin and Vern Paxson, "Detecting stepping stones," *Proceedings of Ninth USENIX Security Symposium*, 2000.