;login:
THE MAGAZINE OF USENIX & SAGE
November 2001 • Volume 26 • Number 7

Special Focus Issue: Security
Guest Editor: Rik Farrow

inside:

USENIX & SAGE
The Advanced Computing Systems Association &
The System Administrators Guild

pssst, wanna buy some network insurance?

by Peter Van Epp

Peter has spent the past 13+ years as a network/security/system administrator (in that order of priority) for Simon Fraser University in B.C., Canada.

vanepp@sfu.ca

From the title, you probably have the image of a seedy guy holding open a raincoat full of insurance policies (and an arm full of watches), but that isn't quite what I mean.

Think what you could do if you had a record of all network transactions that crossed your link to the Internet in an amount of disk space you could afford (without being a major government). There is an open software tool called argus that provides exactly that option. In this article, we are going to explore some of the things you can do with such information and state some reasons why you might want to have it, even if you can't interpret the results immediately. Hopefully, your network will become a safer place (and thus so will the Internet as a whole). You can certainly make friends with your internal/external auditors because this can be used to audit the effectiveness of a security policy, intrusion detection system (IDS), or firewall.

Let's start the ball rolling with a true war story (and the first good use of this tool). As the reader of our abuse email account in April a year or two ago, I received an external complaint about someone our way doing a port scan of a site. We are a university site, so nothing unusual there – just another contestant in the "you bet your account" contest. What was unusual was that a review of the weekend logs (this being Monday and the scan being Saturday) didn't turn up any scan. Then I looked at the date on the supplied firewall log: it wasn't from the previous Saturday but a Saturday last February. No problem: crank up the old disk machine, run the log files from last February through

ra (one of the argus data reporting tools included in the argus distribution) and, sure enough, there was a port scan. Thus I was able to tell the person reporting the "problem" that we had dealt with it the Monday after it happened (by declaring a new winner in the "you bet your account" contest) and that perhaps they should check their firewall logs a little more frequently than every two months or so.

Several points to make here: first, you should check the date and timestamp on a complaint using the argus logs to verify that a reported attack from your site really originated with your site (and wasn't forged as coming from your site), and that the reporting site (or you, when converting from a remote time zone) hasn't gotten the time wrong. This is important for things like dialup connections, where a different time zone can cause someone to be blamed erroneously and a slip up can be embarrassing and possibly expensive. Next, because you are recording all traffic, you can monitor outbound traffic from your site, which may or may not be important to you. Around our place, the standard comment on firewalls is that "the firewall faces inward to protect the Internet from our user community," not necessarily the other way around. On a commercial site, where you can (at least in theory) trust your user base, this may be less of an issue. However, in the case of a breach, the log provides you with a record of who else the attacker may have attacked from your site. Since the argus server can also be invisible to the network and tightly secured, chances are good the attacker won't find it and delete his tracks, even if he does so on the machines he broke in to.

A Hypothetical Case

While we are on this topic (i.e., sites that may have machines that are less than secure in large numbers) let's look at a hypothetical situation (which so far hasn't happened to me, and I hope never does). One morning a local law enforcement agent appears with a warrant to seize one or more of our machines on the grounds that they have been used in a DDoS attack. What are you going to do in this case? My guess is you're going to say, "Yes, sir, here it is, sir," because you can't refute the allegation (although even being able to do so may well not help until later in the face of a warrant).

I have a 50/50 chance. I can check the argus logs and either refute that we were the source of the DoS attack (best case) or confirm that we were the source (toast!). Because in the first case, while the reply packets are appearing on my Internet connection, there are no corresponding source packets issuing from my net, which means the source is being spoofed by a site without anti-spoof filters on their border router. If the attack did come from our site, our risk manager can choose to settle out of court or argue due diligence since we would have (and, in real life, have before it got this far) detected and dealt with the problem in a timely manner because of the argus logs.

This also illustrates another argus feature. When the analysis of the DDoS clients first came out in December a few years ago and indicated the control mechanism was an ICMP echo reply packet, the author provided a scan program that I was able to run on a segment monitored by argus. I determined what the scan looked like (an ECR flag on the flow rather than ECO indicating an echo reply with no corresponding echo request). Looking back, I noticed scans using ECR packets against my network starting in September of that year (luckily without success) and continuing even as I type this article and undoubtedly still going on as you read it. This procedure can be generalized to most new exploits: determine what the exploit looks like in the argus log and scan back over the time you (and more importantly your firewall and/or IDS) didn't know

I can check the argus logs and either refute that we were the source of the DoS attack (best case) or confirm that we were the source (toast!)

A firewall would ignore this, as they were outgoing HTTP accesses on port 80.

you should be looking for the signature to detect compromised machines. Knowing of a break in after the fact, while undesirable, is much better than not knowing of the break in at all.

A news break: as I type this (on July 19) the IIS Code Red worm has broken out. This provides a case study in using argus for unconventional things. A CERT notification yesterday gave me my first infected machine. A look at the argus log indicates a signature of many, mostly 0 length, connection attempts to offsite Web servers. A quick Perl script that takes the argus ra data reporting tool output as input and selects accesses to port 80 that are not on any of my local nets is quickly written. The source and destination IP addresses get stored in an associative array indexed by source IP address and then sorted. As long as the source address is the same, increment a counter (because this is a new destination access from this same source host). Once the source address changes, store the source address in a new associative array indexed by the remote host count. When the entire file has been processed, sort the array of counts in reverse numeric order and write it to standard out.

This is about a page of Perl and an hour or so of work. The output looks like this (with the addresses obscured to protect the guilty):

```
100539
        1xx.yy.zzc.65

271
        1xx.yy.zze.6

269
        1xx.yy.zzf.161

...
```

The first address (and 13 more like it across both our campuses) is a machine affected with the Code Red worm scanning other machines. The other two hosts are normal accesses. This is a fine example of why having a record of all the data passing through your network is very useful. A firewall would ignore this, as they were outgoing HTTP accesses on port 80. There are reports that the initial Snort rule, because of a rule mistake, wasn't catching all of these. Since argus is recording everything that comes by, the mark one eyeball (after some thinking and data processing) has no problem picking the difficulty out of the noise in the raw data. A manual human scan of the argus log verifies that this really is the worm from the ra output itself (again with addresses obscured):

```
Thu 07/19 06:56:44 s  tcp    1xx.yy.zzc.65.60806   ->   aaa.165.233.142.80   3  0  0  0  REQ
Thu 07/19 06:56:44 s  tcp    1xx.yy.zzc.65.60791   ->   bb.147.29.238.80     3  0  0  0  REQ
Thu 07/19 06:56:44 s  tcp    1xx.yy.zzc.65.60813   ->   cc.123.5.126.80      3  0  0  0  REQ
Thu 07/19 06:56:44 s  tcp    1xx.yy.zzc.65.60768   ->   dd.60.30.131.80      3  0  0  0  REQ
```

. . . (lots more just like this – 100,535 of them in fact . . .) So whack this host (and the 14 others) off the network.

Since we mentioned your firewall and IDS a couple of paragraphs ago (before I was so usefully interrupted), and since I promised at the start to make your auditor smile upon you, let's look at some of the ways argus can be useful in conjunction with these devices. While we all know none of us would ever misconfigure our firewall, in the hypothetical situation where this did happen, how do you demonstrate that your firewall/IDS is doing what your security policy specifies? One good way is to put an argus

server on both sides of it and perhaps use a tool such as tcpreplay to insert known attacks into the input data stream.

When you compare the two argus logs you would expect to see the attack packets in the outside argus log. If you also find such packets in the log from the argus server inside the perimeter, you have just found a problem. Of course, you also have logs back in time to see if the hole has bitten you already. In such a situation, I often find the standard operating procedure for swatch syslog watcher to be useful: filter out (one class at a time) the expected packets from the log output and carefully examine that which remains until you have explained why it should be there or have identified a problem to be dealt with.

Another nice feature of argus is that, since it is not deciding anything at all about the data stream but is just recording it in a compact form, it is harder – and with sufficiently large hardware, perhaps impossible – to overload the argus server. Thus, should the attacker be able to overwhelm your IDS or firewall, argus may still give you a record of everything that happened on your net. With argus you at least have the data; with only an overwhelmed IDS or firewall you don't (or at least not all of it). Something to think about, especially in terms of insurance.

With argus you at least have the data; with only an overwhelmed IDS or firewall you don't (or at least not all of it).

Now let's look in detail at some argus-detected incidents to get a clearer idea of some of the ways in which argus logs are useful. In all cases below, the addresses have been obscured to protect both the innocent and the guilty. The data is being read from files captured by the argus daemon program writing its output to a file (it can also output to a socket in real time if desired) and then later processed with the ra reporting tool. While argus and the reporting tools both accept tcpdump-style filters, the argus daemon is running with no filters in place (i.e., capturing everything). Note as well that these logs are from older versions of argus, so they will look a little different than current version logs.

This listing was generated using the argus ra reporting tool, which outputs a human-readable interpretation of the argus data. As mentioned, it accepts tcpdump-type filters; this listing was generated with a command line like this:

```
ra -r argus.log.file -c -n host bbb.cc.dd.75
```

which displays all records containing host bbb.cc.dd.75 with packet and byte counts (-c flag) and without DNS lookups (-n flag) on stdout. We see the root break-in, along with a large amount of additional data (including a port scan, which is what attracted my attention in the first place), extracted below. First, an explanation of what we are looking at. The first field is a timestamp, following that (blank in this instance) is a flag field which indicates things like retransmissions in a flow and a variety of other items documented in the ra man page. Following that is the protocol type (IP only in these old logs, many more in current argus versions). Then the source IP and port (the port being confusingly separated with a "."). Note that this is the machine that argus believes started the flow, either because it saw the SYN-ACK handshake, or it is its best (and sometimes incorrect) guess if it didn't see the original SYN ACKs. Next there is a flow indicator which, in this case, indicates a bi-directional flow (again the possible values and meanings are documented in the ra man page). Following that is the destination IP and port number. Then come the source and destination packet and byte counts (added by the -c command-line flag).

In version 1.8 (which this log is from) the counts are application data count; in the current version 2.02, by default, the counts are total packet length to allow traffic calculation, although the old meaning is possible with a command-line flag. The last field on the line is the connection-status field.

```
Fri 12/10 21:54:18  udp  aaa.255.11.140.2344  <->  bbb.cc.dd.75.111     1   1  64   36    ACC
Fri 12/10 21:54:18  udp  aaa.255.11.140.2344  <->  bbb.cc.dd.75.32774   1   1  200  40    ACC
Fri 12/10 21:54:07  tcp  aaa.255.11.140.3225   ->  bbb.cc.dd.75.23      12  10 62   123   CLO
...
```

Now knowing what we are looking at, we see that the attacking host aaa.255.11.140 made a connection from port 2344 to attacked host bbb.cc.dd.75 port 111, which happens to be the port mapper. The 36-byte reply from host bbb.cc.dd.75 told the attacker that rpc.statd was running on port 32774 of the attacked host. The attacker then sent the buffer overflow exploit code to port 32774 on the attacked host and created a new root account (from post-breach analysis of the system rather than strictly this log), although the successful telnet connection in the third line is also a good indication of a problem. The compromised machine was then used to port scan other machines on the Net, looking for more victims, which is what caught our attention.

Now, let's look at a DDoS attack and an echo/echo reply flow (as opposed to an echo reply only flow) for controlling DDoS clients. To create this data stream the following ra command line was issued:

```
ra -r argus.log.file -c -n net ccc.dd.245.0 mask 255.255.255.224
```

which selects only the hosts belonging to the 32-host network that comprises this address space and demonstrates some of the flexibility available in filters with argus.

The first line below is a normal echo flow from a network management station. Note the flow is bi-directional (<-> flag), and the status flag of ECO indicates a perfectly normal echo request/reply flow from ping.

```
Tue 05/15 16:09:55    icmp  aaa.bb.184.131  <->  ccc.dd.245.2  3 3      ECO
```

Sometime later, we see a unidirectional flow (-> flag) from control machine eee.ff.126.2 (somewhere in France) to compromised machine ccc.dd.245.2 (previously compromised via an unpatched RPC program) flagged as "only an echo reply" by the ECR end flag. Unfortunately, because this is an argus 1.8 log, we only get packet counts on the ICMP, not packet and byte counts as with argus 2.x. However, from both the acknowledgment ECR packet sent back to the attacking machine and the following DDoS and/or port scan, we see this is an attack. The first indication of a problem in this case was the abnormally large size of the argus log file due to the number of DDoS packet flows that were recorded.

```
Tue 05/15 16:16:20    icmp  eee.ff.126.2   ->  ccc.dd.245.2  4 0      ECR
Tue 05/15 16:18:39    icmp  ccc.dd.245.2   ->  eee.ff.126.2  4 0      ECR
```

Below, we see the DDoS attack; despite the varying source addresses in the same subnet (and many more being caught by the anti-spoof filters in the router before making it this far), all of these packets are in fact from machine ccc.dd.245.2. If there weren't anti-spoof filters on this subnet, then the source addresses would have been from all over the Net, making this a very hard attack to trace back. With anti-spoof filters, the attacked site would know to whom to complain, although it wouldn't point to the compromised machine. It would direct me to the appropriate subnet on my network because they are all appropriately anti-spoof filtered, and of course, the argus log would do the rest.

```
Tue 05/15 16:18:44   tcp    ccc.dd.245.25.2009 ?>  ggg.hh.141.177.71   1 0   0   0   TIM
Tue 05/15 16:18:44   tcp    ccc.dd.245.9.1012  ?>  ggg.hh.141.177.72   1 0   0   0   TIM
Tue 05/15 16:18:44   tcp    ccc.dd.245.30.1822 ?>  ggg.hh.141.177.73   1 0   0   0   TIM
Tue 05/15 16:18:44   tcp    ccc.dd.245.27.1204 ?>  ggg.hh.141.177.78   1 0   0   0   TIM
. . .
```

Now that we have seen some of the things argus is good for, let's look at what's needed to run it. It's likely a lot cheaper than you think, and it is certainly less than the traffic charges for your link. Up to a moderately fast link (such as FastEther or an OC3), a reasonably large PC, such as an 800MHz P3 class with 256 or 512MB of RAM and a 40 or 80GB UDMA66 IDE drive, will do fine. For Gig links you probably want something in the class of a Sun 450, and of course larger/faster is better (and even then, 200 to 400 megabits per second of data is as much as I am aware of being successfully captured on either argus or commercial IDS systems at present, although there may be faster systems out there). Benchmarking with iozone and bonnie disk benchmarks indicate that the 7200 RPM UDMA66 IDE drives can get as much throughput as a 36GB SCSI drive, so use what you've got and test! Argus runs under UNIX, so any of Solaris, Linux or the BSDs will do. Note on the BSDs, at least on current versions, that there is a bpf bug relating to the select system call, and you need to apply a kernel patch and rebuild the kernel (a source for patches is listed in references at the end of this article). There is a test procedure in the patch README file, and I recommend using it to check your setup so that you know whether you are seeing everything there is to see.

On the low end of the scale, a 33MB 386 (yes a 386) with 16MB of RAM and a SMC WD8013 10 baseT NIC running FreeBSD can keep up to 3 or 4 megabits per second of traffic, i.e., more than enough for a cable or a DSL connection at home. This was measured with tcpreplay, which we are going to discuss in a while. That also means that if you aren't blessed with a high-speed link (or money), any old machine (a surplus 486 for instance) may be able to keep up with your link, making experimentation or production cheap.

Although the capital costs are reasonable, there are additional costs. Interpreting the output data is a high-skill occupation at present, and I suspect it is likely to remain so. However, before I lose half the audience, let me point out that as an insurance solution this doesn't have to be a show stopper. As we have seen, the capital costs for argus are modest. Disk is cheap and getting cheaper. That means that even if you lack the expertise to interpret the output data, it is possible (and in my view very prudent) to collect, verify you are collecting (this is important and we will discuss how below), and save the data from your network. Should the worst happen and you have an incident, you can hire a consultant and have him or her interpret the argus data for you. Ideally, you will never use it, and it will sit on the disk unused until you decide it's old enough that you don't care anymore and overwrite it.

If you choose to do this for insurance, you will, on a regular basis, want to display the output of the argus log files with the argus ra data display tool to verify that you are in fact collecting data. The idea is to make sure something is in fact being collected from your network so that you don't discover when a disaster hits that the disk filled months ago and the argus data is lost.

In addition to the regular testing, when you first install argus and any time you increase the speed of your link or change the hardware argus is running on, you'll also want to run a test to make sure that argus can keep up with the maximum speed of

The open source tool called tcpreplay will replay a tcpdump file, either in real time (according to the timestamps in the file) or at a selected rate or (best of all) as fast as it possibly can given your hardware.

your link. This same thing (and the same tools) are useful for testing your firewall and IDS system. If you haven't stressed them, how do you know how they will perform when you need them to? More importantly, if they are going to fail, how do they fail: by passing all traffic (deadly if this is your firewall) or by ignoring attacks (deadly if this is your IDS system)?

The open source tool called tcpreplay will replay a tcpdump file, either in real time (according to the timestamps in the file) or at a selected rate or (best of all) as fast as it possibly can given your hardware. Tcpreplay has been added to the FreeBSD ports collection (at least in the 4.3-STABLE branch if not yet the release branch), which means that you can install it by just typing `make` in the appropriate directory under FreeBSD. In addition I have available both the original source (because the home site of tcpreplay is currently being reworked and a new version being produced) and a modification that will take two tcpdump files and two NICs and output a full duplex data stream. There are performance issues. I was only able to get about 160 megabits per second on a 600MHz P3 machine with a single UDMA66 IDE disk, but two disks on separate controllers may boost that. More importantly, there's a timing issue (which I currently punt): the two streams are not synchronized so it is possible that a response will appear before the corresponding source packet is sent on the other channel. The solution to this would be a tcpdump file compiler that would rearrange and/or insert packets to ensure that at full speed, data would come out in the appropriate order. This is very similar to a RISC C compiler having to insert `no ops` in the instruction stream to maintain memory access timing. In this case, you need to either move a later packet or insert a padding packet to ensure that, when played at full speed, the response will appear after the request in the other data stream. It isn't rocket science, but it is work that I currently haven't had time to do. Again, if someone is interested in writing it before me I'd be happy to test.

This setup is what identified the bug in the `bpf` routines of the BSDs with the select system call during a test of argus. It's also what I used to determine that Solaris and (of all things) Linux are able to capture data at a full 100MB (Solaris cheated by using an E450 rather than a PC; I haven't run the test on Solaris 86 yet). Surprisingly, the BSDs (at least FreeBSD 4.2 and OpenBSD 2.7) lose a small percentage of the traffic at 100 (on the identical hardware that Linux was using). It is also the setup that verified that a 386 can keep up with 3 to 4MB of data without losing anything. Being able to repeat an exact sequence of packets (and to vary the speed of delivery) is a very powerful test tool, not only for argus but also for your firewall, IDS, sniffers, and network gear.

Deploying Argus

Now that you have implemented argus and verified its performance, let's look at some of the issues with deploying it. First, it is a juicy target. It contains lots of sensitive data about your network all in one convenient place, so you need to secure it heavily. It is also potentially a privacy issue, and you need to have appropriate approvals and policy around what will and won't be done with the data. Since argus by default is only looking at the headers, the issues are somewhat less problematic than with an IDS that is also looking at the data, but in either case the issues still need to be addressed before implementation proceeds.

Like your IDS system, it only needs read access to the network. That suggests that you should use a splitter such as the Shomiti Century tap (for 10/100 baseT) or an 80%/20% optical splitter for a fiber connection to isolate your systems (both argus and

an IDS). You should tighten down the UNIX box it runs on to run either nothing at all (which implies management access from the machine console and is likely too paranoid a reaction in many cases) or with only a well-patched SSHD running on a private network deep in the well-protected heart of your network. This is likely the configuration to run. It allows you to move the captured data (via SCP for instance) to another heavily secured (and large-memory) machine where you want to run the analysis programs. This prevents the analysis process from stealing resources from the capture process, which in turn may result in packet loss during capture. If your link speed is not that high and you have a fairly large machine, this may not be an issue, but it is something to keep in mind in any case. Remember to secure the backup tapes (assuming you are archiving to tape) as well; the data on them is just as sensitive as the data on the disk (and more often overlooked when security is being considered).

If you are on a busy/fast network, you probably need to increase the bpf buffer size in your kernel (the default is usually 8,000 or so with a max of 32,000, at least on the BSDs). Boost it to the full 32,000 and larger, if possible, to give yourself the best chance you can of capturing all the packets. This is where the "packets lost by kernel" message in argus (and other libpcap-using programs) comes from. Note that even if this number is 0 it is still possible that your network interface is silently dropping packets at a point before this counter in the bpf routines, so you need to use a tool such as tcpreplay to send a known data stream at the maximum rate to determine that your installation can keep up with the maximum data stream.

Once you have done all of these interesting (and time-consuming) things, and argus is happily collecting data, what kinds of things do you want to look for (and, better yet, write tools to look for automatically) in the data?

The prime one comes under the label of "history." As you get more and more logs from your network, you can also get a baseline of what is normal for your network. New patterns of access (in or out) are of interest. If there is suddenly a connection to a machine that hasn't occurred before, either someone new has been granted access to the machine or there is a problem. Politely asking the owner of the machine (or alerting the owner to the change automatically) can catch a problem early.

A connection in from the outside to a machine followed by a connection to a machine on the outside is a common cracker trick for laundering connections to avoid detection. Such a pattern (which again a firewall isn't likely to flag since it looks legit) deserves to be asked about to make sure it is authorized. Again, more of concern on a university campus, port scans both from the outside world heading in and from the inside against external hosts are of interest (and usually automatic first prize in the "you bet your account" contest if done from my site). With experience, you begin to be able to pick out the tool being used to scan you from the pattern of data in the logs.

One interesting feature of an argus log is the ability to detect slow port scans. I've seen some where that one probe is done around every 40 minutes for weeks. I say they are of limited interest because the event of interest isn't the scan, but any machine that is compromised, which will typically show up as a much more immediate log entry.

One use of scan (including slow scan) detectors is to populate a "suspicious IPs" file. This consists of IP addresses that have scanned or attempted compromises against your network. Future connections from such a site are "interesting"; they may also be suspicious, but they are certainly worth paying attention to because they may be coming back to exploit a hole that they have found on an earlier scan, or they may be a dif-

As you get more and more logs from your network, you can also get a baseline of what is normal for your network.

Since argus captures the number of bytes transferred in both directions and the TCP flags, it is possible to see (and thus to automatically filter out) connections that failed, reducing the amount you need to consider suspicious.

ferent customer of an ISP or cable company (with DHCP) doing something perfectly legit such as accessing a public Web page. As long as you remember that and don't overreact, it's fine to be suspicious of such an access, and, sometimes it will net you a cracker, a compromised machine, or both.

Since argus captures the number of bytes transferred in both directions and the TCP flags, it is possible to see (and thus to automatically filter out) connections that failed, reducing the amount you need to consider suspicious. Again, because argus records the byte and packet counts of all connections by post-processing them, you can acquire a complete traffic record (down to the source / destination IP / port level, if desired) in your network. This is interesting, as mentioned above, because of patterns and history. If you process traffic counts by machine and sort them (usually reverse order is the most interesting) and keep history of previous "normal" (for some value of normal) traffic patterns, the result of a compromise will leap out at you (along, unfortunately, with a number of false positives). If a machine that has previously done almost no traffic off-site suddenly is transferring large amounts of data to machines all over the net, you can be reasonably sure you have a compromised machine running a warez site or someone (perhaps the authorized user) running one of the distributed file-sharing programs.

I find that a query along the lines of "What has this to do with university business?" or "What account number should the bandwidth charges be charged to?" will either cause an abrupt halt or provide a valid explaination (more usually the former than the latter). Basically, what I'm saying here is that for external compromises (of any kind) to be useful, the attacker has to change the traffic profile of the machine (otherwise, it is of no use to them). When they do that and you have a complete log of the traffic, it becomes detectable and stoppable with limited damage. There isn't any easy way around this, other than, of course, finding a less protected system somewhere else.

This is another great advantage of a complete connection history: you can step back in time and see if you have been compromised by a newly discovered (on the white-hat side of the fence) exploit before you knew it was an exploit. Again, while it would be better to find this before the compromise (and it may show up in the changed traffic pattern before the compromise itself surfaces), better late than never (or when the lawyer and/or cops are pounding on the door). Your complete log can also absolve you of attacks with forged source addresses (your forged source address!) since the reply packets from the attacked host will show up in the argus log, but there will be no corresponding attack packets in the log from your site.

There are a couple more classes of traffic, such as a large amount of apparently purposeless traffic to or from a site. Unfortunately, on a university campus this is often hard to differentiate from legit research traffic which often seems to have no purpose. I remember what turned out to be a DoS attack directed at us coming from a university machine with ldap in its hostname on port 500 (X.500), which could plausibly be someone doing an X.500 update across the net as research. This may be a DoS or DDoS attack. We have seen a number of these involving the DNS, queries with (I assume) a forged source address for a largish DNS record repeated over and over.

I've thought about (but not implemented) an authenticated Web page that could be accessed by the sysadmin of a machine or network which would display the history data of connections and possibly traffic patterns to and from their machines. There are privacy issues here that would need to be carefully considered and debated within the

user community (and it may well need to be opt in), but it may make a very powerful tool. If this interests you, feel free to join the argus developers list and implement.

Finally, what challenges does the future hold? The main one is the increase in link speed (without a corresponding increase in memory, disk, and CPU speed). That will make it much more difficult in the future to be able to keep up with the higher speed links and will make life very, very challenging. One solution is to divide the link traffic up among IDS/firewall/argus boxes using a demultiplexor. The problem is that a skillful attacker who is aware of that can arrange to flood enough traffic at your box to possibly slip an attack by you. It is possible to keep up with a 100baseT/OC3 Internet link (that's what I have now). When my link speed goes to gigE, and then possibly 10GigE (since the underlying backbone is going to OC192), how do we monitor it (or more correctly, who has that somewhat used Cray?). We most certainly will be living in interesting times.

REFERENCES

Argus: *http://www.qosient.com/argus*

tcpreplay: *http://www.anzen.com/research/nidsbench* (not available at the moment so also at *ftp.sfu.ca /pub/unix/tcpreplay)*

bpf patches: *ftp.sfu.ca /pub/unix/argus*

Century tap: *http://www.shomiti.com/*

optical splitters: *http://www.netoptics.com/*