ISSN 2229-5518

USENET: InternetNews Software, Security – Needs and Goals

Monika Saxena, Praneet Saurabh, Bhupendra Verma

Abstract — Usenet is a distributed bulletin board system, built as a logical network on top of other networks and connections. By design, messages resemble standard Internet electronic mail messages as defined in RFC822. The Usenet message format is described in RFC1036. This defines some additional headers. It also limits the values of some of the standard headers as well as giving some of them special semantics. Newsgroups are the classification system of Usenet. The required Newsgroups header specifies where a message, or article, should be filed upon reception. In addition to InterNetNews, there are two major Usenet packages available for UNIX sites. All three shares several common implementation details. USENET at first was built with effectively no security. There was limited auditing even to detect abuse, let alone prevent it. Over time abusers came, and this meant in many cases that "privileged" functions had to be in some places either shut down or "put on manual" at great administrative cost to admins. In some cases, actual security using digital signature was applied, for newgroup messages (pgpverify), moderated groups (pgpmoose) and NoCem. PGP was commonly used because it is a widely distributed standalone program capable of doing digital signature.

Keywords – Usenet, bulletin board, internet news software, newsgroups, internetnews architecture, usenet security

1 Introduction

Usenet is a distributed bulletin board system, built as a logical network on top of other networks and connections. By design, messages resemble standard Internet electronic mail messages as defined in RFC822 [Crocker82]. The Usenet message format is described in RFC1036 . This defines some additional headers. It also limits the values of some of the standard headers as well as giving some of them special semantics. Newsgroups are the classification system of Usenet. [Adams87] The required Newsgroups header specifies where a message, or article, should be filed upon reception. Sites are free to carry whatever [transliteral] groups they want. Most sites carry the core set of so –called ``mainstream' groups.There are currently about 730 of these groups, and one or two new ones is created every week. Messages generated at a site are sent to the site's ``neighbors'' who process them and relay them to their neighbors, and so on. Sites can be interconnected -- indeed, on the Internet, this is quite common. See Figure

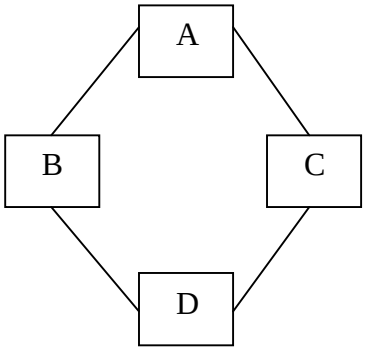

Figure 1: Small Usenet topology (all links are two -way).

The Path header is used to prevent message loops. For example, an article written at A could get sent to B , D , C , and then back to A. Before propagating an article, a site pretends its own name to the Path header. Before propagating an article to a site, the receiving host checks to make sure that the site that would receive the article does not appear in the Path line. For example, when the article arrived at site C , the Path would contain A!B!D , so site C would know not to send the article to A. Sites also keep a record of the Message -ID's of all articles they currently have. If D receives an article from B , it will reject the article if C offers it later. For self -protection, most sites keep a record of recent articles that they no longer have. This is very useful when another site dumps a (usually quite large) batch of old news back out to Usenet. For the past few years, the amount of data generated by Usenet sites has been doubling every year. A site that receives all the mainstream groups is receiving over 17 megabytes a day spread out over 11,000 articles . [Adams92] About 20% of the data is article headers, and while all of them must be scanned only half of it is must be processed by the Usenet software. The number of sites participating in Usenet has been growing almost as quickly. Based on articles his site receives and survey data sent in by participating sites, Brian Reid estimates that there are 36,000 sites with 1.4 million participants [Reid91] . A ``sendsys'' message to the ``inet'' distribution in June of 1989 received about 200 replies in the first twenty -four hours. A year later, nearly 700 replies were received. (Sendsys is a special article that asks all receiving sites to send back an email message, usually without human intervention; by convention, inet is

primarily the set of sites on the Internet.) [Adams87] The NNTP protocol is defined in Internet RFC 977 [Kantor86] published in February, 1986. This was

accompanied by the general public release of a reference implementation, also called ``nntp.'' This has been the only NNTP implementation that is generally available to UNIX sites. [Adams87]

Usenet Software

In addition to Internet News, there are two major Usenet packages available for UNIX sites. All three share several common implementation details. A newsgroup name such as comp.foo is mapped to a directory comp/foo within a global spool directory. An article posted to a group is assigned a unique increasing number based on a file called the *active* file. If an article is posted to multiple groups, links are used so that only one copy of the data is kept. A *sys* file contains patterns describing what newsgroups the site wishes to receive, and how articles should be propagated. In most cases, this means that a record of the article is written to a ``batchfile'' that is processed off-line to do the actual sending. The first Usenet package is called B News, also known as B2.11. The B news model is very simple: the program *rnews* is run to process each incoming article. Locking is used to make sure that only one *rnews* process tries to update the active file and history database. At one site that received over 15,000 articles per day, the locking would often fail so that 10 to 100 duplicates were not uncommon. Because each article is handled by a separate process, it is impossible to pre-calculate or cache any useful data. More importantly, file I/O had become a major bottleneck. A site that feeds 10 other sites does over 150,000 open/append/close operations on its batchfiles. It is generally agreed that B news cannot keep up with current Usenet volume; it is no longer being maintained, and its author has said more then once that the software should be considered ``dead.'' C News gets much better performance then B news by processing articles in batches [Collyer87] . The *relaynews* program is run several times a day to process all the articles that have been received since the last run. Since only one *relaynews* program is running, it is not necessary to do fine-grain locking of any of the supporting data files. More importantly, it can keep the entire active and sys file in memory. It can also use buffered I/O on its batchfiles, reducing the amount of system calls by one or two orders of magnitude. An alpha version of C News was released in October, 1987. Within four years it surpassed B news in popularity, and there are now more sites running C News then ever ran B news. From the beginning, the NNTP reference implementation was layered on top of the existing Usenet software: an article received from a remote NNTP peer was written to a temporary file and the appropriate *rnews* or *relaynews* program processed it. In order to avoid processing an article the system already has, it first does a lookup on the history database to see if the article exists. It soon became apparent that invoking *relaynews* for every article lost all of C News's speed gain, so the NNTP package was changed to write a set of articles into a batch, and offer the batch to *relaynews*. When articles arrive faster then *relaynews* can process them, they must be spooled. If two sites (B and C in the previous examples) both offer a third site (D) the same article at the ``same time'' then an extra copy will be spooled, only to be rejected when it is processed, wasting disk space; this problem multiplies as the number of incoming sites increases.To alleviate this problem, most sites run Paul Vixie's *msgidd* , a daemon that keeps a memory -resident list of article Message -ID's offered within the last 24 hours. The NNTP server is modified so that it tells this daemon all of the articles that it is handing to Usenet and queries the daemon before telling the remote site that it needs the article. This is not a perfect solution -- if the first, spooled, copy of the article is lost or corrupted, the site will likely never be offered the article after the *msgidd* cache entry has expired. Going further, *msgidd* is work -around for a problem inherent in the current software architecture.

Other problems, while not as severe, lead to the conclusion that a new implementation of Usenet is needed for Internet sites. For example:

- Since all articles are spooled, *relaynews* cannot tell the NNTP server the ultimate disposition of the article, and the server cannot tell its peer at the other end of the wire. This hides transmission problems. For example, a site tracing the communication has no way of finding out an article was rejected because the remote site does not receive that particular set of newsgroups.

- The NNTP reference implementation is showing signs of age. Maintaining the server is becoming a maintenance nightmare; over one-tenth of its 6,800 lines are *#ifdef* related.

- All articles are written to disk at extra time. Disks are getting bigger, but not faster, while CPU's, memory, and networks are.

InterNetNews architecture

There are four key programs in the InterNetNews package :

- *Innd* is the principal news server for incoming newsfeeds;
- *innxmit* reads a file identifying articles and offers them to another site;
- *ctlinnd* sends control commands to *innd* ;
- *nnrpd* is an NNTP server oriented for newsreaders.

Of these programs, the most important is *innd*. We first mention enough of its architecture to give a context for the other programs, and then discuss its design and features in more detail at the end of this section.

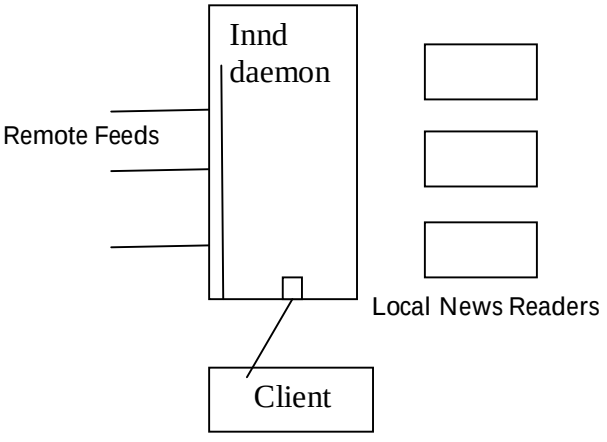

Figure 2 : Innd architecture

Innd is a single daemon that receives all incoming articles, files them, and queues them up for propagation. It waits for connections on the NNTP port. When a connection is received, a *getpeername* (2) is done. If the host is not in an access file, then an *nnrpd* process is spawned with the connection tied to its standard input and output. (Unlike other implementations, no single INN program implements the entire NNTP protocol.) It is worth noting that *nnrpd* is only about 3,500 lines of code, and 20% of them are for the ``POST'' command, used to verify the headers in a user's article. *Nnrpd* provides all NNTP commands except for ``IHAVE'' and an incomplete version of ``NEWNEWS''. On the other hand, it does provide extensions for pattern -matching on an article header and listing exactly what articles are in a group. The NNTP protocol seems to be a good example of the UNIX philosophy: it is small, general, and powerful and can be implemented in a very small program. Articles are usually forwarded by having *innd* record the article in a ``batchfile'' which is processed by another program. For Internet sites, the *innxmit* program is used to offer articles to the host specified on its command line. The input to *innxmit* is a set of lines containing a pathname to the article and its Message -ID. Since the Message -ID is in the batchfile, *innxmit* does not have to open the article and scan it before offering the article to the remote site. This can give significant savings if the remote site already has a percentage of the articles. Until recently, *innxmit* used *writev* to send its data to the remote host. At start -up it filled a three - element *struct* iovec array with the following elements:

```
[0] { ".", 1 };
[1] { placeholder };
[2] { "\r\n" }
```

To write a line, the *placeholder* was filled in with a pointer to the buffer, and its length, and a single *writev* was done, starting from either element zero or one. While this implementation was clever, and simpler then what was done elsewhere, it was not very fast. *Innxmit* now uses a 16k buffer and only does a *write* when the next line would not fit. This is also consistent with ideas used throughout the rest of INN: use the *read* and *write* system calls, referencing the data out of large buffers while avoiding the copying commonly done by the standard I/O library. The *ctlinnd* program is used to tell the *innd* server to perform special tasks. It does this by communicating over a UNIX -domain datagram socket. The socket is behind a firewall directory that is mode 770, so that only members of the news administration group can send messages to it. It is a very small program that parses the first parameter in its command line and converts it to an internal command identifier. This identifier and the remaining parameters are sent to *innd* which processes the command, and sends back a formatted reply. For example, the following commands stops the server from accepting any new connections, adds a newsgroup, and then tells it to recompute the list of hosts that are authorized newsfeeds: ctlinnd pause "Clearing out log files" ctlinnd newgroup comp.sources.unix m vixie@pa.dec.com

```
ctlinnd reload newsfeeds "Added OSF feed"
ctlinnd go ""
```

The text arguments are sent to *syslog* (8) for audit purposes.

The most commonly -used *ctlinnd* command is ``flush.'' This directs the server to close the batchfile that is open for a site, and is typically used as follows:

```
mv batchfile batchfile.work
ctlinnd flush sitename
innxmit sitename batchfile.work
```

The flush command points out another difference between INN and other Usenet software. The B News *inews* program needed no external locking. Files were opened and closed for a very short window, the time needed to process one article. The C News *relaynews* could be running for a longer period of time. The only way to get access to a batchfile is to either lock the entire news system, which is overkill for the desired task, or to rename the file and wait until the original name shows up again. The INN approach is more efficient and conceptually cleaner.

USENET SECURITY -- NEEDS

USENET at first was built with effectively no security. Anybody, anywhere could introduce any article which could do anything. There was limited auditing even to detect abuse, let alone prevent it. Over time abusers came, and this meant in many cases that "privileged" functions had to be in some places either shut down or "put on manual" at great administrative cost to admins. In some cases, actual security using digital signature was applied, for newgroup messages (pgpverify), moderated groups (pgpmoose) and NoCem. PGP was commonly used because it is a widely distributed standalone program capable of doing digital signature.

Authority on USENET

USENET has no government. It is an anarchy -- the absence of government -- but this does not mean total chaos. It has rules, guidelines, traditions, movements and principles of governance, if not government. USENET is, in spite of its public nature, a privately owned network. It is a cooperative, owned by the owners of the sites that are on it. Nobody gets on the network or uses it without the permission, directly or indirectly of these owners. In these site owners lies all the authority on USENET. This makes sense, as anything on USENET involves storing or changing data on the site owner's machines. Those files are theirs. Of course, having each individual site owner privately administer all aspects of the net on their machine would never work. There are over a few hundred thousand machines on the net, serving millions of users. So means to delegate administration have been found. As noted, the first way to delegate it was to simply let anybody do anything. In fact, at first anybody could create a new newsgroup just by typing a new name. In the past there were not many malicious users, so the system worked.

Today we have malicious users. Both spammers and the like who abuse for imagined gains, and plain sociopaths like trollers and crackers who abuse the net or people on it for the sake of abusing it. Barring malice, in the past we still had politics -- different groups wanting different things. To solve this various anarchic and pseudo-democratic systems evolved to develop group consensus or a measurement of group will, and everybody agreed, without force, to go along with the group will where it was important. One example was the newsgroup voting system. This works because in fact to get anything done in a co-op like USENET, you need the almost unanimous consent of the site owners. Any site owner is free to not participate in any group, hierarchy or other activity. So you must keep them all happy if you want to do something netwide. [Postel82] While total unanimity is hard, near-unanimity, won through compromise, has actually worked better than might be expected. This is true in part because almost all of us are drilled from childhood to accept the democratic principle and accept things the majority wants so that we can get our way later when we agree with the majority.

What needs to be secured?

Security on USENET amounts to the question, "Should anybody and everybody be able to perform this action?" If the answer is yes, you need no security. If no, you need some security to divide those who you do wish to perform the action from those who you don't. Security of course has a cost, so sometimes you're willing to accept letting anybody perform some action if the risk of that is less than the bother of security. When the net was smaller, and there were few malicious people about, security wasn't necessary simply because even though anybody could do certain things, they tended not to. [Postel82]

Now on USENET, the only "action" is the posting of an article.

However, this gets broken down based on what the headers of the article do, and in particular the Control header on control messages. So while "post an article" is not the unit you secure, you are interested in "post a cancel" or "post an article in a moderated newsgroup."

Here is a list of the actions on USENET I believe most people would prefer not be available to anybody and everybody. As such, we must address how to secure them.

Arbitrary users should not be able to:

1. Cancel an article they were not involved in originating
2. Post an article identified as coming from somebody else against that person's will.
3. Post to a moderated group without the moderator's approval
4. Violate a newsgroup's policies on crossposting, article size, mime-types, etc.
5. Machine post vast volumes of articles to one or many groups
6. Create a group
7. Create or replace a named article with a specified purpose
8. Change or set the status of a group
9. Issue a sendsys in the name of another, or possibly any sendsys at all
10. Delete a group
11. Modify the headers or body of another's message
12. Block the propagation of another's message by hijacking its message-id
13. Issue a checkgroups or change a group description

While there is some fine debate about some, and in some cases these rules may vary in some hierarchies (for example alt might allow any party to create a group) I think that for the mainstream of USENET, ideally most people would prefer these functions were not entirely open. [Reid91]

Who can be trusted

If not all parties can be trusted to perform these actions, who can or should be trusted? Well, that varies from action to action. In some cases, like the cancel message, everybody agrees the original poster of a message should be trusted, and most agree the administrators of the equipment used to insert the posting into the net should be trusted as well. Many others wish to pick specific 3rd parties and trust them, to deal with abuse. For other functions it's more political. The actions themselves require subjective judgement and must be performed by individuals or groups who win the trust of the machine owners who in turn grant it. It turns out that the vast majority of people on USENET can be trusted, at at least given the benefit of the doubt, with their trust revoked only after it is abused. That's how the net used to work, but there was no way to revoke the trust when people started abusing. The answers as to who people want to trust to perform these actions are varied and many. The underlying security system has to allow people to create the various structures of trust and enabling that they desire.

System Goals

Delegation

It's vital that all trust be easily delegated. Site admins don't want to worry about the administrative problems of newsgroups or other small subsets of the net. They want to examine the big picture at best, and sometimes no picture at all. Most would like to just leave things to work, and only deal with problems if problems occur that are big enough to reach their attention. The delegation itself must be secure.

Synchronization

In addition, secure delegation is the only way that cooperation on USENET can work. If each site tunes its own parameters for a newsgroup independently, either by deliberate will or more commonly just by accident, then the group starts being useful to none. All sites have to be reasonably in sync -- the near unanimity -- about how most components of the net work. You can't have a third of the sites thinking a group is moderated with one moderator, another third naming a different moderator and the other third thinking the group is unmoderated. The group becomes damaged for everybody.

Extensibility

While I've listed common things we want to secure today, it is certain that other things will come up. New control messages or headers. New newsgroup policies and ways

to run newsgroups. Good software design insists that there be one system -- since this is complex enough as it is

-- and that it be general and extensible, just like the other facets of USENET are.

Security

Good security is of course secure. Once you have a general security system, you don't leave things unsecured without an explicit reason for doing so. Sadly, when you have malicious attackers, if you close one security hole they just move to the next. You must close all the holes they will try. This is usually not totally possible, but one tries to get as close as possible to the goal as one can.

Anarchic

Unlike most systems, a security system for USENET has to factor in its anarchic nature. Authority remains with site admins, you can never prevent that. However, the system must allow people to work together, to cooperate and compromise as they see fit. Security becomes a tradeoff between the burdens of complexity of security and cooperation and the risks of insecurity. [Reid91]

Conclusions and Comparisons

The InterNetNews architecture works. Profiling a production installation for 24 hours showed that *open* (2) accounted for 10% of the run time. Since the server only does one *open* (2) per article, it is not clear if any other performance tuning is needed. The profiling overhead accounted for 5% of the run-time. Several optimizations are available because there is only one process, and because it is always running. For example, avoiding duplicates is an integral part of the server. If a second site offers an article while a first site is sending it, the NNTP code will put the channel to ``sleep'' for a short while before replying to the second offer. This is usually enough time to have the first site finish sending the article, reducing the number of duplicates from hundreds to nearly none, with no external programs. Since the server is always running, the system has a much smoother performance curve. As a result, it ``feels'' much faster to users. Another unexpected benefit is that articles are accepted or rejected synchronously. A user can post an article, and by the time their posting agent has returned, it has been written to the spool directory and queued for remote transfer. If there is a problem such as having an illegal newsgroup specified, the user founds out immediately. The design of the server seems to be very good, split into abstractions that are very independent. For example, sites have no knowledge of incoming NNTP connections. Using callbacks lets each portion of the server safely do I/O without worrying that it might affect other parts. Much of the Usenet processing becomes trivial when serialized, such as access to the history file. The design has also led to a fairly small program: it is under 13,000 lines, and about 20% of them are comments. This compares favorable to the 7,400 lines in the equivalent C News program and the 7,600 lines in the NNTP reference implementation. [Reid91]

References

1]. [Crocker82] David H. Crocker, *Standard for the Format of ARPA Internet Text Messages* , Request For Comments 822, Marina del Rey, CA: Information Sciences Institute, 1982.

2.][Adams87] Rick Adams, Mark Horton, *Standard for Interchange of USENET Messages* , Request For Comments 1036, Marina del Rey, CA: Information Sciences Institute, 1987.

3] [Adams92] Rick Adams, *Total traffic through uunet for thelast2weeks*Usenetmessage<1992Apr8.193050.8963@uunet.uu.net> in *news.lists* , April, 1992.

4] [Collyer87] Geoff Collyer and Henry Spencer, *News Need not be Slow* , Usenix Winter Conference, 1987.

5] [Kantor86] Brian Kantor, Phil Lapsley, *Network News Transfer Protocol*: A Proposed Standard for the Stream - Based Transmission of News , Request for Comments 977, Marina del Rey, CA: Information Sciences Institute, 1986.

6] [Postel82] Jonathan B. Postel, *Simple Mail Transfer Protocol* , Request For Comments 821, Marina del Rey, CA: Information Sciences Institute, 1982.

7] [Reid91] Brian Reid, *Usenet Readership Summary* Report for May 91 , Usenet message<1991Jun2.141124.12753 @pa.dec.com