Using archived argus flow records to secure and troubleshoot your network

Peter Van Epp
Operations and Technical Support
Simon Fraser University
vanepp@sfu.ca

The first part of this paper is going to try to demonstrate by way of examples why a record of all traffic in and out of your network archived to disk is valuable for security and network troubleshooting. While there are a variety of ways of collecting this data (argus, netflow, and ipaudit being some of them) we will only look at argus because its what I happen to be running. Once we have seen the value of such a record we will look at some of the issues around performance on high speed (~1 gigabit) and very high speed (~10 gigs / OC192) networks since that is an area of interest to the folks here.

In an article in Usenix's Login: magazine some years ago [1] I argued that an archived record of all traffic across your net is excellent insurance for your network even if you only archived the data and didn't look at it (other than to verify it is being correctly collected) until you have a problem. Note that this should be your last line of defense and as or more importantly an audit trail rather than the only line of defense because it is reactive rather than proactive (i.e it records that something bad happened but doesn't do anything about it other than to record it). To protect data that needs to be secure you likely need a bunch more possibly proactive defenses to protect the data. However in the event that they fail (or are compromised) the audit trail tells you that it has happened and possibly provides some sense of the scope of the breach (such as how much data the attacker was able to access). Both of which are very valuable. For data that isn't confidential which is probably the bulk of your network traffic, this record and after the fact enforcement can and does contain your traffic charges to the commodity Internet and does perfectly well at protecting your network and the rest of the Internet from abuse both towards your network from the Internet and as importantly (at least on our network) from your network towards the Internet. It has been our opinion for years that our firewalls need to face inwards to protect the rest of the Internet from a small fraction of our user community (in the early days when we were the only Internet game in town, the same 200 or so accounts out of 22,000 were the source of all discovered problems, that works out to about .1% that we need to guard the rest of the Internet against).

I hope here to demonstrate that it is even better to collect, post process and act on the data on a daily basis to keep your network both secure and running correctly. Most of this (with the exception of the act part) of this process can be and has been automated. My installation collects, archives to disk (in two physically separated machines, but you don't need to be that paranoid if you don't want to) and post processes the data to generate a report (an example of which is included below) on the data for the previous 24 hours. The main report sorts the top 30 hosts by traffic volume in the last 24 hours and displays them (highest traffic first) along with the top 10 destination hosts (again sorted by high traffic) and then traffic by destination port to that host. This report can be scanned in a few minutes in the morning and will tell you about the health of your network. Normally (at

least on our networks) there will be the usual suspects, campus wide servers of various kinds that as long as they are within their usual traffic usage can be ignored (and could be but, aren't at this point, automatically ignored). If the volume is abnormally high the reason will likely be easily visible in the list of destination hosts. However the most interesting entries on the list are random hosts on your network that have suddenly made the top 30 list. Often enough this is an indication that the machine has been compromised and is being used as a file store / file transfer host by an attacker. An alternate reason is that there is a new peer to peer file transfer protocol out that your usual control mechanism isn't aware of yet, and finally (but relatively rarely on my network) it may be someone doing something high volume that actually qualifies under the Acceptable Use Policy. The easy differentiation between the traffic types is number of destination ports and destination port numbers (the P2P stuff that is port agile gets more difficult though). If the transfers are to many different machines, many of them in the cable modem / ADSL networks around the world it is suspicious as possibly P2P or possibly a compromised machine and is worth checking in to a little more carefully. Some of these are easy to dismiss as acceptable, a large transfer from a faculty member's office to a single offsite host I would usually ignore for instance. Some known problem areas such as public labs with student access doing P2P to cable modems all over, are easy to classify as probably worth inquiring about to make sure they meet the AUP. I would guess that about %90 of the time on our network they don't meet the AUP, I have hit very few legit P2P transfers although there are some. It is usually profitable to have a look at the raw argus logs for the host in question at a time a bit before the suspicious transfer starts. Sometimes you will see evidence of a compromise from offsite. Other times you will see the traffic just start on its own (which is likely an indication that someone on the machine started it intentionally).

I'll note that when the p2p folks start encrypting the data stream and or use standard ports such as 80 and 443 to try and hide the transfer I (and of course you, if you are smart) are still going to see blocks of high traffic going from a single local host to a wide variety of off site hosts which start suddenly one day as suspicious. It may well bypass our automatic traffic limiting device, but I much doubt that it will survive the IP being blocked from off campus access (or the network port or MAC address being blocked in the network for the extreme sport fanatic that has a desire to meet important people to try their marketing skills to get network access back) until an explanation of why the traffic was valid is forthcoming.

In addition to the traffic count stats there is a rudimentary port scan detector included, although it is very much a work in progress at this point. The most useful part of it so far is a check for local machines sending email to more than 50 unique hosts within any given hour. So far in every case where this has happened it has been as the result of a virus or trojan on a machine, which are apparently being sold. We see a connection usually from somewhere in Russia to establish the machine is still infected followed about 20 minutes later by a connection (usually from a compromised machine in the US) which spams for 20 to 40 minutes and then stops. Since this is often in the middle of the night, and rarely causes any external complaint argus finding and flagging them is the only thing that is getting the machines cleaned. There are a variety of control channels in use, sometimes an IRC server, sometimes a connection on port 113 (identd) sometimes a high random port number.

Unfortunately a compromised host our way that is doing high volume scanning out currently most often gets found because the morning traffic report has crashed and burned by running out of memory trying to account for the traffic to all the scanned hosts. While this works well enough, a scan of the raw argus logs generally identifies our infected host, and arranging to ignore it gets the reports running again, it is annoying. The solution in progress will be to first run the port scan detector and flag any of our hosts that are scanning and arrange to ignore small traffic volumes to the hosts they are scanning (which is what runs the script out of memory). The exciting part of course is going to be not ignoring a scanned host that is high volume for some other reason than it is being scanned or is scanning.

In addition the post processing is split in to phases to allow a high volume site to spread the processing load out. When the argus file cycles (every hour in my case as often as every couple of minutes at a high volume site) some post processing (looking for scans, pre calculating some traffic counts) occurs. Since on a single cheap machine this post processing can take longer than a collection interval (if the cycle interval is 2 minutes for a 100 meg file for instance), a pool of machines can spread both the archiving I/O load and the post processing load across a pool of inexpensive PCs to keep hardware cost somewhat reasonable (at high link speeds things become expensive no matter however, this only reduces that cost somewhat). I so far haven't needed this but other argus sites already do, as may we at some point in the future, so a little forethought and planning now may pay off later. We will return to this subject in the second half of this paper where we look at performance issues on high speed links.

Now having seen in general what we are doing lets get more specific about how our setup is configured and why, and what other options there are. Our argus setup consists of 3 machines which happen to be spread across 2 physical sites. For a low speed link (such as my ADSL line at home) a single small machine (a 16 meg clock speed and ram 386 running FreeBSD) storing the archive on the local disk does just fine. At some point controlled by link speed, I/O and memory bandwidth on the motherboard and CPU speed the disk I/O of writing the archive files to the disk start to use enough internal bus bandwidth that the Ethernet card(s) start to drop packets. When you hit that point (and my 35 meg link hits that point) you need to move to a dedicated machine for the argus sensor which only runs the argus daemon (preferably from a memory file system so there is no disk I/O even for update to steal bus bandwidth) and writes the output data stream to a socket. The next machine in line (via a cross over ethernet cable in my case, so network problems won't interrupt the link capture) uses the argus ra client to read the argus data stream from the sensor and archive it to disk. There is at least a 10 to 1 (and it may be much higher) reduction in the amount of data appearing on the monitored connection to what gets written out as the argus data stream. We will see later, this and some other features are very useful at very high link speeds. At the moment what is of interest is that because of the speed reduction the archiving machine has no particular problem with reading the argus data stream from the sensor machine and storing it to disk. I use a 160 gig disk on the intermediate archive machine so that the local archive can keep around 2 months of data as a safety backup (more because I can, than because its useful, several days to a week would be more than adequate, you likely want to be able to survive a problem on a weekend when you may not be looking for problems). This way if something happens to the post processing machine which

is the primary data archive or the network between them, the data will be saved on the archive machine until that is fixed and the data can be transferred and post processed. Once the data is archived on the archive machine, it gets transferred via scp over the network to the post processing host. There it gets saved in the online archive (400 gigs in my case about 5 to 6 months of data at the present rate) and moves from there off to tape backup in our backup robot where it is kept basically forever (although an increase in volume may change our retention policy later). As well this machine runs the post processing perl scripts which generate the daily reports and as noted above sometimes run the machine out of memory and cause the scripts to crash. This of course is no more than inconvenient (even if it managed to crash the entire machine, which it hasn't yet) because the data is still being faithfully archived by the archive machine at the sensor. Once the post processing machine is fixed up the data can be copied across to it and processed as normal. The post processing machine is also the place that I normally experiment on the scripts and/or scan the raw argus logs with no danger of impacting the capture and archiving of the data off the live network.

Now having seen how the data is collected, lets look at the output of a typical day's archiving and processing. First a complete day of archive files (the first half of which created the report below) from our commodity link which averages 35 megabits per second. This should give some sense of the storage needed for a given speed on an Internet link (I'd expect the data volume to scale somewhere close to linearly with link speed). It is however quite traffic dependent, our Westgrid link which is often over 500 megabits per second with peaks to 900+ megabits per second on a dedicated gigabit circuit of a multisite grid setup is much smaller than this, but is also only about 30 hosts running jumbo frames which generate many fewer records (with of course big internal counts) so your mileage may vary. I have modified the argus supplied archive script to allow for instance names because I process and archive more than one link and want to keep the traffic separated by link. I currently have three archives: one for commodity (the one below), one for CA*net4 and one for Westgrid as an experimental high speed test bed in case we ever get that high.

```
% pwd
/usr/local/argus/com_argus.archive/2005/06/13
% ls -l
total 1557424
-rw-r--r--  1 argus  argus   35545923 Jun 13 01:10 com_argus.2005.06.13.00.00.00.0.gz
-rw-r--r--  1 argus  argus   36558286 Jun 13 02:10 com_argus.2005.06.13.01.00.00.0.gz
-rw-r--r--  1 argus  argus   28902922 Jun 13 03:08 com_argus.2005.06.13.02.00.01.0.gz
-rw-r--r--  1 argus  argus   26556315 Jun 13 04:07 com_argus.2005.06.13.03.00.00.0.gz
-rw-r--r--  1 argus  argus   28848750 Jun 13 05:09 com_argus.2005.06.13.04.00.00.0.gz
-rw-r--r--  1 argus  argus   29253288 Jun 13 06:08 com_argus.2005.06.13.05.00.00.0.gz
-rw-r--r--  1 argus  argus   32840935 Jun 13 07:09 com_argus.2005.06.13.06.00.00.0.gz
-rw-r--r--  1 argus  argus   39232225 Jun 13 08:10 com_argus.2005.06.13.07.00.00.0.gz
-rw-r--r--  1 argus  argus   54435241 Jun 13 09:17 com_argus.2005.06.13.08.00.00.0.gz
-rw-r--r--  1 argus  argus   76061691 Jun 13 10:25 com_argus.2005.06.13.09.00.00.0.gz
-rw-r--r--  1 argus  argus   95854656 Jun 13 11:30 com_argus.2005.06.13.10.00.00.0.gz
-rw-r--r--  1 argus  argus  104160852 Jun 13 12:33 com_argus.2005.06.13.11.00.00.0.gz
-rw-r--r--  1 argus  argus  111222813 Jun 13 13:36 com_argus.2005.06.13.12.00.00.0.gz
-rw-r--r--  1 argus  argus  111448144 Jun 13 14:37 com_argus.2005.06.13.13.00.00.0.gz
-rw-r--r--  1 argus  argus  102526412 Jun 13 15:33 com_argus.2005.06.13.14.00.00.0.gz
-rw-r--r--  1 argus  argus  101504198 Jun 13 16:33 com_argus.2005.06.13.15.00.00.0.gz
```

```
-rw-r--r--  1 argus  argus   97979320 Jun 13 17:30 com_argus.2005.06.13.16.00.00.0.gz
-rw-r--r--  1 argus  argus   82283657 Jun 13 18:23 com_argus.2005.06.13.17.00.00.0.gz
-rw-r--r--  1 argus  argus   71820302 Jun 13 19:22 com_argus.2005.06.13.18.00.00.0.gz
-rw-r--r--  1 argus  argus   70560448 Jun 13 20:21 com_argus.2005.06.13.19.00.00.0.gz
-rw-r--r--  1 argus  argus   68249479 Jun 13 21:19 com_argus.2005.06.13.20.00.00.0.gz
-rw-r--r--  1 argus  argus   65421414 Jun 13 22:18 com_argus.2005.06.13.21.00.00.0.gz
-rw-r--r--  1 argus  argus   66636150 Jun 13 23:20 com_argus.2005.06.13.22.00.00.0.gz
-rw-r--r--  1 argus  argus   55889496 Jun 14 00:15 com_argus.2005.06.13.23.00.01.0.gz
```

An annotated and truncated (to save space) example of a traffic report file for a 24 hour period ending at 6 AM (so it is processed and waiting when I get in in the morning) created via cron by the perl scripts postprocessing argus logs on the post processing host. The scripts that generate this are available via anon ftp from

```
ftp.sfu.ca in /pub/unix/argus/argus.traffic.perl.tar.gz
```

although be warned they are quite ugly, still changing and without warrantee. The report below is quite butchered both to anonymize the data and because I have inserted editorial comments via the "(words)" lines below. I typically edit this with vi and do a search on "Tot" which jumps me to the total lines of each host. On a normal day with no trouble a 10 minute look through the file indicates that all is well. When a troublesome host appears there may be enough information in the detail section to dictate a response, or I may end up using the ra client to dump the raw argus log and probe deeper in to what is going on which may in turn cause a request for clarification of what is going on from the person using the IP in question.

In addition to the security aspects of this, argus is also capable of doing complete (or as I do selected hosts and address ranges) traffic accounting down to the port level if desired, of all the hosts on your network. In a number of cases we have users that are affiliated with SFU, but wish to run their own domain (in some case because they are startup companies from University research, or multi site research projects that wish to use a project domain rather than sfu.ca). In such cases their commodity traffic gets accounted for (because it attracts traffic volume charges) as a byproduct of the argus security scanning and if the volume is significant enough, they get charged for the bandwidth. This comes along essentially for free as a byproduct of looking for security problems in the data. You will see that noted in a few places in the report below (the billing data goes in to another file).

An example traffic summary report from a recent Sunday. Note that the by host traffic appears twice, once for the source host and once for the destination host so if you add them all up you would get twice the value of the traffic summary total (which only counts the traffic once). Doing this correctly ranks the busiest hosts on your network in terms of traffic volume which is something that I wish to know.

```
Traffic Summary From: Sun Jun 12  5:58:58 2005 To: Mon Jun 13  5:58:58 2005

     206,657,306,996 Total     145,806,723,952 Out      60,850,583,044 In
```

(in and out counts are adjusted to reflect out from our network and in to our network to give an indication of load on the network. Argus declares the source as the host that sends the syn without regard to which side of the

physical network the traffic is on.)

(external client, pays for bandwidth and thus is accounted for by argus)

```
aaa.bbb.cc.ddd      41,962,421,623 Tot    39,771,005,841 Out    2,191,415,782 In

   bbb.cc.ddd.ee             1,197,659,246         72,583,979     1,125,075,267
   bbb.cc.ddd.ee:50033         441,253,997         11,700,746       429,553,251

... (9 more hosts)
```

(www.sfu.ca, normally the #1 bandwidth user, except Sundays)

eee.ff.bbb.aa 31,236,470,372 Tot 30,085,681,542 Out 1,150,788,830 In

(local cable co's web proxy)

```
   cc.dd.eee.fff             1,297,979,324      1,259,691,372        38,287,952
   cc.dd.eee.fff:80          1,160,999,352      1,123,487,908        37,511,444
   cc.dd.eee.fff:443           136,979,630        136,203,356           776,274
   cc.dd.eee.fff:21                    342                108               234

... (9 more hosts)

eee.ff.aaa.a        10,327,873,548 Tot     7,626,300,731 Out    2,701,572,817 In
```

(SFU web mail server)

```
... (10 detail hosts)

eee.ff.ccc.dd        8,772,878,396 Tot     3,259,298,186 Out    5,513,580,210 In
```

(SFU library proxy server)

```
... (10 detail hosts)

eee.ff.aaa.bb        7,339,041,869 Tot       927,149,225 Out    6,411,892,644 In
```

(external mail server)

```
... (10 detail hosts)

eee.ff.aaa.cc        6,579,966,620 Tot     6,367,286,382 Out      212,680,238 In
```

(cgi web server, licensed software distribution server which is most of this traffic)

```
... (10 detail hosts)

eee.ff.aaa.dd        5,637,568,833 Tot     5,244,821,091 Out      392,747,742 In
```

(internal mail server)

```
... (10 detail hosts)
```

```
eee.ff.aaa.ee        4,916,276,147 Tot     1,680,363,646 Out    3,235,912,501 In
```

(general time share server)

```
... (10 detail hosts)

eee.ff.aaa.ff        4,052,547,078 Tot     3,657,692,982 Out      394,854,096 In
```

(SFU web portal)

```
... (10 detail hosts)

eee.ff.eee.ff        3,649,121,236 Tot     1,942,566,046 Out    1,706,555,190 In
```

(departmental web server)

```
... (10 detail hosts)

eee.ff.ggg.hh        3,428,989,162 Tot     3,311,345,002 Out      117,644,160 In
```

(departmental web server)

```
... (10 detail hosts)

eee.ff.iii.jj        3,368,546,694 Tot     3,298,324,986 Out       70,221,708 In
```

(demonstration traffic alert machine, not a server, running file sharing, after a few days, complaint to owner. Started Saturday in this case, dealt with on Monday)

```
   aa.86.101.181            1,391,068,293      1,373,987,241        17,081,052
   aa.86.101.181:1234       1,391,068,293      1,373,987,241        17,081,052

   aa.85.240.114              580,779,894        563,184,426        17,595,468
   aa.85.240.114:8000         580,779,894        563,184,426        17,595,468

   aa.87.106.118              264,644,874        259,217,846         5,427,028
   aa.87.106.118:8000         264,644,820        259,217,792         5,427,028
   aa.87.106.118:61907                54                 54                 0

   aa.80.170.122              118,948,790        115,119,693         3,829,097
   aa.80.170.122:8000         118,948,790        115,119,693         3,829,097

   bbb.95.39.184              102,453,410         99,853,024         2,600,386
   bbb.95.39.184:8000         102,346,300         99,760,726         2,585,574
   bbb.95.39.184:80               107,110             92,298            14,812

  ccc.121.230.70               95,583,668         93,861,359         1,722,309
  ccc.121.230.70:1234          95,581,176         93,859,836         1,721,340
  ccc.121.230.70:80                 2,492              1,523               969

   ddd.144.33.22               92,716,874         90,520,369         2,196,505
   ddd.144.33.22:8000          92,608,501         90,459,519         2,148,982
   ddd.144.33.22:80               108,373             60,850            47,523

     ee.68.229.6               91,377,936         89,049,094         2,328,842
     ee.68.229.6:8000          91,268,832         88,956,280         2,312,552
```

```
     ee.68.229.6:80                109,104             92,814            16,290

      ff.70.1.28                83,777,516         80,825,224         2,952,292
      ff.70.1.28:8000           83,569,794         80,642,565         2,927,229
      ff.70.1.28:80                207,722            182,659            25,063

  gg.183.123.109                76,282,669         74,937,473         1,345,196
  gg.183.123.109:1234           76,274,954         74,933,366         1,341,588
  gg.183.123.109:80                  7,715              4,107             3,608

  hhh.250.222.22                59,916,599         58,493,609         1,422,990
  hhh.250.222.22:1234           59,910,297         58,490,019         1,420,278
  hhh.250.222.22:80                  6,302              3,590             2,712



eee.ff.aaa.gg        2,916,248,616 Tot       467,369,585 Out    2,448,879,031 In
```

(SFU proxy server)

```
... (10 detail hosts)

aaa.bbb.cc.eee       2,886,902,688 Tot     2,815,622,690 Out       71,279,998 In
```

(external customer, traffic charged)

```
... (10 detail hosts)

aaa.bbb.cc.ff        2,813,977,499 Tot     2,291,693,799 Out      522,283,700 In
```

(external customer, traffic charged)

```
... (10 detail hosts)

eee.ff.bbb.gg        2,040,721,927 Tot     1,868,208,092 Out      172,513,835 In
```

(Caucus web server)

```
... (10 detail hosts)

eee.ff.ccc.dd        1,846,565,050 Tot       986,145,578 Out      860,419,472 In
```

(client machine running file sharing in the warn range)

```
... (10 detail hosts)

bbb.cc.ddd.ee        1,654,144,904 Tot       133,277,341 Out    1,520,867,563 In
```

(external host out on the net connecting to a collection of our hosts)

```
... (10 detail hosts)

fff.gg.hhh.i         1,627,185,047 Tot        95,338,709 Out    1,531,846,338 In
```

(external host out on the net)

```
... (10 detail hosts)
```

```
eee.ff.aaa.hh        1,604,405,412 Tot     1,228,336,032 Out      376,069,380 In
```

(Webct server)

```
... (10 detail hosts)

cc.dd.eee.fff        1,581,765,492 Tot     1,475,340,260 Out      106,425,232 In
```

(external host out on the net)

```
... (10 detail hosts)

ff.gg.hhh.181        1,394,660,278 Tot     1,377,425,329 Out       17,234,949 In
```

(cable proxy server)

```
... (10 detail hosts)

ii.jj.kkk.203        1,325,185,249 Tot     1,284,232,777 Out       40,952,472 In
```

(cable proxy server)

```
... (10 detail hosts)

ll.mm.nnn.204        1,260,377,195 Tot     1,154,876,720 Out      105,500,475 In
```

(cable proxy server)

```
... (10 detail hosts)

jjj.kk.lll.m         1,251,169,940 Tot       969,072,774 Out      282,097,166 In
```

(affiliated research project, their own address space / domain, traffic level monitored but not charged unless volume very high)

```
... (10 detail hosts)

eee.ff.bbb.87        1,191,565,802 Tot     1,130,915,628 Out       60,650,174 In

... (10 detail hosts)

eee.ff.ggg.h         1,150,255,995 Tot     1,103,605,105 Out       46,650,890 In
```

(departmental web server)

```
... (10 detail hosts)

eee.ff.ww.bad        1,141,962,121 Tot       403,824,967 Out      738,137,154 In
```

(wireless user file sharing)

```
    aa.80.118.27               93,630,648          1,718,526        91,912,122
    aa.80.118.27:2006          93,630,432          1,718,418        91,912,014
    aa.80.118.27:2296                 216                108               108
```

```
    bb.8.234.122              47,632,357         22,536,831        25,095,526
    bb.8.234.122:25573        47,630,311         22,534,785        25,095,526
    bb.8.234.122:17415             2,046              2,046                 0

   cc.140.27.158              43,680,569         18,330,765        25,349,804
   cc.140.27.158:25573        43,680,569         18,330,765        25,349,804

   dd.81.149.213              43,577,804         42,281,850         1,295,954
   dd.81.149.213:2520         43,577,642         42,281,796         1,295,846
   dd.81.149.213:1561                162                 54               108

... (6 detail hosts)


eee.ff.bbb.c         1,133,555,346 Tot       451,673,578 Out      681,881,768 In
```

(DNS server should usually look like this, a large number in the top position indicates a problem that needs looking in to ...)

```
   aaa.225.87.43              25,406,642         10,755,404        14,651,238
   aaa.225.87.43:53           25,406,642         10,755,404        14,651,238

     bbb.98.98.1              25,321,537         10,132,338        15,189,199
     bbb.98.98.1:53           25,321,537         10,132,338        15,189,199

   cc.93.117.162              23,721,275          9,641,166        14,080,109
   cc.93.117.162:53           23,721,275          9,641,166        14,080,109

... (7 detail hosts)


eee.ff.ggg.hh        1,065,623,058 Tot     1,024,001,449 Out       41,621,609 In
```

(departmental web server)

```
... (10 detail hosts)
```

Some 30 hosts later we come to the end of the morning traffic report. Now lets see how successful all this is in real life.

This is a selection of annotated and anonymized incident logs from our network.

External report of an IRC control bot running on a Unix box.

Although traffic in the argus logs looked normal, an account was found to be compromised and running an IRC client. When the account was disabled, the mail server in the department was DDOSed (may still be in fact) indicating the report was indeed correct. Argus wouldn't have found this because IRC clients commonly run on this and many other of our hosts and the traffic wasn't unusual.

DNS traffic doubled in the last 24 hours. A look at the argus logs (and then tcpdump) indicated a looping query from a remote host. Block at the border and notify the owner (not directly security, but performance impacting on a base service).

Spam complaint as a result of forged Received: header

```
> Received: from eee.ff.xxx.yyy by 65.87.aaa.bbb; Thu, 24 Jun 2004 20:54:29
> +0100
```

looks legit, except the source address isn't in use on our network and the connection doesn't show up in the argus logs (and the receiving host is a cable modem and thus likely the source of the spam trying to cover its tracks / fool automated reporting systems, which it will). We are pretty much the only ones that can say for sure what is going on here, mostly because we have the argus logs to verify someone on our net didn't steal an unused IP.

MSSQL breach, machine used to transfer movie files:

```
eee.ff.aaa.bbb  total traffic: 1,487,088,538
          eee.ff.aaa.bbb    195.128.cc.dd   1183              0      15,011,404

          eee.ff.aaa.bbb    195.128.cc.dd   1350              0      15,007,616

          eee.ff.aaa.bbb    195.128.cc.dd   1371              0      14,255,016

          eee.ff.aaa.bbb    195.128.cc.dd   1384              0      15,024,208
```

This is an interesting side effect of an old argus bug (this log is from an older version of argus too). These are actually an ftp where argus has inverted source and destination port number for (the real dest port is constant, but not 20 or 21). Whichever ftp server they use transfers in ~15 meg chunks per transfer which is an easily seen (and suspicious) pattern in the data in the morning traffic report (or at least used to be).

This suggests that a filter that selects for this (~ 15 meg chunks from multiple source ports to a single dest port) would be useful in finding compromised hosts in raw argus data. This was caught when the user machine suddenly started using over a gig per day in bandwidth (to many different hosts although only one is shown here). This bug is corrected (and the reverse port test hasn't been added yet) in the current reporting scripts and argus version.

DOS with forged source addresses from one of our nets.

This was thought to be a compromised NT machine which was removed from the network. As we will see in October, this was incorrect, a Sun on this same network is the culprit and they laid low for the next several months with a compromised machine (argus isn't perfect!). Had there been an argus sensor on this network segment (rather than a router hop away) I could have queried the argus log for the source MAC address and identified the machine.

Traffic alert bit torrent:

Not security related but rather traffic related (since this is on the

traffic charged commodity link) before the Packeteer recognized BitTorrent, which of course didn't help the person running this because argus understands the effect of BitTorrent just fine, just a little later in the consume cycle.

```
eee.ff.aaa.b    total traffic: 8,969,630,470.935
          12.151.aaa.bb     eee.ff.aaa.b   6881            173             185

          12.202.aaa.bb     eee.ff.aaa.b   6881            192               0

         12.214.aaa.bbb     eee.ff.aaa.b   6881          2,551       1,278,001

         12.217.aaa.bbb     eee.ff.aaa.b   6881            288               0
```

This is an X rated site that was streaming video in at some 7 megs per second according to the Packeteer. When the owner of the machine was queried it turned out that an admin had left a machine in a lab without setting a screen password on X and some helpful passing student had indicated this is a bad idea by accessing video on a porn site (undoubtably knowing what would happen).

While I could have found both sides of the conversation with the Packeteer, given the source address from the packeted, argus was an easier bet to find the destination.

This is also an example of argus ra client output rather than one of the post processed reports.

```
26 Jul 04 15:59:15   S       tcp    aaa.bb.cc.dd.1801   ->  209.120.ccc.bb.80
216      382       11664        552068      RST
26 Jul 04 15:59:17   S       tcp    aaa.bb.cc.dd.1803   ->  209.120.ccc.bb.80
487      872       26298        1260232     RST
26 Jul 04 15:59:15   S       tcp    aaa.bb.cc.dd.2138   ->  209.120.ccc.bb.80
196      347       10907        497150      RST
26 Jul 04 15:59:17           tcp    aaa.bb.cc.dd.1834   ->  209.120.ccc.bb.80
4        3         216          4542        RST
```

Sep 1 2004: Copyright violation. note the several day lag after the offense looking back in the argus log for Aug 27 verified that the accused host was in fact doing file sharing at this time.

```
Title:Norton Internet Security 2004
Infringement Source:eDonkey
Initial Infringement Timestamp:27 Aug 2004 18:29:05 GMT
Recent Infringement Timestamp:27 Aug 2004 18:29:05 GMT
```

Complaint from off site about a suspicious connection attempt

External email complaint about one of our hosts that showed up as trying to connect via ssh. Our argus logs make it look a lot more like a compromise (from the volume of data transferred). When the remote site looked more closely

after being advised of that they discovered that they had in fact been compromised and logged in to from a number of places in addition to us. Since I didn't see one of the characteristic ssh attack patterns from our host I don't think we were the source of the original breakin, but rather just another compromised host being used to log in:

```
06 Oct 04 07:56:51  *        tcp   ccc.dd.aa.bbb.4305   ->  aaa.bbb.ccc.dd.22
169      123       14049        26023       CON
06 Oct 04 07:57:52  *        tcp   ccc.dd.aa.bbb.4305   ->  aaa.bbb.ccc.dd.22
67       39        5234         4442        FIN
```

A look back at our host in the argus logs indicated that they had been running an IRC client for a number of days that apparently got hijacked an hour or so before the login to the machine above. As noted, that isn't enough suspicious or unusual activity to cause comment in and of itself without the external complaint.

Yahoo robot goes nuts

Our first case (or first detected case) of rabid search engine robots. A yahoo robot was sucking down the entire contents of a large MP3 repository on our site. The original alarm was again the Packeteer with a 25 megabit per second MP3 file transfer rate. Argus indicated this was a yahoo robot (from DNS) and the machine on our end was a colocate who pays for bandwidth. Rather than do anything I notified both sites of the activity. Never heard back from yahoo, but the machine owner said it appeared to be transferring down the entire contents (rather than just the file names to index) of their large store. They ended up having to block the netblock in their firewall to discourage it from doing this. Again not security related but certainly cost related (although not directly our cost this time, the machine owner was none the less grateful for the heads up before he would have gotten a largish bill).

DOS to machine in Russia (breach from 3 months back)

The attacker (or another one, since this machine was root kitted a few days before this) on the same machine that was using forged source addresses earlier in this list got careless this time and didn't forge the source addresses when DOSing someone else in Russia. They had also laid low in the months in between and used the host only for IRC which it was running anyway and thus staying below our radar:

```
xxx.yy.aaa.b         2,175,990,947 Tot     2,095,667,715 Out       80,323,232 In

     80.97.aa.bb            1,860,232,546      1,860,232,546                0
     80.97.aa.bb:58131            102,254            102,254                0
     80.97.aa.bb:51720            101,953            101,953                0
     80.97.aa.bb:13672            101,867            101,867                0
     80.97.aa.bb:31552            101,695            101,695                0
     80.97.aa.bb:61470            101,652            101,652                0
     80.97.aa.bb:30519            101,437            101,437                0
```

The large amount of transmit bytes with no receive bytes makes things very suspicious. When the Sun was carefully checked over they found it had been compromised and root kitted a couple of days previous to this probably via the IRC connection since there wasn't anything else suspicious in the argus logs at the time of the breakin.

Copyright violation

```
Title:  Shrek 2
Infringement Source:  eDonkey
Initial Infringement Timestamp:  21 Oct 2004 08:01:01 GMT
Recent Infringment Timestamp: 21 Oct 2004 08:01:01 GMT
Infringer Username:
Infringing Filename:  Shrek 2 - German - MVCD Scr By Magic Roots.mpg
Infringing Filesize:  726681604
```

Which turned out to be yet another compromise from someone on tdial.net in Germany:

Yep, the machine was compromised on Sunday morning. You will likely need to reinstall the entire machine to be safe because you can't know what all was changed (this person from Germany is a frequent attacker here and is quite good at it). The accesses from aa.bb.cc.ddd are an SFU person (at least an SFU person is reading email from there), the attack came from 217.85.aa.bb which is a dialup line in Germany. I'd guess the original compromise was some time before this breakin, because the attacker looks to have connected to a pre existing program and started a data transfer (probably movie files from past experience).

ask.com robot eating 6 gigs per day.

This one got away with it for several days, because it is a bioinformatics web machine that sometimes does large transfers offsite. I only looked at it more closely when the transfer went on day after day which is unusual and it again turned out to be rabid search engine robots (3 or 4 this time) downloading the entire site over and over. Again a block in the firewall was required to stop it (robots.txt files were ignored).

services9 virus comes through

One of your hosts looks to be being a botnet controller for the latest PC virus (services9.exe). Times are PST:

```
14 Mar 05 14:16:03          tcp  eee.ff.aaa.bbb.3224   ->   aaa.bb.ccc.dd.6667
1        2         80           135         CON
14 Mar 05 14:16:38  *       tcp  eee.ff.bbb.ccc.1915   ->   aaa.bb.ccc.dd.6667
3        4         214          270         CON
```

Of interest here (once the first two virused machines were IDed) is the IRC controller in use. Scanning the argus logs for this address (aaa.bb.ccc.dd) identified additional machines that were infected. The first controller was on

a .edu site who promptly took it down when advised, but the controller just moved (presumably via dynamic DNS) to a new host at a hosting company who didn't respond (they may or may not have done something about it). It took almost a week for the AV companies to begin detecting this (our folks figured out for them selves how to clean it and spread the info to the rest of our community until the AV kicked in). When a machine was thought to be disinfected, re enabling it and checking argus for connection attempts to IRC confirmed if the cleaning was successful or not (in a number of cases it hadn't been).

External complaint about machine scanning:

Machine scanning out for port 1433, a look back in the argus logs indicated that it had been compromised 5 months before this in the year before and apparently not used/needed except for low level probes for since then. This is not unusual for compromises our way. I assume they have so many machines it takes time for them to get around to them. When they do we normally detect the activity and whack them in short order.

anon ftp config goes wrong

This is mighty suspicious (and the start of the traffic ramp up) an ftp attempt from a dial up line in Greece, which later looks to have become successful. By the 10th it had risen from not making the top 30 traffic generators to being mid list (at about 4 gigs) with a steady progression over the last few days.

```
07 Apr 05 00:23:52  d        tcp    eee.ff.aa.bb.20     ->  144.139.aa.bbb.1198
5        3         1703         166         FIN
07 Apr 05 00:23:53  s        tcp  144.139.aa.bbb.1199   ->    eee.ff.aa.bb.21
17       19        1067         2261        CON
07 Apr 05 00:24:47           tcp   61.179.aa.bbb.33087  ?>    eee.ff.aa.bb.80
1        1         54           54          RST
```

a couple of days later discovered the admin had blocked ftp but not cleaned out the directory and traffic was still moving out via the web (which the continuing traffic spike brought to my attention and caused me to ask about having seen both ftp and http traffic from the site during the breach). After that it dropped off the top 30 list as it should have.

Knowing I was writing this paper, a nice virus writer arranged to infect one of our wireless users with a spamming trojan horse so you could see the spam detector in operation (and as a side effect find a problem for me, since the wireless user shouldn't have been able to infect backbone machines but managed to do so). While in reality this got caught because the machines were all scanning for port 135 and 445 and set of the scan detector which got them all whacked, if that had not happened they would have been caught the next morning because the morning reports include a .smtp file which indicates someone has been spamming:

```
-rw-r--r--  1 argus  argus     49070 Jun 22 06:27 Tue.c4_argus
-rw-r--r--  1 argus  argus   9493929 Jun 22 06:20 Tue.c4_argus.all.hosts
-rw-r--r--  1 argus  argus       927 Jun 22 06:27 Tue.c4_argus.scans
```

```
-rw-r--r--  1 argus  argus     69949 Jun 22 07:27 Tue.com_argus
-rw-r--r--  1 argus  argus  69859098 Jun 22 06:33 Tue.com_argus.all.hosts
-rw-r--r--  1 argus  argus     18637 Jun 22 07:28 Tue.com_argus.scans
-rw-r--r--  1 argus  argus       314 Jun 22 07:28 Tue.com_argus.smtp
```

Dumping the .smtp file we see that the first host (a wireless user) started spamming somewhere between 18:00 and 19:00 (because it needs to have hit 50 hosts before the alarm is tripped it may have started earlier but not hit its stride til the next hour). The next two hours see a couple more hosts doing the same thing:

```
% cat Tue.com_argus.smtp
com_argus.2005.06.21.19.00.00.0.gz eee.ff.aa.228 57
com_argus.2005.06.21.20.00.00.0.gz eee.ff.ccc.10 292
com_argus.2005.06.21.20.00.00.0.gz eee.ff.aa.228 87
com_argus.2005.06.21.21.00.00.0.gz eee.ff.ccc.10 225
com_argus.2005.06.21.21.00.00.0.gz eee.ff.aa.228 72
com_argus.2005.06.21.21.00.00.0.gz eee.ff.ab.234 75
```

Now is the time to use the argus ra client against the archive files to see whats happening. To make that easy the archive file names involved are in the .smtp report:

```
% ra -r /usr/local/argus/com_argus.archive/2005/06/21/com_argus.2005.06.21.19.00.00.0.gz -nn
host eee.ff.aa.bbb
21 Jun 05 18:58:10           tcp   eee.ff.aa.bbb.1035   ->  aaa.ttt.aaa.kkk.7000  6        6
427          324         RST
21 Jun 05 18:58:41  s        tcp   eee.ff.aa.bbb.1028   ->     hhh.ii.j.kk.1863  8        7
523          1559        FIN
...
21 Jun 05 19:00:48  *        tcp   eee.ff.aa.bbb.1046   ->    gg.hh.iii.jj.25    111      89
93106        6181        FIN
21 Jun 05 19:00:49  *        tcp   eee.ff.aa.bbb.1063   ->    dd.ee.fff.gg.hh    119      93
91785        6339        FIN
21 Jun 05 19:00:49  *        tcp   eee.ff.aa.bbb.1074   ->  iii.jjj.kk.lll.25    129      117
91888        6862        FIN
```

We see that this is a case where the trouble started earlier (which is often the case) so we need to go back an hour:

```
vanepp@r2d2% ra -r
/usr/local/argus/com_argus.archive/2005/06/21/com_argus.2005.06.21.18.00.00.0.gz -nn host
eee.ff.aa.bbb
21 Jun 05 18:48:57  d        tcp   eee.ff.aa.bbb.2123   ->     bb.cc.ddd.e.80    8        8
913          8325        FIN
...
21 Jun 05 18:54:17           tcp  lll.mmm.nn.ooo.3368   ->   eee.ff.aa.bbb.113   1        1
74           54          RST
21 Jun 05 18:54:28  s        tcp   eee.ff.aa.bbb.1987  <->    vv.uu.jjj.kk.25    3        0
186          0           TIM
21 Jun 05 18:54:05  *        tcp   eee.ff.aa.bbb.1028   ->     rrr.pp.q.ww.1863  93       149
6700         53803       CON
21 Jun 05 18:54:05  d        tcp   eee.ff.aa.bbb.1027   ->    xx.yy.zzz.aa.1863  7        6
524          579         FIN
21 Jun 05 18:54:09  *        tcp   eee.ff.aa.bbb.1035   ->  aaa.ttt.aaa.kkk.7000  12       16
973          5204        CON
...
```

I'm going to cheat a bit for space reasons here. The connection aaa.ttt.aaa.kkk.7000 repeats a bunch more times in the full data. Chances are good that this is the host/port controlling the trojan. Having this information it is now profitable to do this over the next few hours or days (and the last few days to see if there are any more that didn't get caught previously, one of the advantages of keeping historical files):

```
% ra -r /usr/local/argus/com_argus.archive/2005/06/21/com_argus.20006.21.18.00.00.0.gz -nn
host aaa.ttt.aaa.kkk
21 Jun 05 18:54:10           udp    eee.ff.bbb.d.53    <->  aaa.ttt.aaa.kkk.32768 1        1
244          97          CON
21 Jun 05 18:54:09  *        tcp   eee.ff.aa.bbb.1035   ->  aaa.ttt.aaa.kkk.7000  12       16
973          5204        CON
21 Jun 05 18:54:10           rtp  aaa.ttt.aaa.kkk.32768 <->    eee.ff.bbb.c.53    1        1
93           220         CON
21 Jun 05 18:55:10           tcp   eee.ff.aa.bbb.1035   ->  aaa.ttt.aaa.kkk.7000  6        6
447          324         CON
21 Jun 05 18:56:10           tcp   eee.ff.aa.bbb.1035   ->  aaa.ttt.aaa.kkk.7000  6        6
447          324         CON
21 Jun 05 18:57:10           tcp   eee.ff.aa.bbb.1035   ->  aaa.ttt.aaa.kkk.7000  7        6
521          324         CON
```

First hour is all the same machine. Two items of note here: argus records get written at connection close, so by default (i.e. here) they aren't in time order. There is a client to sort them if needed, but I rarely use it and just remember what I'm looking at. In this case the connection from the attacker on port 7000 started at 18:54:09 and is therefore really before the DNS lookup at 18:54:10 which comes before it in the listing (because the DNS transaction was short unlike the control connection). This is presumably the attacker seeing what its found, its unclear why, perhaps the attack modifies depending on what DNS says in some cases.

So suppress the first host and see who else has been caught:

```
% ra -r /usr/local/argus/com_argus.archive/2005/06/21/com_argus.20006.21.19.00.00.0.gz -nn
host aaa.ttt.aaa.kkk and not host eee.ff.aa.bbb
%
```

No one new next hour but three new players by the hour after that:

```
vanepp@r2d2% ra -r
/usr/local/argus/com_argus.archive/2005/06/21/com_argus.2005.06.21.20.00.00.0.gz -nn host
aaa.ttt.aaa.kkk and not host eee.ff.aa.bbb
21 Jun 05 20:03:02           udp  aaa.ttt.aaa.kkk.32768 <->    eee.ff.bbb.c.53    1        1
93           220         CON
21 Jun 05 20:03:01  *        tcp   eee.ff.aa.ccc.2652   ->  aaa.ttt.aaa.kkk.7000  13       15
1006         5372        CON
21 Jun 05 20:03:02           udp    eee.ff.bbb.d.53    <->  aaa.ttt.aaa.kkk.32768 3        3
564          252         CON
...
21 Jun 05 20:11:53           udp  aaa.ttt.aaa.kkk.32768 <->    eee.ff.bbb.c.53    1        1
97           244         CON
21 Jun 05 20:11:53           udp  aaa.ttt.aaa.kkk.32768 <->    eee.ff.bbb.c.53    1        1
93           220         CON
21 Jun 05 20:11:02           tcp   eee.ff.aa.ccc.2652   ->  aaa.ttt.aaa.kkk.7000  6        6
447          324         CON
```

```
...
21 Jun 05 20:21:58           udp  aaa.ttt.aaa.kkk.32768 <->    eee.ff.bbb.d.53    1        1
86           212         CON
21 Jun 05 20:21:02           tcp   eee.ff.aa.ccc.2652   ->  aaa.ttt.aaa.kkk.7000  6        6
447          324         CON
21 Jun 05 20:21:44           tcp   eee.ff.aa.ddd.1028   ->  aaa.ttt.aaa.kkk.7000  4        4
298          216         CON
21 Jun 05 20:21:57  *        tcp   eee.ff.ccc.dd.1907   ->  aaa.ttt.aaa.kkk.7000  13       13
1006         3775        CON
```

And another 4 hosts after that in later logs. Far more valuable from my point of view is that the instigator of this attack was a wireless user, and thus likely a laptop, infected at home and brought on campus. There is only one problem with this, the netbios ports are supposed to be blocked from the wireless networks on to the campus backbone and thus this machine shouldn't have been able to infect backbone hosts (but they clearly did get infected by the same virus, making the laptop the likely vector). So switch log files from argus beyond my border to the flow log from the wireless authentication system we use. This log is GMT which is why the different time stamp:

```
Jun 22 01:55:34 auth3.sfu.ca 172720520 0 00:02:2d:7c:xx:yy tcp eee.ff.aa.bbb:1398
eee.ff.kkk.jjj:445 eee.ff.aa.bbb:1398 eee.ff.kkk.jjj:445 168 88 userid
```

This selected entry indicates that the infected host managed to get a reply from a probe on port 445 to a backbone host. It turns out one of the access lists that is supposed to block such connections was missing and in fact allowing the wireless network to infect the backbone which it shouldn't have been able to do. Unfortunately the last time I did an analysis like this one the infected host was in fact a laptop that had been brought from home to an office and attached to the wired network (which isn't blockable for policy and AD reasons), and so I never noticed that one of the wireless networks was in fact not properly protected (we have 4 or 5 wireless subnets, so the correct block logs appear on most of the wireless networks, adding to the false sense of security). So writing this paper has increased the security of our network, and perhaps reading it will allow you to increase the security of yours.

One last cute argus trick: argus keeps two traffic counts, the data length (from the IP header) of all packets (which under reports total traffic volume slightly because of padding in less than minimum length packets), and application level data which for TCP is the data (minus headers and retransmitted packets) that is presented to the application after tcp reassembly and retransmissions. Which count gets reported is changed by a command line flag to the ra client. By default the data length is reported, adding a -A flag to the command reports the application data. While checking the accounting traffic counts from argus one day I had occasion to process the same archive records with and without the -A flag. Most of the hosts in the list show about a %10 increase from application to traffic counts due to the headers being counted in the traffic count. On one particular set of hosts (all belonging to the same customer) the increase is closer to %50. Argus flags TCP retransmissions in the status field, but both sets of hosts were occasionally flagged as retransmitting so I hadn't thought much about it. It turns out (when we got out the trusty sniffer) that this set of hosts for reasons unknown retransmit on almost every connection. While I would think that this would impact server performance, as far as I know the owner of the machine has not

done anything about fixing this. Of course that could be because I don't know what I would even start looking for to fix it (although configuration would be a good bet), because I don't know what you could do to cause it. Whatever it is they have done it consistently (which argues configuration of some kind) on several hosts and the hosts are similar to other machines on the same network segment that don't behave like this, so it appears to be host problem rather than a network problem. The bottom line is that argus can tell you all sorts of interesting things that you probably didn't know (and in some unfortunate cases, which you were happier when you didn't know) about your network.

Now having looked at why having a complete record of the traffic on your net is a good idea, lets change focus and look at why (if your connection is fast enough) collecting this data becomes expensive and hard. First let me express my prejudice against using netflow records from your router (which will certainly work and is quite popular) to collect this data. In my view I want our routers routing traffic first last and always. At high link speeds thats hard enough in and of itself. Adding the load of collecting netflow data can either compromise the routing or result in loss of netflow records at high volumes (when you may want them most to find out what is causing the high traffic) to give preference to routing. I believe that the correct way to do this is to let the router route, and add a network tap to the link and have a dedicated box or boxes running argus (or ipaudit) collect the flow data. That way the tap protects the network from the netflow collector and at most you can lose the netflow data not effect the network in any way. It also lets you do whatever it takes to insure that you don't lose any of that data if you don't want to and can afford to (which may well be the limiting factor).

One of several advantages that argus has is that it only needs the headers of the packet to produce the flow records. While you can choose to collect up to the first 512 bytes of data of a flow, if you don't do so, argus only looks at the packet headers, not the data, which may ease privacy concerns around collecting this data. The very first thing to do when considering collecting flow data is to consider the privacy implications and your site's policy around privacy. You likely want to consider the argus logs confidential records, and that may impact your retention policies of the data. You certainly want to have considered who is allowed access to the data in relation to your site's policy on privacy.

Only needing the first 128 bytes or so of a packet can be a big win in required I/O bandwidth at high link speeds (high starting, as noted in my case as slow as 35 megabits per second) becoming ever more critical on a fully loaded gig link and a major crisis requiring special hardware on a 10 gig link. An additional benefit with argus is that it can happily collect only one side of a full duplex connection (with another physical box collecting the other side of the connection of course, half a connection isn't a lot of use) and later merge the two data streams together to recreate an archive just as if there were two interface cards in a single box collecting both sides of the link at once. This is important (especially at 10 gig) because it cuts the I/O performance required of the capture box in half and may be the difference between being able to capture the data and losing some of it.

The real win against an IDS (at the cost of doing a lot less than an IDS of course) is that since argus is collecting everything that comes by

without attempting to select it at all (other than to classify it in to a flow), you aren't attempting to do a textual search at wire speed to see if this is something bad that you should be collecting. This is an enormous performance win at high link speeds in the amount of machine performance you need to keep up with the link, since searching, even with fancy algorithms such as Boyer-Moore, or better yet perfect hashes, still increases the required CPU power and memory required very fast the more rules you need to search for (and then it may miss things you would like to see because, they didn't trip a signature). The ideal case is of course use both or in a couple of cases I know of, all three of argus (for the audit record of anything that gets missed by the other two devices), and an IPS fronting an IDS to get the IDS alert rate down to something that can be managed by an affordable number of staff.

While I expect I'm stating the obvious to most of the folks here I will point out that bandwidth can also be limited by the memory and bus speeds of the host computer, device drivers, kernel configuration and undoubtably many other things. All of these things have to be fast enough before you can achieve anything even close to the rated speed of the network and making them so can be a difficult task.

On a gigabit link with a PCI bus (PCIX or one of the newer ones are better than this of course) you are getting close to the maximum bus bandwidth followed closely by them maximum memory bandwidth available (which is being shared with the CPU as is the bus bandwidth) just to keep up with a fully loaded full duplex link. You want your solution to be able to keep up with a fully loaded preferably small packet (the worst performance case) link because if your attacker is aware you have such a system on your link they may well use a zombie net to supply the maximum line rate of data to your net in the hope that their attack will be what the over loaded monitor host drops letting them get away with the attack. If you can afford a monitor system that can do full link speed, this isn't a defense that works against you (although the traffic volume may in any case make your link or systems unusable).

My argus capture box is a Tyan Thunder motherboard with dual 1.2 gigahertz Athelon processors and SysKonnect GigE fibre cards. It is now about 4 years old and long in the tooth (in fact probably obsolete for even full speed Gig as we will see). At the time they were new, the pair of them running netperf, out of memory, in to a cross over fibre cable could achieve about 1.6 gigabits per second full duplex. It is unclear whether the bottleneck was PCI bus bandwidth, memory or kernel / driver inefficiencies from older kernels, but it is likely one of those, because the SysKonnect cards could achieve around 995 megabits per second one way on that same hardware.

One thing this indicates is that you should get the performance tools (our HPC guys prefer netperf, but iperf and I'm sure others are also around) and test your chosen configuration's performance preferably before you buy it but that isn't always easy. All hardware isn't created equal and it most certainly can be and probably will be the limiting factor on performance.

I'm lucky enough to have our HPC folks be one of the nodes in the Westgrid [2] grid computing cluster. From my point of view this is wonderful because they have high speed expensive toys such as a 200 Terabyte file store that can fill a Gig E pipe out over CA*net4 and a bunch of folks with clue that know how to actually get that performance out of them. I'm lucky because

I'm not officially connected to Westgrid, but the file store sits in our machine room and the fibre off to BCnet and then CA*net4 is my responsibility and passes by (and since it was near, passes through) my multiport optical tap. I also have a gig capable sniffer that connects to that multiport tap. Thus when they manage to break things in interesting ways I get to help diagnose the problem without having any responsibility for doing anything about it. This is the best of all possible worlds as far as I'm concerned and since they get sniffer traces when there are problems both sides are happy. The relevance of this to the current discussion is two fold: first, this machine provides my benchmark for a machine that I should take seriously as being able to fill a gig pipe with useful work. It is something like 4 filled 19 inch racks, needing half a dozen 30 amp power circuits to power it and a good part of a multiton airconditioner to cool it. It can get pretty much a full gig pipe to disk or tape because I've seen it do it, and may be capable of even more than that if we have the data path to feed it (late data says an upgrade may be required to go to 10 gigE). The moral of this story is that you need a substantial machine (which probably isn't going to fit beside your desk because of power and cooling requirements) to get useful work done while keeping a gig link busy from a single machine. It also means that on a fast link you may need a substantial machine to successfully capture and archive the audit data even though argus manages to reduce the input line rate by 30 to 1 or perhaps even higher when it produces the records of the flows that you then need to get to disk.

The second thing is this machine (and Westgrid in general) provides an illuminating practical test bed for testing capture performance with argus. I deployed one of my sensor boxes with dual SysKonnect Gig cards off an optical tap in the Westgrid fibre. Then one of the Westgrid folks arranged a quick netperf test at ~950 megabits per second to an off site Westgrid host when the network was quiet. The argus sensor was not archiving to disk on the sensor but rather writing to another host via a socket. The interesting result of this test (which was with a stock FreeBSD kernel with the BPF buffer maxed) was that argus caught less than %50 of the traffic that actually passed over the link. The bpf packet loss counter was reporting a fair bit of loss, but I suspect (but didn't poke too far) that more of the loss was occurring at the device driver and possibly kernel buffer areas because the libpcap reported loss wasn't anywhere near the real loss. There are at least three places where packet loss in a libpcap application can take place, the first two are invisible to libpcap so you need to go poking in to the kernel log files and error counters when you discover packet loss. We then switched to a SUSE Linux 2.6 kernel tuned for HPC use that Westgrid uses on one of their clusters with about the same results. Luckily I was moaning about this on the argus developers list and someone that had been at SANE in Europe pointed me at the linux ring buffer code available from the author of ntop at www.ntop.org. This is a kernel mod that truncates the packet at the kernel level to whatever the slice length is, then uses mmap calls to bypass all the kernel buffering and kernel / userland copies and maps the page directly in to a modified libpcap stub in the user process. You can (and obviously should) boost the size of the ring buffer (although you may have to increase the compiled in kernel memory limit to get it big enough) I believe we started with 50 megs for the capture buffer. With that in place we appear to usually be able to keep up with a full speed netperf at ~950 megs per second using 9K jumbo frames (a close to ideal traffic mix for good performance though, small packets would likely perform much worse). If you are looking to monitor gig links, this is a fine place to

start since this is using the open source version of argus. I expect the correct thing to do (although I currently don't have another largish machine to do it with) is to split the sensor boxes in two. One gig card with the ring buffer code in each box capturing one half of the connection. With that in place I expect we could keep up with a full speed gig link even with my oldish hardware. We see, while not trivial nor cheap, it is possible to do a good job on a gigabit link with enough care and attention (and measurement!).

The exciting time comes when we are talking about a 10 gig link. I'll note that I don't have any direct experience here yet (we may have lit our first 10 gig link about the time this paper is presented if my luck holds) but argus's author also has a commercial version of argus in addition to the open source one I have been talking about. He reports that the commercial version, presumably using Endace DAG interface cards and I expect a very large processor argus can keep up with an OC192 / 10 GigE link. The DAG interface cards are of course what you want to be using for doing capture. The interface card has an onboard CPU, ram and a GPS synced time register. I expect that (because I would) they only store the slice length of the packet in ram, and when the packet is complete check the crc and store the good/bad status and enough low order bits from the time stamp register to recreate an accurate arrival time for the packet in the onboard ram. Once a suitable number of packets to fill a reasonable size buffer have been processed by the card, an interrupt to the host processor then DMAs the block of data to the host processor to be fed to the application via a pcap like interface.

So far the host processor hasn't been bothered by pesky things like interrupts for each packet which tend to chew a large amount of CPU cycles / memory bandwidth to deal with, thats all being done by the onboard processor. The GPS time stamp allows the correct arrival time of each packet to be calculated by adding the stored value to a base time stamp. This fixes the problem that GigE ethernet cards that interrupt once for multiple packets received (to try and make the same performance saving), which is that the actual arrival time for each packet is lost, since they will all get the time stamp at the time of the interrupt.

This is important because netperf at 1.6 gigabits per second on the athelons was using 1.5 CPUs doing nothing but netperf probably largely because the older SysKonnects interrupt for every received packet. As noted earlier argus is happy with only seeing on side of the conversation, so I expect if you deploy two expensive cards machines you can capture the 10 gig link at full speed and archive it to a host with lots of fast write performance raid disks (or perhaps even a memory file system) to create the archive. There is little question that dealing with 10 gig links is going to be exciting. I expect in the short term it may be much wiser to deploy 1 gig argus/IDS/IPS boxes on each gig link that aggregates in to a 10 gig link. This of course leaves the folks such as Westgrid who intend on putting in a 10 gig host adapter out in the cold (or possibly I should say in the very center of a very hot fire) and looking at the limited and expensive options of 10 gig monitors or not monitoring at all which may be safe enough on a closed network. There is a snort based IDS appliance that will do some 600 rules at up to 6.5 gigabits per second apparently available so I understand, although I expect it is pricy.

Hopefully this glimpse in to how we use argus has inspired you to setting up a trial host on any old hardware (even if it loses packets). Then

when you find it has become indispensable you can come back through and try out the various performance suggestions in the last part of the paper.

[1] http://www.usenix.org/publications/login/2001-11/pdfs/epp.pdf

[2] http://www/westgrid.ca

Below are a few interesting papers that I have come across on the net:

www.hep.man.ac.uk/u/rich/PFLDnet2003/Fast_wan_v5.pdf

www.hep.man.ac.uk/u/rich/net/nic/GE_FGCS_v18.doc

http://www.malmedal.net/Malmedal_Master_Thesis.pdf