Sets, Bags, and Rock and Roll*

Analyzing Large Data Sets of Network Data

John McHugh[1]

Cert Coordination Center, Carnegie Mellon University, Pittsburgh, PA 15313, USA,
jmchugh@cert.org

Abstract. As network traffic increases, the problems associated with monitoring and analyzing the traffic on high speed networks become increasingly difficult. In this paper, we introduce a new conceptual framework based on sets of IP adresses, for coming to grips with this problem. The analytical techniques are described and illustrated with examples drawn from a dataset collected from a large operational network.

1 Introduction

It is not unusual for relatively modest networks today to exhibit trans border flows on the order of megabits per second. Monitoring even a small network with a few hundred hosts can generate many gigabytes of TCPDUMP data per day. Capturing only headers can reduce the volume somewhat, and more compact formats based on abstractions such as Cisco's NetFlow can reduce the volume further. Even so, the volume of data collected is sufficient to overwhelm many analysis tools and techniques. In general, the problem is one of grouping and classifying the data in such a way that uninteresting phenominae can be pushed aside, allowing the investigator to extract and further scrutenize tata that is of interest. Recently, CERT has been involved in the analysis of large sets of NetFlow data. To support this effort, they have developed a set of tools, collectively known as the SiLKtools[1]. In the remainder of the paper, we begin by sketching our thesis and analysis approach. We then digress to describe the NetFlow data collected, noting that the analysis can be applied equally well to tcpdump or other data forms with a bit of preprocessing. The basic functionality of the SiLKtools suite and some of the extensions made in support of our analysis efforts are then described. The remainder of the paper will present examples of the analyses that we can perform using the tools suplemented by relatively simple programs to characterize and organize the reduced data. The paper concludes with a discussion of our plans for further extensions of the tools and additional analyses.

---

* The mantra "Sex, Drugs, and Rock and Roll" enjoyed currency in the 1960s. To the ears of an older generation, Rock and Roll was just a particularly unpleasant form of noise. Since the general theme of this paper is separating signal from noise in network data, the title is not too strained.

[1] The SLK are the initials of the late Suresh Konda who was instrumental in the initial development of the tool set.

2 The Thesis and Approach

Sets and set theory are abstractions that facilitate reasoning about many classes of problems. We have been exploring the use of sets to provide a compace way of describing and reasoning about the internet and about traffic observed at various points on it. For example, it is useful to consider such things as the set of hosts on a given network that are active during a given time interval. It might also be useful to consider the set of external hosts that are observed performing a questionable activity such as scanning during such an interval. Similarly, one might want to identify the set of users of some service provided by the local network to the outside world (e.g. web services) during the interval. In the case of the first set, the set of active machines, we could attempt to obtain the answer by asking the system administrators, by examining the state of the DHCP server responsible for leasing IP addresses to the network, by consulting the responsible DNS server, or we could approximate the result by observing those hosts within the network that either originate traffic or respond to connection requests. If we can observe the traffic passing in and out of the network at some border point such as a border router, the observations may coinstitute a reasonable approximation of the active set[2] of hosts. The set of active hosts, can be used to partition incoming traffic into that addressed to active hosts (hits) and that which is not (missses). Arguably, the later partition consists of a mix of malicious traffic, indirect evidence of malicious traffic, and, possibly, some amount of benign, but misdirected traffic. This partition can be further processed to identify interesting classses of senders. For example, originators attempting TCP connections will send packets containing a SYN flag. If we select flows containing SYN flags and count the number of flows per source address using a "bag[3]", we can sort the bag and identify high volume scanners. It is not uncommon see a single host performing scans of an entire /16 network in the course of a relatively few minutes. Having identified such a scanner, it is trivial to find the flows from the scanner to the active or hit partition and create the set of active machines included in the flow. At that point, it is useful to determine if any of the targets responded to the scanner, and, if so to examine the traffic between the scanner and the target (and the subsequent behavior of the target) to determine if the target has changed its behavior in ways that might indicate that it has been compromised.

As can be seen from the example of the previous paragraph, the use of sets and bags, combined with simple filtering based on properties of the data records themselves allows the clustering of data with some particular security (or other) propertys in common. Since we are dealing with many thousands of flows per

[2] We assume that all traffic in and out of the monitored network passes through an observation point. Multiple border crossing points are possible. For the moment, we assume that hosts within the monitored network do not spoof addresses and that multiple responders such as honeypots are not deployed within the monitored network.

[3] A bag is a counted set or multiset in which the number of occurances of each member of the basis set is recorded.

minute on large networks, the constructions of sets and bags allows us to abstract from individual behaviors to clusters of activities. As the paper develops, we will elaborate on this thesis and develop the tools and techniques that we need in more detail, however, we have a number of utilities available including:

- An efficient set representation that allows us to represent IPv4 address sets directly in memory. There is also a compact disk representation that can be read and written efficiently
- An extension of the set representation, a bag, that allows a 32bit counter to be associated with each IP address. It too has an efficient disk representation.
- Routines that allow set unions and intersections to be computed, producing additional set files.
- Routines that allow sets and bags to be created from files containing network flow data.
- Routines that allow sets and bags to be created from ascii lists of IP addresses in both "dotted" form (possibly containing wild cards), and in unsignede integer form.
- Routines to list the contents of sets and bags at various levels of detail, including the network structure (subnet relationships) of a set.

These are sufficient for our initial analysis, though we plan to add other programs to the suite as the need for them becomes clear.

3 NetFlow and other data sources

NetFlow was developed by Cisco as a mechanism for gathering traffic statistics to support billing and network management. NetFlow operates on routers and switches to report traffic statistics on a per interface basis. Although it is not standardized, it is supported in more or less compatible ways by a number of other router and switch manufacturers. According to Cisco[4], the detailed traffic statistics collected by NetFlow include:

- Source and destination IP addresses
- Next hop address
- Input and output interface numbers
- Number of packets in the flow
- Total bytes (octets) in the flow
- First and last time stamps of packets that were switched as part of this flow
- Source and destination port numbers
- Protocol (and flags as appropriate)
- Type of service (ToS)
- Source and destination autonomous system (AS) numbers, either origin or peer (present in V5 and select V8 datagrams)

[4] `http://www.cisco.com/en/US/products/sw/netmgtsw/ps1964/products_user_guide_chapter09186a00801ed569.html`

– Source and destination prefix mask bits (present in V5, V7, and V8 datagrams)

Note that NetFlow records include one or more packets, and represent unidirectional flows. As such, NetFlow lies somewhere in between tcpdump records which contain data about individual packets and connection records which would abstract an entire tcp session to a single record. Because NetFlow is resource intensive, there is a limit to the number of open flow records that the router can maintain at one time. New records are created whenever a new flow is seen. A flow is deemed to be new if it contains a (source/destination/protocol[5]) set that is not currently being monitored. A flow is closed if it has been inactive for a prescribed period of time(typically some seconds), if it has been explicitly closed (TCP FIN or RST), or if it has been open and active for a prescribed period of time (typically some minutes). Note that this has the effect of breaking up long, steady TCP sessions as well as intermitant TCP sessions with long pauses. It also creates pseudo sessions from sequences of UDP packets as might be associated with streaming media.

The individual flow records are aggregated and sent in batches encapsullated in a UDP packet to a central collection point for processing. In our case, the processing point stores the flow records in a compact format that can be sequentially searched and extracted based on times and a number of match criteria as discussed in the next section. Since our tools operate from this format, it is worth considering whether other forms of data might be stored in the same format and processed with the tools. The answer is a qualified yes. It would be trivial to extract most of the required data from packet based sources such as tcpdump or Argus records. Since packet data is typically captured on a link, router specific information such as interfaces, AS numbers, and next hop addresses are not available, but these seldom appear in our analysis. If we were to aggregate data from a number of collection points, these fields could be used to indicate the collection point and directionality of the packet.

Degenerate flow records could be constructed on a packet by packet basis. This would obviate the space efficiencies of the flow aggregation paradigm, but would be trivial to implement, either in real time or for prerecorded data.. For prerecorded packet header data where real time performance is not critical, flow aggregation could be done by implementing the NetFlow aggregation or a variation thereof. Given the availability of large amounts of ram, the rules for closing flows could be relaxed somewhat, and a low memeory garbage collection mechanism implemented that would preferentially close least recently updated flows when memory became low. This should lead to the optimum of one flow per session for most well formed TCP flows. We hope to add packet to NetFlow conversion code top the SiLKtools suite in the near future.

[5] In the case of TCP and UDP, ports are included.

4 The SiLKtools suite and its extensions

According to the SiLK website[6]:

> SiLK, the System for Internet-Level Knowledge, is a collection of netflow tools developed by the CERT/AC to facilitate security analysis in large networks. SiLK consists of a suite of tools which collect and examine netflow data, allowing analysts to rapidly query large sets of data. SiLK was explicitly designed with a tradeoff in mind: while traffic summaries do not provide packet-by-packet (in particular, payload) information, they are also considerably more compact and consequently can be used to acquire a wider view of network traffic problems.
>
> SiLK consists of two sets of tools: a packing system[7] and analysis suite[8]. The packing system receives Netflow V5 PDU's and converts them into a more space efficient format, recording the packed records into service-specific binary flat files. The analysis suite consists of tools which can read these flat files and then perform various query operations, ranging from per-record filtering to statistical analysis of groups of records. The analysis tools interoperate using pipes, allowing a user to develop a relatively sophisticated query from a simple beginning.
>
> The vast majority of the current code-base is implemented in C, Perl, or Python. This code has been tested on Linux, Solaris, Free/OpenBSD, AIX and Mac OS X, but should be usable with little or no change on other Unix platforms.
>
> The SiLK software components are released under the GPL.
>
> The project is the fruits of work done at the CERT Coordination Center (CERT/CC) that is part of the Software Engineering Institute at Carnegie Mellon University.

The analysis suite includes a number of applications and utility programs we discuss in some detail only those that are used in the examples below, however, manual pages for the entire suite are available from the web site. For convinience, we refer to the packed data files used by some of the programs as "rwdata" files. In most cases, input can come from stdin or from a rwdata file and it is posssible to associate an output of most programs with stdout, allowing chains of programs.

---

[6] `http://silktools.sourceforge.net/`

[7] The SiLK Packing System is a server application that receives Netflow V5 PDU's and converts them into a more space efficient format, recording the packed records into service-specific binary flat files. Files are organized in a time-based directory heirarchy with files cover an hour at the leaves.

[8] The SiLK Analysis Suite is a collection of command-line tools for querying packed netflow data. The most important tool is rwfilter, an application for querying the central netflow data repository for netflow records that satisfy a set of filtering options.

rwfilter is the frontend for the system. It extracts a subset of the centrally stored rwdata based on a number of aggregation and filtering criteria or applies filtering data to a previously extracted rwdata file. Although it is much more general than the usages we show, our typical use starts by selecting data based on some time interval and possibly some range of network addresses. This data will be stored in a single file and subsequently refiltered based on other criteria. The program can be used to partition data based on the filtering criteria. Thus, TCP data could be extracted to a file in one pass with other protocols going to different file (or piped to another invocation of rwfilter to extract UDP, etc. for as many levels as desired.) Since sets of ip addresses can be used as filter criteria, traffic whose source (and / or destination) addresses appear in given sets can be extracted or partitioned.

rwcut lists selected fields from an rwdata file. The records are listed one per line in the field order specified on the command line. This allows the extraction of specific fields for subsequent analysis. In addition to the explicit fields, rwcut supports a dynamic library feature that allows derived quantities or selection criteria to be specified. As an example, rwdata files contain total bytes and packet count fields, but it is often more useful to spesk in terms of bytes per packets when extracting scan data. A dynamic library to support this usage exists.

rwset creates sets of ipaddresses from rwdata files. In its original form, the user specifies whether the set should contain source or destination addresses. We have extended this to support the additional functionality described under rwsuperset.

rwsuperset can create multiple sets and bags from rwdata files. It is possible to greate sets containing source addresses, destiunation addresses, or both. Similarly, bags can be created based on source addresses, destination addresses, or both. An additional feature allows a diminished set of the input data to be passed to one or both of two additional outputs associated with the source and destination addresses, respectively. If one of these outputs is specified, a set is created for the source or destination as specified and data is passed to the output only if its source (destination) address already appears in the set. This creates a set of addresses that appear one or more times in the input stream and an output stream containing addresses that appear two or more times. As we will see, the address frequency is often skewed towards small numbers of occurences and this mechanism allows sets (requiring about 1 bit per entry) to be used for large numbers of addresses with small counts while bags (requiring 32 bits/entry) are reserved for addresses with larger counts. Experience shows that 9 sets with steadly diminishing size can reduce the space required for the remaining bag (counts of 10 or more) by more than 99%. As currently implemented, the data structure used for a set consists of an array of 2^16 pointers to blocks of 2^16 bits. The latter are only allocated when an address in the range of the pointer matching the appropriate /16 network block is seen. If every /16 has at least 1 IP, the set will occupy a bit over a half a gigabyte, not unreasonable in machines with 4 gigabyte address spaces and at least that much physical memory. The bag has a similar

structure, but an additional level of indirection, 2^16 pointers to 2^8 pointers to blocks of 2^8 32 bit counters. Again, secondary pointers and blocks are allocated only as needed. A fully populated bag would require a little over 16 gigabytes on memory, so we resort to the cascaded set representation described above. The disk representation of both structures contains a list of 32 bit headers containing a /24 network prefix followed by a block of either 256 bits or 256 32 bit counters, as needed.

readset reads in a set file and lists its contents in various ways. by default, it simply lists the set size. It can also list the members of the set, in dotted or unsigned integer form, and it can list the network structure of the set in terms of any or all of /24, /16, and /8 prefixes with or without host counts and summaries.

readbag is similar to readset, but will only enumerate the bag, listing each nonzero IP address and its corresponding count. IPs can be printed either in dotted or unsigned integer form and the count can be printed first to facilitate sorting as in listing the 10 IP addresses with the highest count (pipe readset to sort to head)

buildset builds a set file from a list of IP addresses. Wildcards are permitted, so giving buildset the single line "10.1.x.x" would create the set from "10.1.0.0" to "10.1.255.255" inclusive. Sets constructed in this fashion can be used to extract a subnet of interest from a set file created from rwfilter output. In one case, we were able to identify misrouted data in a large tcpdump trace by forming a list of destination addresses from the data, converting them into a set with buildset, creating another set describing the monitored network and differencing them.

buildbag is like buildset except that it counts the number of occurances of each IP address in its input list. It also accepts wild cards, so the inputs "10.1.0.x" and "10.1.0.0-127" would create a bag with counts of two associated with the first 128 addresses of 10.1.0.x" and a count of 1 associated with the remaining 128.

setintersect performs intersection operations on set files. It also can perform intersection with the complement of a set file. The command line allows the user to specify one or more "add files" which are intersected and one or more "subtract files," the union of which is complemented and intersected[9] with the result of the intersection of the add files. The output is another set file, a list of IPs, or both. If the list is produced, it may be in either dotted or unsigned integer form.

rwsetunion performs the union of two or more set files.

rwcat concatenates multiple rwdata files into a single file or stdout stream. Many of the programs in the suite will only work from a single file or stream and this provides a mechanism for combining files. Because rwdata files have a self describing header, the system "cat" utility is not applicable here. An example might be to join hourly files in order to perform an analysis at the daily level.

[9] In the implementation, entries in each subtract file are removed from the set resulting from the intersection of the add files if they are present in it.

rwsort provides the ability to sort rwdata files based on address, size or start time fields. Note that data enters the rwdata archives as flows are closed and thus is not ordered by start time unless it is sorted this way. rwsort performs a memory sort and is limited to files with $5 * 10^7$ or fewer records. No rwmerge program is available at the present time.

rwstats can provide a variety of statistics about the contents of an rwdata file.

5 Examples and sample analyses

In this section, we illustrate our analysis techniques with two examples. One is a brief data sample from a large cross section of networks that have been aggregated together. the other represents a detailed view of a weeks activity on a /16. In all cases, no real IPs are contained in the analyses.

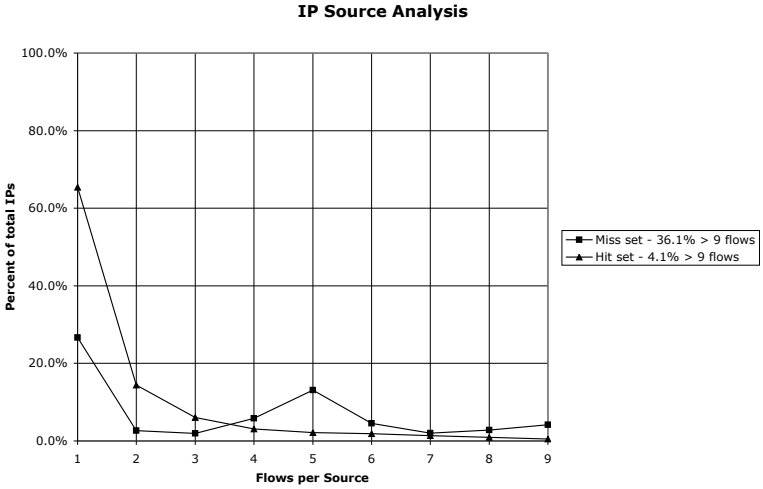

Fig. 1. Reduction in IP Source Set Sizes as a Function of Number of Flows

5.1 A brief examination of a large sample

We obtained data from a cross section of monitored networks for a small interval of time. Data leaving the networks was monitored for an interval began slightly

before interval used for data entering the network and extended slightly beyond it. This insures that the internal hosts that responded to traffic from the outside are included in the sample even in the incoming clocks are not perfectly synchronized with the outgoing clocks[10] or internal hosts respond only after a small delay. The set of active hosts in the internal network is approximated by creating the set of source addresses observed in the outgiong traffic during the covering interval. This set is then used to partition the data entering the network into flows directed to active hosts and those directed to inactive or non-existant hosts. About 2/3 (65.4%) of the flow records are directed at nonexistant targets, the remaining 1/3 (34.6%) are directed at active hosts. Further partitioning by protocol simplifies subsequent analysis. As expected, the vast majority of the data is TCP. We further analyze the TCP data by creating bags for source and destination addresses for the hit and miss partitions. The fall off in set size is illustrated in Figures 1 and 2. Note that hit and miss sizes follow similar, but distince patterns. The big differences are between source and destination behaviors. About 36% of the destination IPs that do not hit an active host have ten or more flows associated with them.

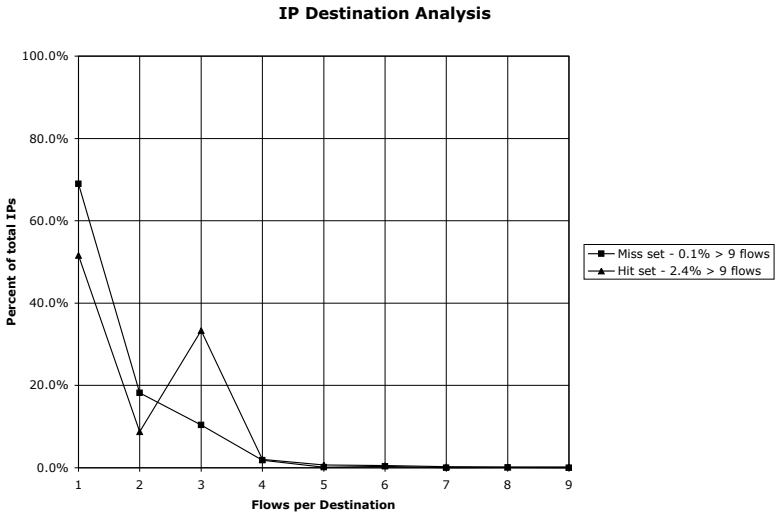

Fig. 2. Reduction in IP Destination Set Sizes as a Function of Number of Flows

[10] The data is aggregated from a number of routers. In some cases, separate routers handle incoming and outgoing traffic.

```
(39) lip $ readbag --count --print jcm-tcp-s-10+.bag| sort -r -n | head
     12994 AAA.BBB.068.218
      6598 CCC.DDD.209.215
      5944 EEE.FFF.125.117
      5465 GGG.HHH.114.052
      5303 III.JJJ.164.126
```

A closer look at the top address is interesting. Filtering for this address in the miss file (jcm-tcp.rwf) extracts some 400K of records. The destination set for this contains 12994 hosts, all from the same /16, XXX.YYY.x.x. The hit set also contains entries from XXX.YYY, 7 in all. All the flows sent to these addresses are 48 byte SYN packets addressed to port 4899 (listed as a "radmin" port by IANA, with other possible usages reported as ChiliASP and iMesh). An inspection of the outgoing traffic to this network indicates no responses to the connection attempts.

The second entry is somewhat different. The traffic from this address scans a different /16. looking for responses on port 7100[11] (X Font service according to IANA). Some 112 responses are seen from hosts SSS.RRR.QQQ.1-78,120-131,224-254. The contiguous ranges ans the consistency of responses on a relatively obscure port may indicate the presence of a honeypot or a similar decoy. In all cases, the connection seems to have been broken after the probed host's SYN/ACK reply.

The third and fourth entries are scans of portions of other /16s, this time looking for service on port 20168. This appears to be associated with the "Lovegate" worm which binds a shell to this port on infected machines.

The fifth entry is a scan of a portion of yet another /16, this time looking for service on port 3127, listed by IANA as the CTX bridge port, but also in the range used by the ChiliASP module in Apache servers acccording to `www,portsdb.org`. This is currently being used by the "MyDoom.C" worm on Linux[12]

At the other end of the spectrum, there are 3335 external hosts that sent exactly one tcp flow into the monitored network during the analyzed time interval. Of these, only two port and flag combinations appear more than 100 times. SYN probes for port 8866[13] are seen 449 times. SYN probes for port 25 (SMTP - email) are seen 271 times. The vast majority of the remainder are SYNs to a variety of ports, mostly with high portnumbers. There are a number of ACK/RST packets which are probably associated with responses to spoofed DDoS attacks.

---

[11] `http://www.cert.org/advisories/CA-2002-34.html` describes a vulnerability in the X font service on solaris. It is likely that the scanner was looking for machines that could be attacked.

[12] `http://www.linuxworld.com/story/43628.htm`

[13] "W32.Beagle.B@mm is a mass-mailing worm that opens a backdoor on TCP port 8866. The worm uses its own SMTP engine for email propagation. It can also send to the attacker the port on which the backdoor listens, as well as a randomized ID number." according to `http://securityresponse.symantec.com/avcenter/venc/data/w32.beagle.b@mm.html`

5.2 A week in the life of a /16

We obtained hourly flow data from a /16 within the monitored network for the one week period from 11 - 17 January 2004. plus two additional days, 26 and 27 January. The data set consists of nearly 400Mb of data divided into hourly files for inside to outside traffic and for outside to inside traffic. The inside to outside traffic was analyzed and a set of IP addresses computed that represent all the hosts that were seen to be active during the initial week. The observed network structure is shown in Table 1.

Table 1. Network Structure for a /16

MMM.NNN.24.x	66 hosts	MMM.NNN.25.x	60 hosts
MMM.NNN.26.x	46 hosts	MMM.NNN.27.x	49 hosts
MMM.NNN.28.x	57 hosts	MMM.NNN.29.x	7 hosts
MMM.NNN.30.x	70 hosts	MMM.NNN.31.x	67 hosts
MMM.NNN.32.x	54 hosts	MMM.NNN.33.x	62 hosts
MMM.NNN.34.x	50 hosts	MMM.NNN.35.x	4 hosts
MMM.NNN.120.x	2 hosts	MMM.NNN.127.x	1 host
MMM.NNN.140.x	1 host	MMM.NNN.251.x	4 hosts

Network Summary
600 hosts ($1.4 * 10^{-5}$%) of 2^{32}
1 occupied class /8 (0.4%) of 256
1 occupied class /16 (0.002%) of 65536
16 occupied class /24s ($9.5 * 10^{-5}$%) of 2^{24}

The set of active addresses seen during the week is an optimistic estimate of the active host set for this network since activity on several subnets was not observed until late in the week. The set includes all hosts seen to be active during the additional days, as well. This set was used to partition each outside to inside hourly data set into hit and miss portions as described earlier. Since these files consist of a small amount of header data and fixed length records, the file sizes are a good surrogate for the number of flows observed.

Figure 3 shows the hit and miss file sizes for the base week of the data. It is interesting to note that many more flows miss than hit during most hours. We have looked at a few of the peaks in detail.

12 January 21:00 Two fairly complete scans of the entire /16, one for port 2000 TCP (61018 flows) and one for port 20168 TCP (60978 flows) with a total of 62616 unique IP addresses hit. Both scans came from a single IP address. As noted above, port 20168 is associated with the Lovegate worm. Port 2000 appears to be associated with a number of Windows remote administration tools (e.g. RemotelyAnywhere).

14 January 15:00 Two scans from different networks, each targeting port 80 TCP. One targets 58394 hosts, the other 53032.

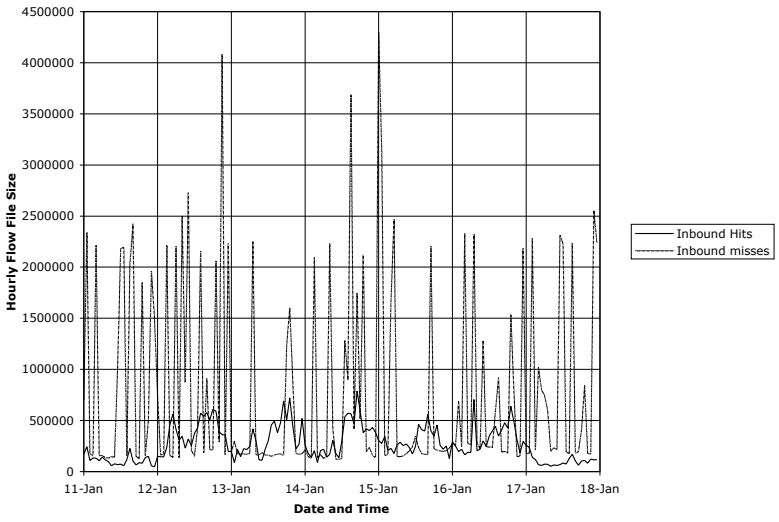

Fig. 3. Hourly flow file sizes for incoming hit and miss flows

15 January 00:00 Two fairly complete scans from two distinct source addresses, one for port 4000 TCP, the other for 5308 TCP, each from a separate source.

Port 4000 appears to be used by the remote administration utility Remote-Anything from TWD Industries (`http://www.twd-industries.com/en/index.htm`).

Port 5308 is associated with cfengine, a configuration and administration mechanism.[14]

[14] "Cfengine, or the configuration engine is an autonomous agent and a middle to high level policy language for building expert systems which administrate and configure large computer networks. Cfengine uses the idea of classes and a primitive intelligence to define and automate the configuration and maintenance of system state, for small to huge configurations. Cfengine is designed to be a part of a computer immune system, and can be thought of as a gaming agent. It is ideal for cluster management and has been adopted for use all over the world in small and huge organizations alike." according to `http://www.iu.hio.no/cfengine/`.

6 Enhancements and Extensions

7 Conclusions

A The bag script

This script is used to create counts of flows occuring in a raw data file. The ten level set creation usually reduces the size of the final bag to a manageable level.

```
#!/bin/bash
# take a raw data file and create 9 levels of set files followed by a
# bag file for the residue
# $1 is the file root (.rwf assumed)
# $2 is either s or d, depending on whether the bagging should be
# done on source or destination ips
rwsuperset --p --$2-s=$1-$2-1+.set --$2-d=stdout $1.rwf |\
rwsuperset --p --$2-s=$1-$2-2+.set --$2-d=stdout |\
rwsuperset --p --$2-s=$1-$2-3+.set --$2-d=stdout |\
rwsuperset --p --$2-s=$1-$2-4+.set --$2-d=stdout |\
rwsuperset --p --$2-s=$1-$2-5+.set --$2-d=stdout |\
rwsuperset --p --$2-s=$1-$2-6+.set --$2-d=stdout |\
rwsuperset --p --$2-s=$1-$2-7+.set --$2-d=stdout |\
rwsuperset --p --$2-s=$1-$2-8+.set --$2-d=stdout |\
rwsuperset --p --$2-s=$1-$2-9+.set --$2-d=stdout |\
rwsuperset --p --$2-s=$1-$2-10+.set --$2-d=$1-$2-10+.rwf \
           --$2-i=10 --$2-b=$1-$2-10+.bag
```