SANS Institute

A Security Analysis of the Gnutella Peer-to-Peer Protocol

by

Kirk Cheney

March 2002

Advanced Incident Handling and Hacker Exploits
GCIH Practical Assignment (version 2.0)

Support for the Cyber Defense Initiative

1. Introduction

The Cyber Defense Initiative is a scheme established by the SANS Institute for the purpose of sharing relevant, timely information regarding current Internet threats amongst security professionals, in much the same way that hackers and crackers use the Internet to share information. The goal of the Cyber Defense Initiative is to provide the information, resources, and tools organisations need to improve the security of their sites [1]. This paper is written in support of the Cyber Defense Initiative, by investigating and analysing possible vulnerabilities in one Internet service that is, at the time of writing, in the list of the top ten most attacked ports, according to statistics provided by the Internet Storm Center at Incidents.org [2].

Incidents.org [2] is an organisation that was set up by members of the SANS Institute to focus on current Internet security threats, and to provide "real-time 'threat driven' security intelligence and support to organisations and individuals" [3]. The Internet Storm Center at Incidents.org is constantly monitoring network traffic from around the globe in order to quickly identify potential new threats to computer systems, and to warn Internet users and advise them of methods of responding to those threats. Computer security practitioners throughout the world are encouraged to contribute firewall and intrusion detection system logs to the Incident Storm Center, to provide them with the data that they need to enable them to monitor and analyse trends which may indicate potential new threats to systems. The data is held in the Consensus Intrusion Database (CID), and provides real-time threat information to the community, through www.incidents.org, in the form of graphical threat maps and tables.

The home page at www.incidents.org provides a current threat map showing the most attacked, or probed, TCP/IP ports and the geographic distribution of the attack sources. The map shown below in Figure 1.1 [1] shows the state of the information provided by the Internet Storm Center on 8th March 2002.

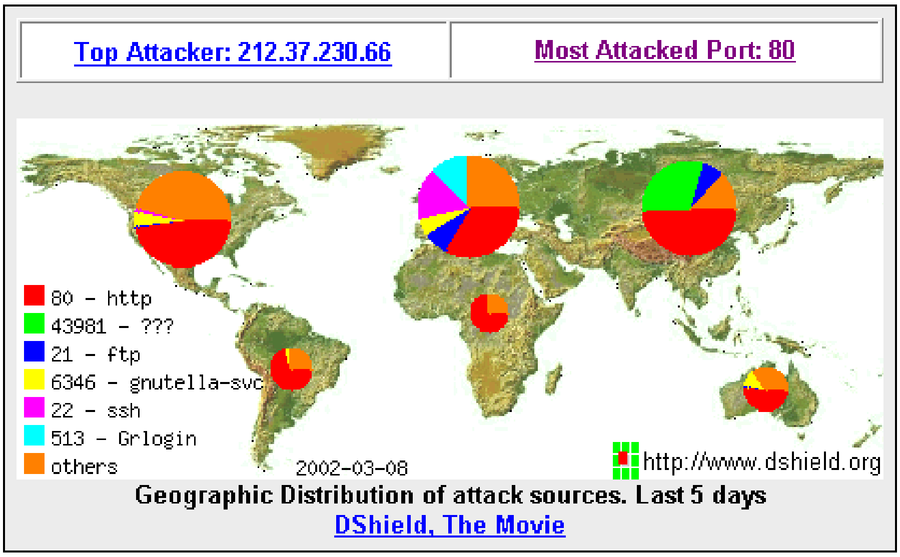


Figure 1.1 – Threat map from http://www.incidents.org/, 8th March 2002

The threat map above details the threat statistics for 8th March, showing intrusion attempts from Internet connected sites all over the world. The key on the left shows which ports were most targeted, and each region has a pie-chart showing how the attacks were distributed in that part of the world. As a computer security professional in the Australia and New Zealand region, I was interested in further investigating the distribution of these attacks within Australasia. Each of the pie-charts shown on the map links to a page at DShield.org [4], which hosts the CID statistics, from which information can be found for each of the continents. The information for Australasia on 8th March is shown in the table below [5]:

								Port
								80
								6346
								123
								53
515	339						339	515
1214	308			7			315	1214
25	258						258	25
6699	200						200	6699
31337	132						132	31337
22	126						126	22
	10136	22	3	68	3	3	10235	
	99%	0%	0%	0%	0%	0%	100%	
	AU	FJ	NC	NZ	PF	TO	Total	

Table 1.1 – DShield.org statistics for Australasia, 8th March 2002

From the information given in the graphical threat map in Figure 1.1 and in Table 1.1 above, it is clear that port 80 is targeted significantly more than any other port, in all regions, as can be seen by the large proportion of red in the pie-charts shown in the diagram. This is not surprising, as it hosts the *http* service, which is the service used by all World Wide Web servers for hosting Web pages, a service that most organisations provide, and also one that has a number of known vulnerabilities in various implementations, which has made it a favourite for hackers for some time. However, it is interesting to note from Table 1.1 that, whilst port 80 is the most attacked port in the Australasia region, there is little consistency shown between this region an the rest of the world. One other port does stand out as being high in both the world average and the local region average, and that is port 6346, the "gnutella-svc" port. For this paper, I have chosen to investigate and analyse this port and the services running on it.

2. Targeted Port

2.1 Port 6346

According to the Internet Assigned Numbers Authority (IANA) [6], port 6346 (both TCP and UDP) is reserved for "gnutella-svc" and port 6347 is reserved for "gnutella-rtr". IANA is the registration body that is responsible for assigning TCP/IP ports as protocols and services are made available on the Internet. TCP/IP uses a client-server paradigm to establish connections between users and the services that they wish to use. The ports below 1024 are referred to as the "Well Known Ports", and these ports have traditionally been used to host common services, such as *http* for Web browsing (port 80), *smtp* for e-mail transfers (port 25), etc. Ports 1024 and above have traditionally been available to clients to establish connections, but in recent years as more services and protocols have been defined for the new applications working over the Internet additional ports have been required. The block of ports between 1024 and 49151 was made available for this purpose, known by IANA as the "Registered Ports", which means that a port in this range could be used by either client or server processes.

The "gnutella-svc" port (port 6346) is used by a protocol known as Gnutella, which has been assigned ports 6346 and 6347 in the range of "Registered Ports". Gnutella is a peer-to-peer file-sharing protocol, which allows Internet users to share files with other users without the need for a dedicated server to store the files on. Gnutella is investigated further below.

2.2 Peer-to-Peer Networking

Peer-to-peer networking refers to the concept of multiple computers sharing resources and services directly with each other, without the interaction of a mediating server. Each computer in the peer-to-peer network can act as both client and server, either providing its resources for others to use or making use of the resources provided by others [7, 8].

The peer-to-peer model has been around for many years, but has gained popularity recently with the spread of Napster, a centralised peer-to-peer network for users to share music and other media files. A number of other peer-to-peer applications have appeared since, one of which is Gnutella. Whilst the traditional model of a peer-to-peer network allowed all useful computer resources to be shared, such as files, idle CPU cycles, cache memory and data storage space, these new services are designed to share data files only. In the case of Napster, it was confined to media files, although packages soon became available that could wrap any file in a media format for sharing via the Napster network.

The Napster network consisted of a central server that listed all the nodes connected to the network, and the files that those nodes had available for download. Napster has since ceased to operate, when it was shut down following legal action brought about by the Recording Industry Association of America (RAII) on behalf of the major recording companies, in December 1999 [7]. The RAII and recording industry bodies considered that the free sharing of unlicensed recordings was in breach of their copyrights, and the courts agreed. The "weakness" of the Napster model, which allowed it to be closed down, was its central servers that mediated all exchanges. Without those, the network could not operate.

An alternative to the centralised model is fully distributed peer-to-peer networking, in which each host communicates only with the hosts around it. Gnutella makes use of this paradigm. The main obstacles to establishing a peer-to-peer network of this nature are that nodes need to know the existence of other nodes, and they need to have a means of finding the files that they seek. The Gnutella protocol has been designed to resolve both of these problems. The advantages of a distributed peer-to-peer model are that, without the central servers, there are no individuals or organisations that are responsible for the control of the network, so they cannot restrict it nor can they be made to withdraw the service. This also gives the network a great deal of robustness, as whole sections of the Internet could be shut off from the network, yet it could continue to operate and expand with little loss of service, much like the Internet itself.

3. Gnutella

The Gnutella protocol was conceived and originally released by Justin Frankel and Tom Pepper of Nullsoft, a subsidiary of AOL [8]. AOL did not wish to proceed with development of the protocol, and withdrew the project, but not before a number of clients had been downloaded, and the beginnings of the network established. The idea was popular, and future development of Gnutella was continued by the open-source community.

Gnutella is a protocol definition, and the protocol defines the way in which nodes in the network can communicate with each other. Each node must be both a server and a client, and is known as a "servent" (a contraction of *serv*er and cli*ent*). Servents can send out searches to the network, looking for the files that the user wants, or they can answer the queries sent out by others by comparing the search request with their own database of files and responding if they have a file that meets the search criteria. The network communicates like a game of Chinese Whispers, or Telephone [10], in which each servent also acts as a hub as well, all the time receiving queries or responses from other servents around them and passing them on all the others to which they are connected. In this way servents can communicate with thousands of others, whilst only having to maintain connections to a few others around them. This means that communications can reach a far wider audience, and at the same time offer anonymity to those that want to look around.

When a search has been completed, and a user has found the file that they are looking for, they can download it from the host that has it, in much the same way that files can be downloaded from web servers. This part of the process is not done through the network of servents, but is instead done directly from the serving servent to the client servent. In this way the network is not weighed down with unnecessary file exchanges.

3.1 The Gnutella Protocol

So how does Gnutella work? In the protocol specification, Gnutella claims to be nothing more than a protocol for distributed search [11]. This is true to some extent, as the protocol has been defined simply to enable servents to query the network for other active servents, and to query those active servents to request files that match a simple search criteria. The actual exchange of files is handled separately using the *http* protocol, and the implementation of this mechanism is left to the developers of the servent software, and is beyond the scope of the Gnutella protocol description. The only assistance given by the protocol in the quest to exchange files is in the definition of the "Push" descriptor, which can help to establish a connection between a requesting servent and a servent that cannot be contacted directly to establish the *http* connection, such as one behind a firewall.

The Gnutella protocol consists of a set of descriptors. There are only five defined descriptors in the current version of the protocol (Gnutella Protocol Specification v0.4 [11]). These are Ping, Pong, Query, QueryHit and Push. The purpose of each of these is described below:

Ping The Ping function works slightly differently to the traditional TCP/IP Ping implementation, which is an "Are you alive?" query to another host address. The Gnutella Ping descriptor is meant more as an announcement that the initiating host has joined the network. The servent that receives the Ping will reply with a Pong, and will also forward the Ping to all other servents to which it is connected.

Pong The Pong function, as described above, is the response to a Ping request. On receipt of a Ping, a servent will reply to the sender of the Ping with a Pong, which contains information including the IP address and port on which the servent is listening, the number of files it is sharing with the network and the total amount of data contained in those files. The recipient of the Pong will be the servent that last sent the Ping, which is not necessarily the servent that initiated the Ping; the queries and responses route their way through the network instead of initiating a new direct connection between the responder and the original Ping sender.

Query The Query function is the basis of the search capability. The Query descriptor contains two basic requests: a minimum connection speed and a search string containing the data that the user is looking for. The Query descriptor will be routed in the same manner as the Ping, to directly connected servents, who will forward it to their directly connected servents, and so on.

QueryHit A servent will reply to a Query descriptor with a QueryHit descriptor if it matches the search string in the Query with one of the files that it is sharing with the network, and if it can also achieve the minimum connection speed specified in the Query. The QueryHit response will be returned to the servent that last sent the Query, who will forward it to the servent that sent him the Query, and so on until it reaches its final destination, which is the host that originated the Query. The QueryHit descriptor contains information such as the number of hits, the IP address and port on which it is listening, the data transfer speed available, and the result set, which for each hit contains the file name and size, and a unique identifier for the file. There is also a unique servent identifier, which is required for the Push descriptor to establish a connection.

Push The Push descriptor is designed to allow the Gnutella network to continue to operate when a servent that is providing files resides behind a firewall that will not allow direct connections to be established to the host. The assumption made is that the internal host will be allowed to establish connections from inside the firewalled network to a host outside the firewall. Once the servent inside the firewall has responded to a Query with a QueryHit and a list of files, the requesting servent is unable to make a direct connection to download the file because of the firewall. In this case, the Push descriptor is sent, which will request that the firewalled servent instead initiate the connection from inside the perimeter, and then send the file. The Push descriptor includes the servent identifier given in the QueryHit descriptor returned in the previous transaction, the file index of the requested file, and the IP address and port on which the requesting servent is listening. On receipt of the Push, the providing servent will attempt to initiate a connection to the IP address and port provided, and then pass the file using the *http* protocol across this connection.

The following diagrams show how the communications described above are propogated through the network:

Ping / Pong Transaction

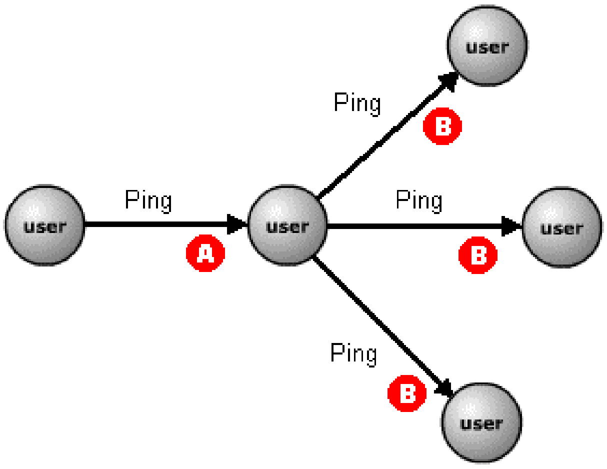


Figure 3.1 - Ping routing

Figure 3.1 above shows the way in which a Ping packet is routed though the network. The initiator of the Ping sends it only to those servents to which it is directly connected (in the diagram, this is only one servent) (A). That servent then forwards the Ping to all servents to which it is directly connected, except for the one from which it received the message (B).

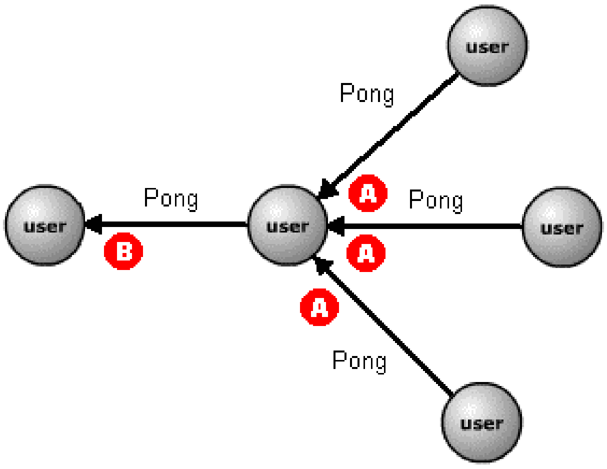


Figure 3.2 - Pong return route

Figure 3.2 above shows the return route of the Pongs. Each servent will reply with a Pong, sent to the servent from which it received the Ping (A). That servent will then forward in the direction from which it came, and so on until it reaches the initiator (B).

Query / QueryHit Transaction

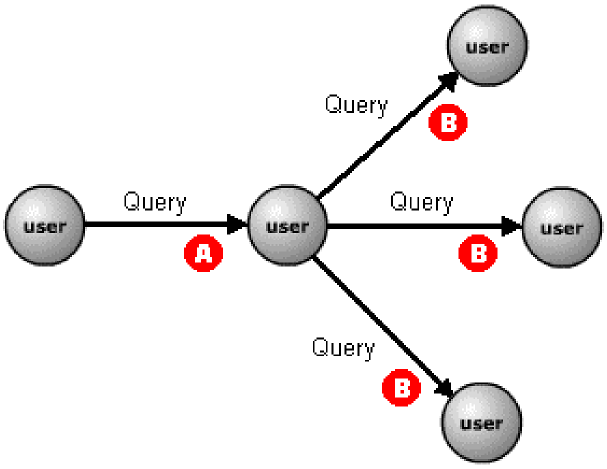


Figure 3.3 - Query routing

The propogation of the Query packet, as shown in Figure 3.3 above, works in the same manner as the Ping. The message will be forwarded through the network from one servent to the next in the chain, until the TTL (Time-To-Live) expires. The TTL field is described in more detail later.

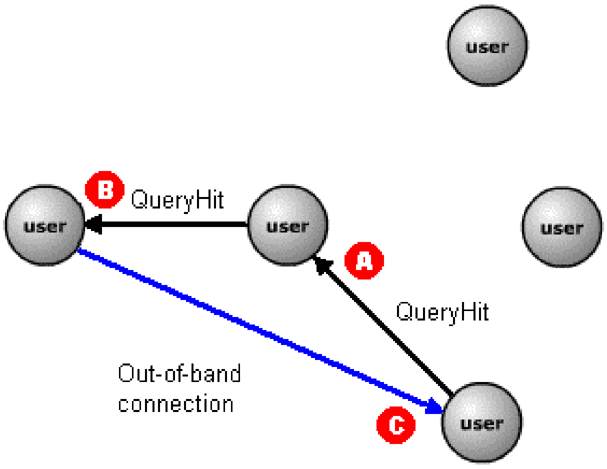


Figure 3.4 - QueryHit return route and subsequent out-of-band connection

In the example shown in Figure 3.4 above, only a single servent was able to match the search criteria given in the Query message with a file in its shared folders. The details of the matching file is returned to the previous servent in the QueryHit message (A), which is again forwarded through the network until it reaches the initiating servent (B). The initiating servent can then open an out-of-band communication channel with the responding servent (C), using the IP address and port given in the QueryHit descriptor that was returned. The file can then be directly downloaded from the responding servent to the requesting servent using *http* protocol commands across this new connection.

Bypassing a Firewall using Push

If the attempt by the requesting servent to establish an out-of-band connection to the servent hosting the desired file is unsuccessful, then it is assumed that the destination servent is located within a controlled perimeter that does not allow incoming connections, and a Push is attempted.

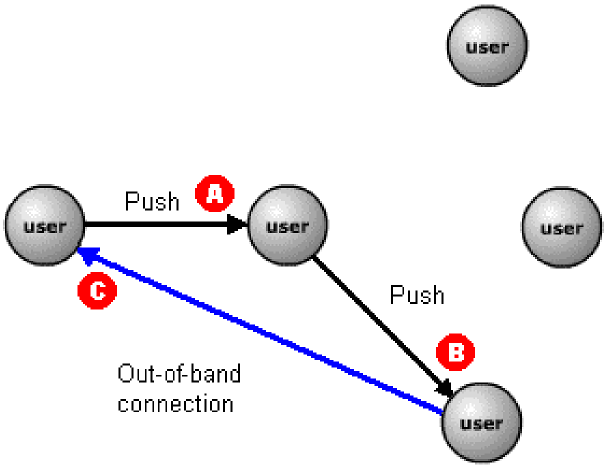


Figure 3.5 - Push request

Once the server has received the QueryHit and unsuccessfully attempted a direct connection, then it can request that the hosting servent initiate the out-of-band connection by sending a Push request. The Push message again passes through the established Gnutella network (A), but gets passed on a direct path to the destination servent through the use of the unique servent identifier (B). The hosting servent can then try to establish an out-of-band connection through the firewall to the requesting servent (C), using the IP address and port given in the Push message. If it is successful, then it will issue a “GIV” command on the *http* channel, including the required file's unique index, filename, and the servent identifier given in the Push message. The requesting server can then initiate the *http* file download in response to this “GIV” message.

3.2 Communication Channels

The Gnutella protocol rides on top of TCP, the language of the Internet, using standard TCP routing and headers for the packets. The Gnutella protocol descriptor packets, which are transmitted in the body of the TCP packets, then have their own header and body, which are fully defined in the protocol specification [11].

In order to establish a connection to another servent to join the Gnutella network, a host must first establish a TCP session with the other machine. To do this, the servent needs to be provided with an IP address and listening port of another active servent within the network. It is important to note here that Gnutella servents are not confined to using the default ports of 6346 and 6347. The port is configurable, and so the servent can listen on any port that the user wishes to set.

Acquiring the initial address and port to connect with is not handled by the protocol, but instead they need to be provided manually by the user, or from a cache of known addresses. In practice, the servent developers often include a number of default caching servents in the application code, and when the application is started the servent connects to one of these defaults and is supplied with a list of active servents to connect to. Another alternative is to use a service such as HostsCache.com [12], which is a network of servents that host only cached servent lists for peer-to-peer networks. After connecting to a HostsCache servent the cached list will be downloaded to your local cache, and then the connection will be closed by the remote servent. Your servent will then attempt to contact to another servent on the newly acquired list, until it is able to connect into the network.

The servent configuration files will allow the user to specify how many connections to keep open. The default number is usually around 4, although this can be changed by the user. This is the number of different servents that your machine will connect directly to at any given time.

Before the two machines can negotiate a connection and start to communicate using the Gnutella protocol structure, they must first establish their identity, and their willingness and ability to open a channel. This is done by the requesting host sending a Gnutella connection request. This is a simple ascii string message in the form [11]:

```
GNUTELLA CONNECT/<protocol version string>\n\n
```

where the protocol version string, for the current version of the protocol, would be "0.4".

If the receiving host receives a message like this, it must choose whether to accept the connection or not. This decision is based on factors such as whether it has an available incoming connection slot, and whether it is using the same version of the protocol. If the connection is accepted, the servent will reply with the message [11]:

```
GNUTELLA OK\n\n
```

Any response other than this indicates that the connection has been rejected, and the local servent will try the next cached address on the list. Some servent implementations may also include additional information in this transaction.

Once this negotiation has been successfully concluded, the TCP connection between the two servents remains open until one or other servent closes it, or until it is terminated by network problems, etc. All communications along this channel will be conducted using Gnutella protocol descriptors. If the connection is terminated in this manner, then the local host will simply connect to another servent in the cache.

3.3 TTL and Hops

The header of each descriptor conforms to a specified format [11]. The header contains five fields; three of these are used to identify the message, the descriptor being used, and the amount of data in the payload. The remaining two fields are the TTL (Time To Live) and the Hops (hop count).

The TTL field specifies the number of times a packet will be forwarded through the network, and is included to prevent packets from overloading the network by illiciting responses to each Ping or Query from enormous numbers of servents, or from getting into a loop and circling through the network forever. Each time a servent receives a packet the TTL is decremented. Before passing it to the next servent, the value of the TTL is checked. If it is greater than zero then the packet is forwarded, otherwise the packet is dropped and expires.

The initial value of the TTL is important in tuning the performance of the network. If the TTL is too small, then the Query desciptors will not travel far, and the chances of matching a search string are greatly reduced. If it is too large, then the huge number of responses might use up all the network bandwidth, and shutdown the network.

The Hops field specifies the number of times that the packet has been forwarded through the network at any given time. When a packet is generated, the Hops field is zero. Each time a servent receives a packet the Hops field is incremented, before passing it to the next servent. The sum of the Hops field and the TTL field at any point in a packet's journey will always give the value of the original TTL. This allows a replying servent to know how large the TTL in the reply must be to reach the requesting servent.

3.4 Packet Examples

In order to better understand the interractions going on between servents, we will look at some packet captures showing a servent communicating with the network.

The first stage in the connection process is to establish a TCP connection with the second servent, which is achieved using the 3-way TCP handshake shown below:

Source	Destination	Protocol	Information
203.33.188.37	62.30.35.66	TCP	1826 > 6346 [SYN] Seq=17812889 Ack=0 Win=8192 Len=07
62.30.35.66	203.33.188.37	TCP	6346 > 1826 [SYN, ACK] Seq=22405320 Ack=17812890 Win=64240 Len=0
203.33.188.37	62.30.35.66	TCP	1826 > 6346 [ACK] Seq=17812890 Ack=22405321 Win=8576 Len=0

When a connection is established, the connecting servent will issue the "Gnutella Connect" message, as seen in the following IP packet:

```
0000   20 53 52 43 00 00 44 45 53 54 00 00 08 00 45 00     SRC..DEST....E.
0010   00 3e 4a 4a 40 00 80 06 c7 c8 cb 21 bc 25 3e 1e    .>JJ@......!.%>.
0020   23 42 07 22 18 ca 01 0f cd 9a 01 55 e0 c9 50 18    #B.".......U..P.
0030   21 80 3b 0f 00 00 47 4e 55 54 45 4c 4c 41 20 43    !.;...GNUTELLA C
0040   4f 4e 4e 45 43 54 2f 30 2e 36 0d 0a                  ONNECT/0.6..
```

The reply to this message includes the following packet, as expected:

```
0000   44 45 53 54 00 00 20 53 52 43 00 00 08 00 45 00    DEST.. SRC....E.
0010   00 3d 0f 6e 40 00 67 06 1b a6 3e 1e 23 42 cb 21    .=.n@.g...>.#B.!
0020   bc 25 18 ca 07 22 01 55 e0 c9 01 0f ce 1a 50 18    .%...".U......P.
0030   fa 70 aa 15 00 00 47 4e 55 54 45 4c 4c 41 2f 30    .p....GNUTELLA/0
0040   2e 36 20 32 30 30 20 4f 4b 0d 0a                      .6 200 OK..
```

In addition, this reply is followed up with further information including a list of cached servent addresses.

Once the Gnutella connection is established, all further communications are done using the Gnutella descriptors described in the previous sections. Shown below are a Ping descriptor, received at the local host, and the corresponding Pong descriptor sent in reply:

```
0000   44 45 53 54 00 00 20 53 52 43 00 00 08 00 45 00    DEST.. SRC....E.
0010   00 3f 0f a1 40 00 67 06 1b 71 3e 1e 23 42 cb 21    .?..@.g..q>.#B.!
0020   bc 25 18 ca 07 22 01 55 e3 08 01 0f ce 48 50 18    .%...".U.....HP.
0030   fa 42 0a 1a 00 00 84 c6 79 53 65 a0 25 d8 ff f8    .B......ySe.%...
0040   e0 c8 ed b9 97 00 00 01 00 00 00 00 00               .............
```

The packet above is the Ping descriptor. The descriptor begins at byte 0036 with the 16 byte ID field, in this case given as "84C6795365A025D8FFF8E0C8EDB99700", then we see the payload descriptor of "00", which is a Ping. This is followed by a TTL of "01", a hop count of "00", and a payload length of zero (the last 4 bytes).

```
0000   20 53 52 43 00 00 44 45 53 54 00 00 08 00 45 00     SRC..DEST....E.
0010   00 6a 52 4a 40 00 80 06 bf 9c cb 21 bc 25 3e 1e    .jRJ@......!.%>.
0020   23 42 07 22 18 ca 01 0f ce 48 01 55 e3 1f 50 18    #B.".....H.U..P.
0030   21 57 cc 2a 00 00 08 f8 e1 a0 63 05 86 db ff 4a    !W.*......c....J
0040   3b 92 48 a0 8c 00 30 01 00 06 00 00 00 00 00 40    ;.H...0........@
0050   00 00 07 84 c6 79 53 65 a0 25 d8 ff f8 e0 c8 ed    .....ySe.%......
0060   b9 97 00 01 01 00 0e 00 00 00 cb 18 cb 21 bc 25    ..............!.%
0070   0d 00 00 00 6d 19 00 00                                ....m...
```

The returned Pong descriptor is shown above. This descriptor begins at byte 0053 with the ID field. This packet has a payload descriptor of "01" (Pong), a TTL of 1, hop count zero and a payload length of 14 bytes (0e hex). The payload of a Pong descriptor then consists of the listening port (6346), the servent's IP address (203.33.188.37), the number of shared files (13) and the number of kilobytes of data in those files (6509).

When a user wishes to search the Gnutella network for files, he will enter a search string into the user interface of the servent, and the servent will send a Query descriptor to the network. The following packet capture shows the Query packet for a search string of “doc”:

```
0000   20 53 52 43 00 00 44 45 53 54 00 00 08 00 45 00     SRC..DEST....E.
0010   00 46 6d 4a 40 00 80 06 a4 c0 cb 21 bc 25 3e 1e    .FmJ@......!.%>.
0020   23 42 07 22 18 ca 01 0f cf 8d 01 55 e3 5b 50 18    #B."......U.[P.
0030   21 1b 04 cf 00 00 93 7f 8e 98 9a ce c3 34 ff 7c    !............4.|
0040   d1 7f 32 f5 58 00 80 07 00 07 00 00 00 00 00 64    ..2.X..........d
0050   6f 63 00 00                                                   oc..
```

We can see from this packet that the payload descriptor is “80”, which specifies a Query descriptor. The TTL for this packet has been set at 7, which is the default value for this servent application, and the hop count remains at zero. The payload length is 7 bytes. The Query descriptor payload contains the minimum download speed, which is zero in this example, and the search text in ascii, which is “doc”. One of the responses is shown here:

```
0000   44 45 53 54 00 00 20 53 52 43 00 00 08 00 45 00    DEST.. SRC....E.
0010   00 3f 11 f0 40 00 67 06 19 22 3e 1e 23 42 cb 21    .?..@.g..">.#B.!
0020   bc 25 18 ca 07 22 01 55 e8 ff 01 0f cf d0 50 18    .%...".U......P.
0030   fa f0 8b 0d 00 00 93 7f 8e 98 9a ce c3 34 ff 7c    .............4.|
0040   d1 7f 32 f5 58 00 81 01 06 e0 02 00 00              ..2.X........
```

This packet is the first in a series of three packets that contain the entire QueryHit response. As with the previous packets, the standard descriptor header here shows us that the descriptor is of type “81”, which is a QueryHit. The TTL is 1 and the hop count is 6, so we know that the responding servent is 7 hops away, the maximum distance that was allowed by the Query that was sent out. The remaining information tells us that the descriptor payload is 736 bytes long.

```
0000   44 45 53 54 00 00 20 53 52 43 00 00 08 00 45 00    DEST.. SRC....E.
0010   02 40 11 f1 40 00 67 06 17 20 3e 1e 23 42 cb 21    .@..@.g.. >.#B.!
0020   bc 25 18 ca 07 22 01 55 e9 16 01 0f cf d0 50 10    .%...".U......P.
0030   fa f0 b4 ed 00 00 08 ca 18 41 1f 50 8e 38 00 00    .........A.P.8..
0040   00 05 00 00 00 00 5e 00 00 65 6d 61 69 6c 73 20    ......^..emails
0050   61 6e 64 20 61 64 64 72 65 73 73 65 64 20 6f 66    and addressed of
0060   20 70 6f 74 65 6e 74 69 61 6c 20 6a 6f 62 73 33     potential jobs3
0070   2e 36 2e 30 32 2e 64 6f 63 00 75 72 6e 3a 73 68    .6.02.doc.urn:sh
0080   61 31 3a 37 35 57 4c 37 42 4c 48 5a 4b 48 50 4f    a1:75WL7BLHZKHPO
0090   35 5a 58 47 58 32 49 44 41 33 55 36 4b 5a 4e 57    5ZXGX2IDA3U6KZNW
00a0   55 35 53 00 06 00 00 00 00 50 00 00 52 65 73 75    U5S......P..Resu
00b0   6d 65 20 43 6f 76 65 72 20 6c 65 74 74 65 72 2e    me Cover letter.
00c0   64 6f 63 00 75 72 6e 3a 73 68 61 31 3a 51 34 57    doc.urn:sha1:Q4W
00d0   46 45 45 4c 52 59 58 43 41 42 34 51 50 36 37 45    FEELRYXCAB4QP67E
00e0   46 44 35 45 43 44 4d 46 4c 4c 42 37 55 00 07 00    FD5ECDMFLLB7U...
00f0   00 00 00 50 00 00 49 20 77 61 6e 74 65 64 20 74    ...P..I wanted t
0100   6f 20 65 78 70 6c 61 69 6e 20 6d 79 20 73 69 74    o explain my sit
0110   75 61 74 69 6f 6e 20 72 65 67 61 72 64 69 6e 67    uation regarding
0120   20 6d 79 20 61 74 74 65 6e 64 61 6e 63 65 20 74     my attendance t
0130   6f 20 73 63 68 6f 6f 6c 2e 64 6f 63 00 75 72 6e    o school.doc.urn
0140   3a 73 68 61 31 3a 53 34 37 34 55 55 58 4f 47 57    :sha1:S474UUXOGW
0150   36 4e 42 52 34 54 44 52 4b 44 33 42 48 35 50 45    6NBR4TDRKD3BH5PE
0160   4a 5a 4a 32 53 33 00 08 00 00 00 00 7e 00 00 44    JZJ2S3......~..D
0170   61 6e 69 65 6c 6c 65 20 53 75 6f 6a 61 72 76 69    anielle Suojarvi
0180   20 72 65 73 75 6d 65 20 33 2e 31 31 2e 30 32 2e     resume 3.11.02.
```

```
0190   64 6f 63 00 75 72 6e 3a 73 68 61 31 3a 44 43 4a    doc.urn:sha1:DCJ
01a0   33 4f 55 49 47 53 47 36 4d 35 34 52 58 35 44 4b    3OUIGSG6M54RX5DK
01b0   41 4d 37 43 36 44 4b 51 49 52 49 4b 51 00 10 00    AM7C6DKQIRIKQ...
01c0   00 00 00 4e 00 00 53 74 65 76 65 73 20 6a 6f 62    ...N..Steves job
01d0   20 6c 65 74 74 65 72 4d 61 72 63 68 20 32 37 2e     letterMarch 27.
01e0   64 6f 63 00 75 72 6e 3a 73 68 61 31 3a 43 57 4a    doc.urn:sha1:CWJ
01f0   34 41 4e 48 4a 35 4c 4f 58 34 44 48 4c 52 37 33    4ANHJ5LOX4DHLR73
0200   51 37 54 53 34 42 50 41 52 45 52 45 59 00 11 00    Q7TS4BPAREREY...
0210   00 00 00 60 00 00 4b 61 74 72 69 6e 61 20 4e 20    ...`..Katrina N
0220   4a 61 6b 75 62 6f 77 73 6b 69 2e 64 6f 63 00 75    Jakubowski.doc.u
0230   72 6e 3a 73 68 61 31 3a 55 53 4b 48 55 36 35 44    rn:sha1:USKHU65D
0240   48 52 51 56 35 50 41 4d 45 4d 46 4f 37 4f           HRQV5PAMEMFO7O
```

The descriptor payload begins in this second packet, at byte 0036. The first byte tells us that the host had 8 matches for the search criteria, then gives the listening port and IP address in hexadecimal (“ca 18” and “41 1f 50 8e” respectively, which gives a listening port of 6346 at IP address 65.31.80.142) and a speed of 56 kb/s. It then lists all the files that matched the Query, giving each a file index, file size and filename. Note the first file in the list, which is named “emails and addressed of potential jobs3.6.02.doc” and has the file index of “05” which we will download from this host.

```
0000   44 45 53 54 00 00 20 53 52 43 00 00 08 00 45 00    DEST.. SRC....E.
0010   00 f0 11 f2 40 00 67 06 18 6f 3e 1e 23 42 cb 21    ....@.g..o>.#B.!
0020   bc 25 18 ca 07 22 01 55 eb 2e 01 0f cf d0 50 18    .%...".U......P.
0030   fa f0 da cf 00 00 48 4c 52 34 35 57 36 56 4b 45    ......HLR45W6VKE
0040   00 12 00 00 00 00 5c 00 00 4b 61 74 72 69 6e 61    ......\..Katrina
0050   20 4e 20 4a 61 6b 75 62 6f 77 73 6b 69 20 4e 45     N Jakubowski NE
0060   57 2e 64 6f 63 00 75 72 6e 3a 73 68 61 31 3a 4a    W.doc.urn:sha1:J
0070   55 34 33 50 33 32 59 35 4f 35 50 5a 46 57 36 49    U43P32Y5O5PZFW6I
0080   45 42 41 43 45 33 35 32 59 54 37 44 50 52 41 00    EBACE352YT7DPRA.
0090   14 00 00 00 00 5e 00 00 45 6d 61 69 6c 20 61 64    .....^..Email ad
00a0   64 72 65 73 73 65 73 2e 64 6f 63 00 75 72 6e 3a    dresses.doc.urn:
00b0   73 68 61 31 3a 56 4c 51 54 43 54 42 48 42 47 33    sha1:VLQTCTBHBG3
00c0   5a 56 55 51 35 34 5a 52 56 4b 4e 50 57 58 50 50    ZVUQ54ZRVKNPWXPP
00d0   35 48 4d 45 5a 00 42 45 41 52 02 1c 09 18 00 03    5HMEZ.BEAR......
00e0   02 00 82 02 00 00 01 01 01 02 24 03 1c 00 d3 f2    ..........$.....
00f0   7b ab 4d b4 b5 38 ff 22 0e 8d 48 cd 2f 00           {.M..8."..H./.
```

The final packet, above, simply completes the list.

Downloading files that have been found through these searches is not done using the same connection. A new connection must be set up directly between the client and server machines. Once again, a 3-way TCP handshake starts this process:

Source	**Destination**	**Protocol**	**Information**
203.33.188.37	65.31.80.142	TCP	1832 > 6346 [SYN] Seq=17875155 Ack=0 Win=8192 Len=0
65.31.80.142	203.33.188.37	TCP	6346 > 1832 [SYN, ACK] Seq=1297270821 Ack=17875156 Win=16616 Len=0
203.33.188.37	65.31.80.142	TCP	1832 > 6346 [ACK] Seq=17875156 Ack=1297270822 Win=8576 Len=0

Once this handshake is completed, the client can request the file using a standard *http* “GET” request:

```
0000   20 53 52 43 00 00 44 45 53 54 00 00 08 00 45 00     SRC..DEST....E.
0010   00 b7 d6 4a 40 00 80 06 0b 02 cb 21 bc 25 41 1f    ...J@......!.%A.
0020   50 8e 07 28 18 ca 01 10 c0 d4 4d 52 c8 26 50 18    P..(......MR.&P.
0030   21 80 be db 00 00 47 45 54 20 2f 67 65 74 2f 35    !.....GET /get/5
0040   2f 65 6d 61 69 6c 73 20 61 6e 64 20 61 64 64 72    /emails and addr
0050   65 73 73 65 64 20 6f 66 20 70 6f 74 65 6e 74 69    essed of potenti
0060   61 6c 20 6a 6f 62 73 33 2e 36 2e 30 32 2e 64 6f    al jobs3.6.02.do
0070   63 20 48 54 54 50 2f 31 2e 30 0d 0a 55 73 65 72    c HTTP/1.0..User
```

```
0080   2d 41 67 65 6e 74 3a 20 4c 69 6d 65 57 69 72 65    -Agent: LimeWire
0090   20 32 2e 33 2e 31 0d 0a 52 61 6e 67 65 3a 20 62     2.3.1..Range: b
00a0   79 74 65 73 3d 30 2d 0d 0a 43 68 61 74 3a 20 32    ytes=0-..Chat: 2
00b0   30 33 2e 33 33 2e 31 38 38 2e 33 37 3a 36 33 34    03.33.188.37:634
00c0   37 0d 0a 0d 0a                                            7....
```

The ascii text in the right hand column of this packet capture clearly shows that the request has been issued as a standard "GET" request, and using the file index and filename returned in the QueryHit packet. The response to this request is:

```
0000   44 45 53 54 00 00 20 53 52 43 00 00 08 00 45 00    DEST.. SRC....E.
0010   01 2b a4 e8 40 00 6a 06 51 f0 41 1f 50 8e cb 21    .+..@.j.Q.A.P..!
0020   bc 25 18 ca 07 28 4d 52 c8 26 01 10 c1 63 50 18    .%...(MR.&...cP.
0030   40 59 5f 7c 00 00 48 54 54 50 2f 31 2e 31 20 32    @Y_|..HTTP/1.1 2
0040   30 30 20 4f 4b 0d 0a 53 65 72 76 65 72 3a 20 42    00 OK..Server: B
0050   65 61 72 53 68 61 72 65 20 32 2e 34 2e 32 0d 0a    earShare 2.4.2..
0060   43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 61 70    Content-Type: ap
0070   70 6c 69 63 61 74 69 6f 6e 2f 6d 73 77 6f 72 64    plication/msword
0080   0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 68    ..Content-Length
0090   3a 20 32 34 30 36 34 0d 0a 43 6f 6e 74 65 6e 74    : 24064..Content
00a0   2d 52 61 6e 67 65 3a 20 62 79 74 65 73 20 30 2d    -Range: bytes 0-
00b0   32 34 30 36 33 2f 32 34 30 36 34 0d 0a 58 2d 47    24063/24064..X-G
00c0   6e 75 74 65 6c 6c 61 2d 6d 61 78 53 6c 6f 74 73    nutella-maxSlots
00d0   3a 20 31 0d 0a 58 2d 47 6e 75 74 65 6c 6c 61 2d    : 1..X-Gnutella-
00e0   6d 61 78 53 6c 6f 74 73 50 65 72 48 6f 73 74 3a    maxSlotsPerHost:
00f0   20 31 0d 0a 58 2d 47 6e 75 74 65 6c 6c 61 2d 43     1..X-Gnutella-C
0100   6f 6e 74 65 6e 74 2d 55 52 4e 3a 20 75 72 6e 3a    ontent-URN: urn:
0110   73 68 61 31 3a 37 35 57 4c 37 42 4c 48 5a 4b 48    sha1:75WL7BLHZKH
0120   50 4f 35 5a 58 47 58 32 49 44 41 33 55 36 4b 5a    PO5ZXGX2IDA3U6KZ
0130   4e 57 55 35 53 0d 0a 0d 0a                                NWU5S....
```

This is a standard *http* response, with some extra Gnutella information included. This packet is followed by the file contents, contained in a number of subsequent packets. Details of the *http* protocol can be found in RFC 2616 [30].

3.5 Servents Behind Firewalls

A firewall or other perimeter filtering device need not prevent a user from establishing a Gnutella servent within the protected network and being able to communicate effectively with the network. Most firewalls are configured to deny all incoming traffic, except for that on allowed services, such as *http* for the web server or *smtp* for e-mail. Even then, the allowed traffic may only have access to the DMZ. However, internal users are usually considered trusted, and much looser controls are often in place for outbound traffic. In some cases, the internal users may have little or no restrictions on which external ports they can connect to through the firewall. If this is the case, then access to the Gnutella network is almost unrestricted, as the internal servent can establish the required TCP connections out through the firewall, and all traffic in both directions will use these connections. However, if a servent outside the firewall requests a file from the internal servent, then the Push request will be required to send the file out. (Note that this will not work at all if both servents are behind firewalls, unless one of them happens to have an open port through their firewall.)

In many cases, the firewall will be configured to only allow internal users access to certain external services. Common services that internal users are allowed access to are *http* for web browsing, *smtp* for e-mail, *ftp* for downloading files, etc. Port 6346 is highly unlikely to be one of these allowed ports, unless Gnutella traffic has been specifically allowed through, but the configurable nature of the Gnutella servent means that this still does not prevent an internal servent from joining the network. All the user has to do is find the address of an external servent that is listening on one of the ports that are allowed through. For example, if the organisation allows unrestricted web browsing by internal users, then the Gnutella user needs to find the IP address of an external servent that is listening on port 80. Once connected to this servent on this port, the internal servent can exchange data with the rest of the Gnutella network through this connection. Again, if a servent outside the firewall requests a file from the internal servent, it can Push the file out, but only if it is also listening for the out-of-band connection on a port that is allowed through the firewall. (Note that this covert channel will not work if the firewall filters requests based on valid protocol commands. e.g. if the firewall proxies port 80, and filters for only valid *http* command strings, then the Gnutella traffic will be denied access.) [13]

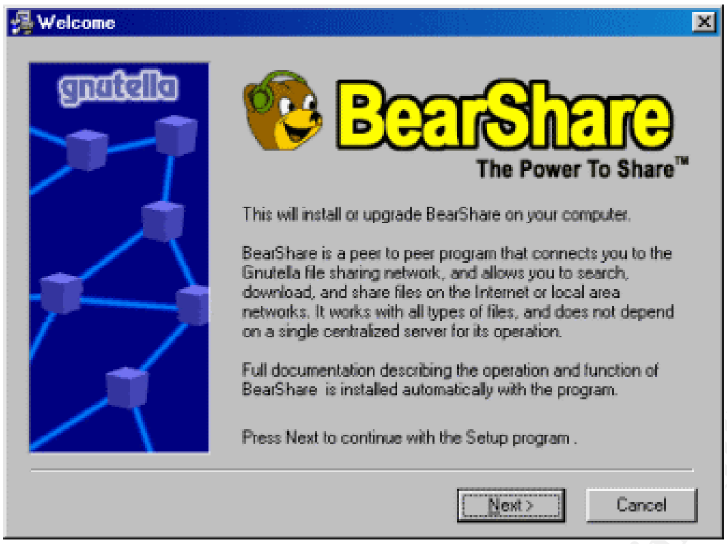
4. Gnutella Servents

Gnutella is an open protocol, which is not platform-specific. The openness allows anyone that wishes to write servent code to be able to do so, as long as they follow the rules for communicating and structuring descriptors, and handling the traffic that passes through them. A large number of servent applications have been written, for most platforms, and they should all be able to communicate on the same network.

Gnutelliums [14] is a directory of most of the currently available Gnutella servent applications. All of the applications listed at Gnutelliums are free to download and install. At the time of writing, Gnutelliums has a list of twenty servent applications available, spread across the Windows, Linux/Unix, Macintosh and Java platforms.

I downloaded two of the more popular Windows based clients, BearShare and LimeWire, in order to test and demonstrate some of the concepts described here.

BearShare



Welcome

This will install or upgrade BearShare on your computer.

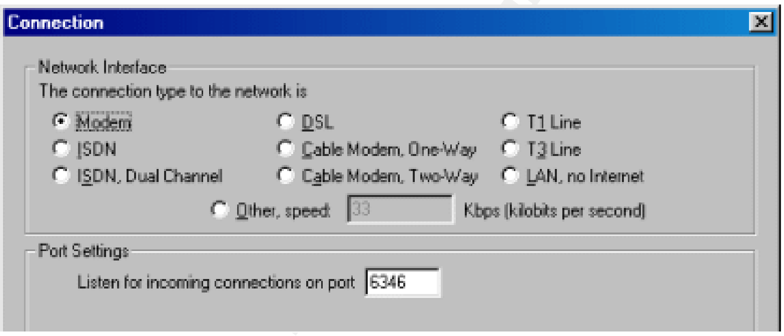
BearShare is a peer to peer program that connects you to the Gnutella file sharing network, and allows you to search, download, and share files on the Internet or local area networks. It works with all types of files, and does not depend on a single centralized server for its operation.

Full documentation describing the operation and function of BearShare is installed automatically with the program.

Press Next to continue with the Setup program.

Figure 4.1 - Welcome to BearShare

BearShare version 2.5.0 is developer by Free Peers, Inc. The installation file is 1.1 Mb in size, and the installation is simple. Figure 4.1 shows a screenshot of the Welcome screen during installation.



Connection

Network Interface

The connection type to the network is

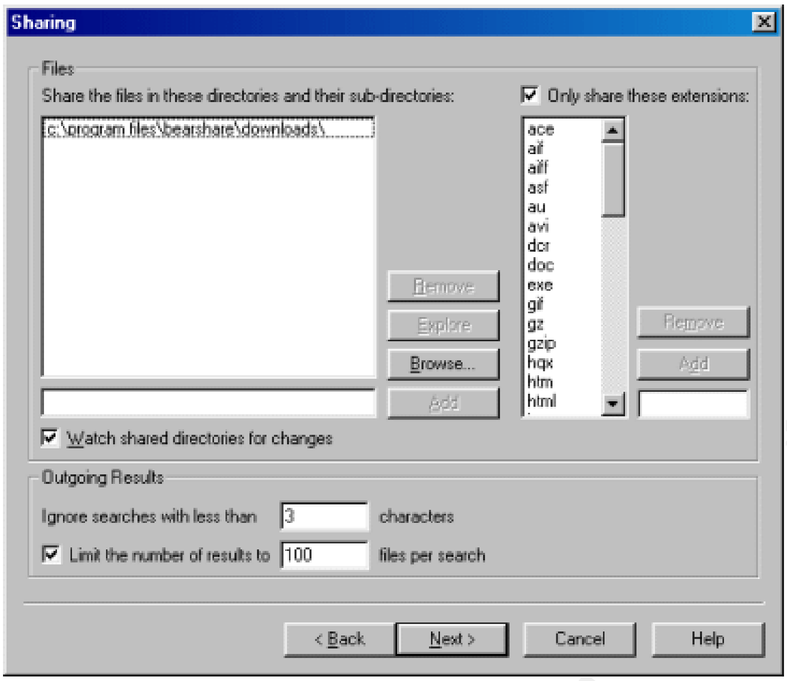
● Modem ○ DSL ○ T1 Line
○ ISDN ○ Cable Modem, One-Way ○ T3 Line
○ ISDN, Dual Channel ○ Cable Modem, Two-Way ○ LAN, no Internet
○ Other, speed: 33 Kbps (kilobits per second)

Port Settings

Listen for incoming connections on port 6346

Figure 4.2 - Configuring the connection in BearShare

Figure 4.2 above shows a screenshot of the configuration screen for the connection. The default port for incoming connections is port 6346, but is completely configurable.



Sharing

Files

Share the files in these directories and their sub-directories:

c:\program files\bearshare\downloads\

☑ Only share these extensions:

ace
aif
aiff
asf
au
avi
dcr
doc
exe
gif
gz
gzip
hqx
htm
html

☑ Watch shared directories for changes

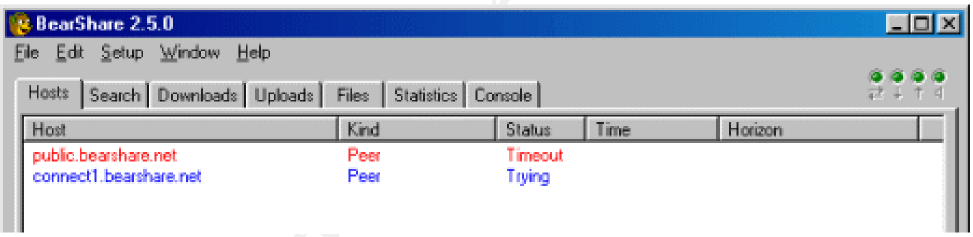
Outgoing Results

Ignore searches with less than 3 characters

☑ Limit the number of results to 100 files per search

Figure 4.3 - Default sharing in BearShare

Figure 4.3 above shows the default shared directory and file extensions in BearShare. The default shared directory is c:\program files\bearshare\downloads\, which is also the default directory for storing files downloaded from other peers. This can be reconfigured at any time.



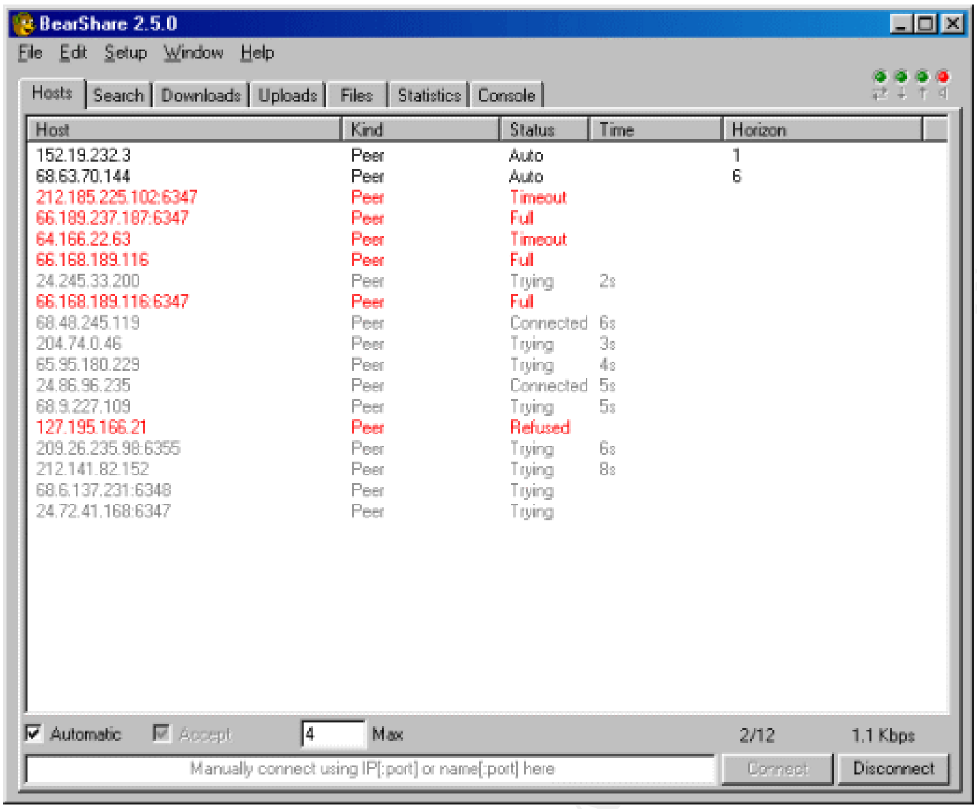
BearShare 2.5.0

File Edit Setup Window Help
Hosts Search Downloads Uploads Files Statistics Console

Host	Kind	Status	Time	Horizon
public.bearshare.net	Peer	Timeout		
connect1.bearshare.net	Peer	Trying		

Figure 4.4 - Connecting to BearShare's host cache

Figure 4.4 shows the default host cache servers that are available for BearShare. Once a successful connection is made to one of these, a list of available peers is downloaded to the local cache, and the connection is terminated. BearShare will try to connect to peers in the downloaded list.


BearShare 2.5.0
File Edit Setup Window Help
Hosts Search Downloads Uploads Files Statistics Console
Connect
Disconnect

Host	Kind	Status	Time	Horizon
152.19.232.3	Peer	Auto		1
68.63.70.144	Peer	Auto		6
212.185.225.102:6347	Peer	Timeout		
66.189.237.187:6347	Peer	Full		
64.166.22.63	Peer	Timeout		
66.168.189.116	Peer	Full		
24.245.33.200	Peer	Trying	2s	
66.168.189.116:6347	Peer	Full		
68.48.245.119	Peer	Connected	6s	
204.74.0.46	Peer	Trying	3s	
65.95.180.229	Peer	Trying	4s	
24.86.96.235	Peer	Connected	5s	
68.9.227.109	Peer	Trying	5s	
127.195.166.21	Peer	Refused		
209.26.235.98:6355	Peer	Trying	6s	
212.141.82.152	Peer	Trying	8s	
68.6.137.231:6348	Peer	Trying		
24.72.41.168:6347	Peer	Trying		

☑ Automatic ☑ Accept 4 Max 2/12 1.1 Kbps

Manually connect using IP[:port] or name[:port] here

Figure 4.5 - Connecting to peers through BearShare

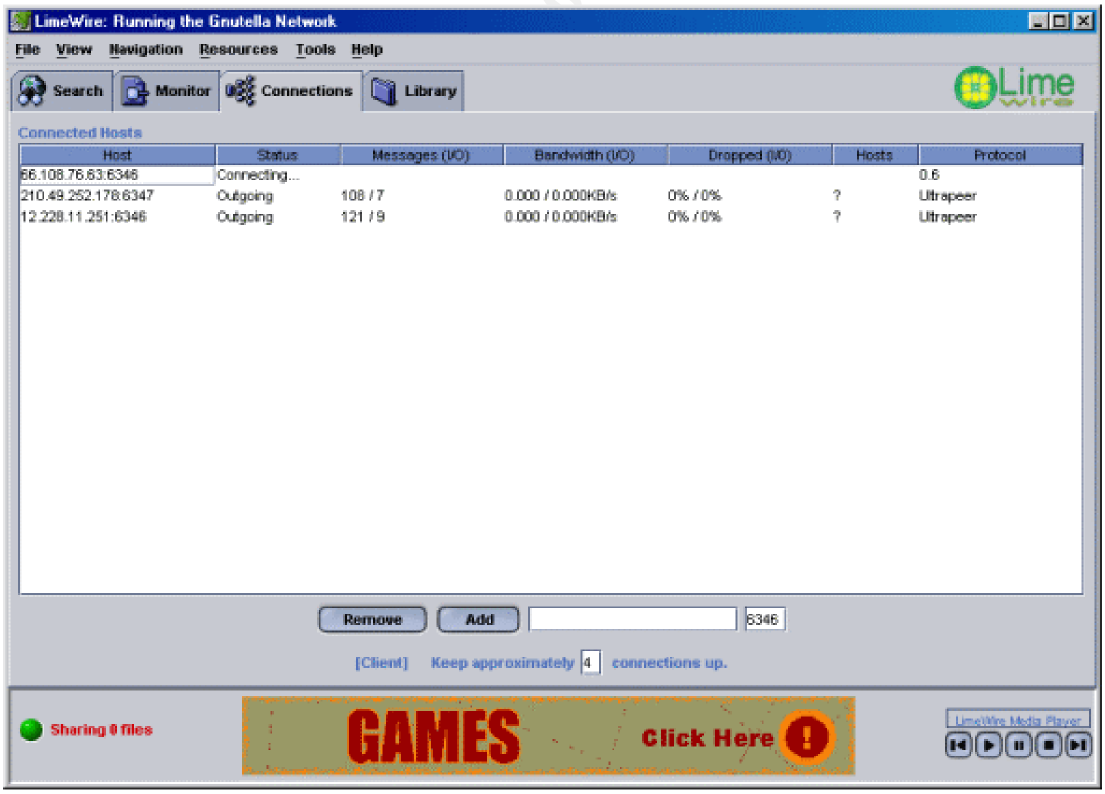
As Figure 4.5 shows, the servent will attempt to open a large number of simultaneous connections, even though the maximum number of connections is set to the default of four. When the maximum number of connections is reached, the remaining connection attempts will be dropped.

Once connected to the peer network, file searching and sharing was a simple process, producing very fast and expansive results to searches, despite being on a slower dial-up connection. BearShare was very easy to install and use, demonstrating the ease with which new users can join the network and share their files.

LimeWire

Figure 4.6 - Welcome to LimeWire

LimeWire version 2.3 is developed by LimeWire, LLC. The installation file is a much larger 3.63 Mb, which makes it less practical to download on a dial-up connection. In addition, LimeWire requires a up-to-date Java Runtime Environment, which required downloading on my system. The JRE was another 5.5 Mb to download. Once all the files finally downloaded, the installation was simple, and when run, LimeWire started up without any problems, presenting the screenshot shown in Figure 4.6 above.



Connected Hosts

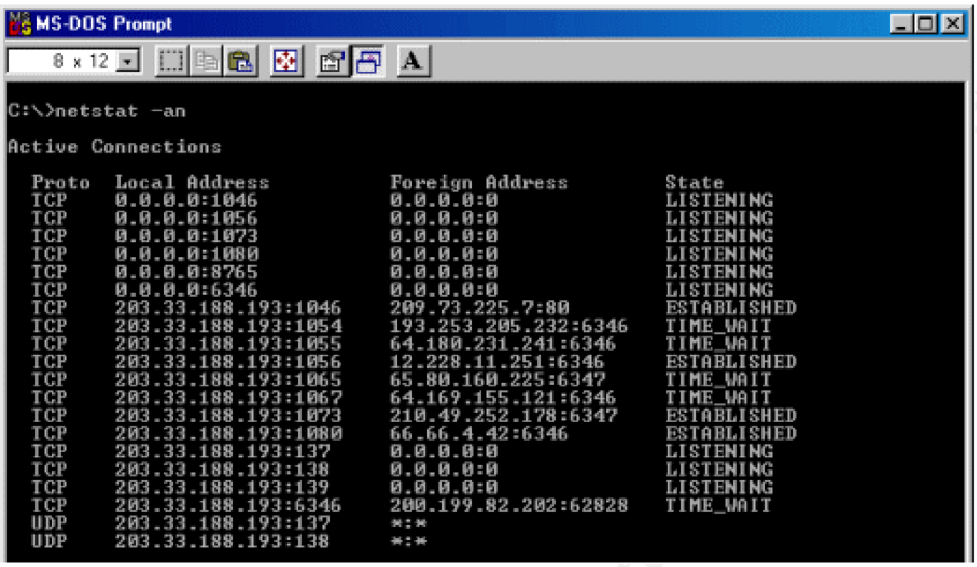
Host	Status	Messages (I/O)	Bandwidth (I/O)	Dropped (I/O)	Hosts	Protocol
66.108.76.63:6346	Connecting...					0.6
210.49.252.178:6347	Outgoing	108 / 7	0.000 / 0.000KB/s	0% / 0%	?	Ultrapeer
12.228.11.251:6346	Outgoing	121 / 9	0.000 / 0.000KB/s	0% / 0%	?	Ultrapeer

[Client] Keep approximately 4 connections up.

Sharing 0 files

Figure 4.7 - Connecting with LimeWire

With its default configurations it is a simple process to get quickly attached to the Gnutella network with LimeWire, as shown in Figure 4.7, and start surfing for files.



```
C:\>netstat -an

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    0.0.0.0:1046           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1056           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1073           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:1080           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:8765           0.0.0.0:0              LISTENING
  TCP    0.0.0.0:6346           0.0.0.0:0              LISTENING
  TCP    203.33.188.193:1046    209.73.225.7:80        ESTABLISHED
  TCP    203.33.188.193:1054    193.253.205.232:6346   TIME_WAIT
  TCP    203.33.188.193:1055    64.180.231.241:6346    TIME_WAIT
  TCP    203.33.188.193:1056    12.228.11.251:6346     ESTABLISHED
  TCP    203.33.188.193:1065    65.80.160.225:6347     TIME_WAIT
  TCP    203.33.188.193:1067    64.169.155.121:6346    TIME_WAIT
  TCP    203.33.188.193:1073    210.49.252.178:6347    ESTABLISHED
  TCP    203.33.188.193:1080    66.66.4.42:6346        ESTABLISHED
  TCP    203.33.188.193:137     0.0.0.0:0              LISTENING
  TCP    203.33.188.193:138     0.0.0.0:0              LISTENING
  TCP    203.33.188.193:139     0.0.0.0:0              LISTENING
  TCP    203.33.188.193:6346    200.199.82.202:62828   TIME_WAIT
  UDP    203.33.188.193:137     *:*
  UDP    203.33.188.193:138     *:*
```

Figure 4.8 - Active connections while running LimeWire

Figure 4.8 above shows a screenshot of the results returned by the Netstat.exe utility in MS-DOS while connected to the Gnutella network using LimeWire. The IP addresses in the established connections shown in Figure 4.7 above can be seen again here in the network statistics shown, as two of the four connections in an "established" state. This shows that a connection has been established, and that connection will stay active for as long as these servents continue to exchange data.

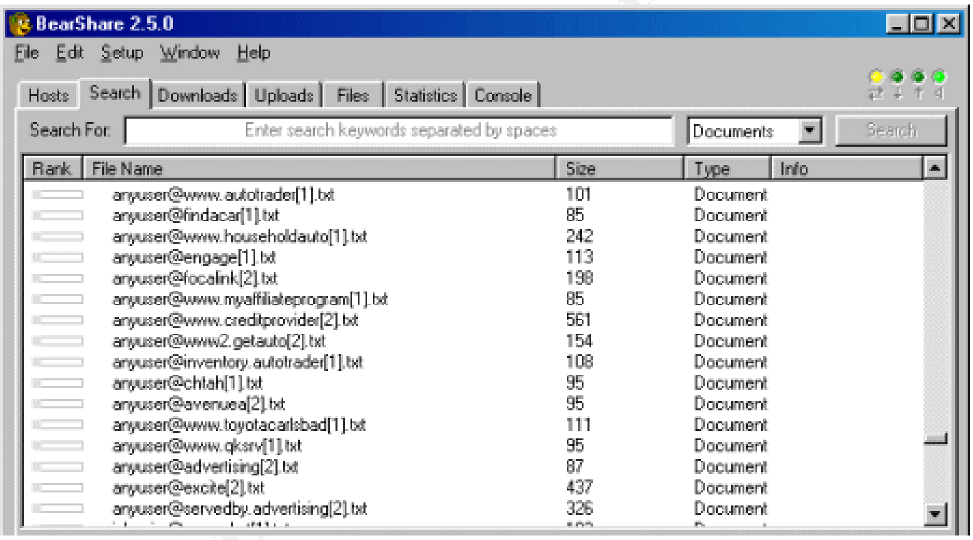
5. Security Issues

Confidentiality

5.1 Exposure of sensitive data

There are a number of known vulnerabilities in the way that peer-to-peer networking is designed. In giving anonymous Internet users widespread access to locally stored files, a user of a system such as Gnutella is opening himself up to the threat of exposing his private or sensitive data to the world. If the system is correctly configured, then this vulnerability can be avoided, but unfortunately many systems are not. Many observers place the blame for misconfiguring systems firmly on the user. As one recent post on the subject to an Internet discussion group says, "If you are dumb as a post, people will take advantage of you. This is the way of the world" [15]. Whilst placing the blame is possibly a reasonable approach to the problem when the vulnerable system is the property of the negligent user, it becomes more complicated when the systems, and the data on them, are the property of an organisation, and not the individual at fault.

The user of the Gnutella servent is responsible for designating files or folders from his local hard drive that can be shared with the Gnutella community. In the case of the servent applications that have been demonstrated, the default folders for this purpose are created at installation, and so no private files should be shared without specific action of the part of the user. Unfortunately, however, many users seem to share a number of folders that would be better kept private. Other application software may, 7by default, share folders that contain sensitive information without the user's knowledge [9]. By initiating a search from a Gnutella client for files with the extensions ".doc" and ".txt", it can quickly be seen that a number of files are returned that should not be shared, such as cookies, log files, shopping lists, resumes and personal letters [16]. Figure 5.1 shows a screenshot of BearShare, showing a small sample of the files returned in response to a search for documents containing the string ".txt":

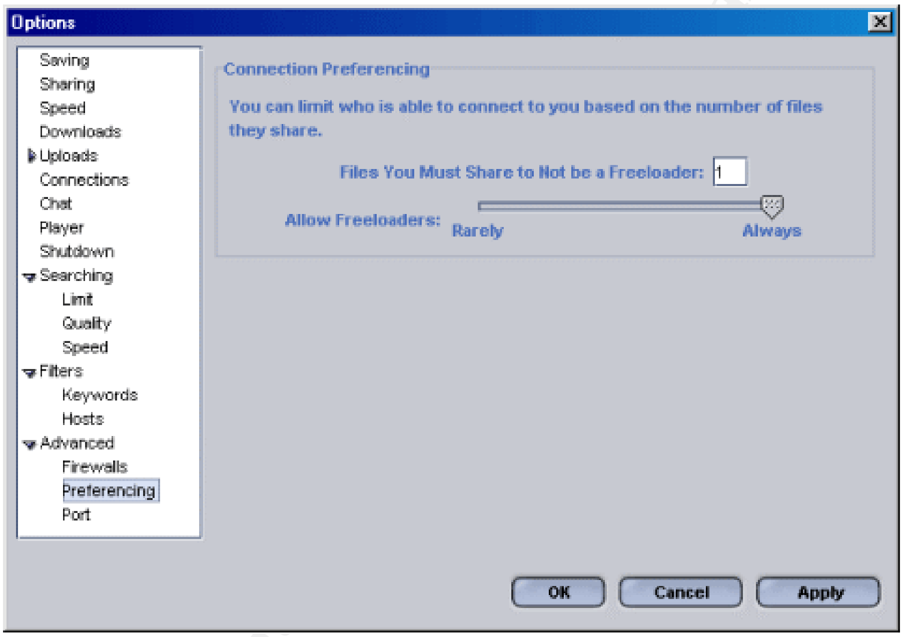


Rank	File Name	Size	Type	Info
	anyuser@www.autotrader[1].txt	101	Document	
	anyuser@findacar[1].txt	85	Document	
	anyuser@www.householdauto[1].txt	242	Document	
	anyuser@engage[1].txt	113	Document	
	anyuser@focalink[2].txt	198	Document	
	anyuser@www.myaffiliateprogram[1].txt	85	Document	
	anyuser@www.creditprovider[2].txt	561	Document	
	anyuser@www2.getauto[2].txt	154	Document	
	anyuser@inventory.autotrader[1].txt	108	Document	
	anyuser@chtah[1].txt	95	Document	
	anyuser@avenuea[2].txt	95	Document	
	anyuser@www.toyotacarlsbad[1].txt	111	Document	
	anyuser@www.qksrv[1].txt	95	Document	
	anyuser@advertising[2].txt	87	Document	
	anyuser@excite[2].txt	437	Document	
	anyuser@servedby.advertising[2].txt	326	Document	

Figure 5.1 - Cookies listed by BearShare

The files shown in the figure above are all Internet cookies, stored on a user's local machine. Some cookies contain personal information stored by a website, including usernames and passwords, and it may be possible for attackers to use these cookies to gain access to sites that the user has visited, such as private web-based e-mail accounts or messages, restricted password-protected sites, etc. These cookies are not intended to be accessible to remote users.

Such files and folders may be inadvertantly shared by users that do not realise the implication of doing so, or they may be shared intentionally by users that believe in the freedom of their information, or even, in the case of corporate networks in particular, they may be shared by malicious users who want the files to be stolen. However, a possibly more likely reason for sharing folders with large numbers of files is to avoid measures put in place to avoid freeloading. When a servent connects to the Gnutella network, it is supposed to be to share files, and not just to copy files from other computers. Servents are expected to join the network with a list of files that they want to share with others, to add to the overall pool of data. It has been established that a large proportion of Gnutella users (approximately 70%) do not share any files with the network [17]. This is known as freeloading, or free riding. To combat this, some servents can be configured to only share files with other servents that also have a certain quantity of files or data available for sharing. LimeWire has this functionality, as shown in the screenshot in Figure 5.2 below:



Connection Preferencing

You can limit who is able to connect to you based on the number of files they share.

Files You Must Share to Not be a Freeloader: 1

Allow Freeloaders: Rarely — Always

Figure 5.2 - Preventing freeloaders in LimeWire

In order to have maximum access to as many files as possible on the network, a servent would need to have a large repository of files in its own shared directories. This creates an incentive for users to share as many files as possible, perhaps by sharing their whole hard drive, or just a folder such as the Temporary Internet Files in a Microsoft Windows based system. Unfortunately, whilst a folder such as this seems harmless, it does contain many cookies. Perhaps the greatest threat for corporations is that a Gnutella user may share their largest file repositories for this purpose, which are likely to contain proprietary data that should not be exposed to the Internet.

5.2 Exposure of system information

As well as using searches to find personal or sensitive information, it is also possible to glean a great deal of information about the system that the servent is running on from the files that can be downloaded. This information would be extremely useful for an attacker, who may be targetting the Gnutella network simply as an information gathering tool, rather than as the final target.

A simple example of this is to search for the file “regedit.exe”. Surprisingly, a number of servents returned positive matches for this file. “Regedit.exe” is the registry editor in all Windows based systems, and the presence of this file gives the attacker some useful information. Assuming that this is not simply a copy of the file placed in another directory, then the attacker can assume that the servent is running on a Windows operating system, and that the Windows system directory may be being shared. If this is that case, then the attacker can target other files in that directory, such as password files. Also, each version of Windows uses a slightly different version of Regedit, and each version is a different size, so the attacker can work out which version of Windows is running on the system from the file size of the returned file. Finally, in the QueryHit packet that returned the file details, the servent included its IP address so that the requestor could download the file. This breaks the anonymity of the Gnutella system, and gives the attacker enough information to begin attacking the remote system.

One of the most basic pieces of system information that an attacker might want to get hold of if the system's IP address. Most Internet attacks begin with a scan of the target host, to find open ports that may be vulnerable to attack. However, before a host can be scanned, the attacker needs to know where that host is, what its IP address is on the network. The Gnutella network, during the course of normal operation, collects, distributes and broadcasts the IP addresses of the active servents in the network. Each servent will maintain a list of other servents, and there are also repositories with cached lists of servents, such as HostsCache.com [12]. Malicious hackers are able to record these IP addresses, which they can then use to attack directly, using “IP address harvesters” [18].

5.3 Copyright and Legal Issues

There may also be legal issues to contend with, in relation to the files that are being exchanged using a corporation's systems. The Napster court case showed that organisations can prove breach of copyright in cases where illegally copied software has been shared on public networks. What is not clear is whether your organisation can be held in breach of copyright if an individual has been using your networks to store and exchange copyrighted material, which may be in violation of the Digital Millennium Copyright Act [7].

Availability

5.4 Resource Utilisation

Peer-to-peer networking is very hungry when it comes to resources, and Gnutella is no exception. The servents generate a huge quantity of traffic, which does a good job of eating a big chuck out of the available bandwidth. Rather than open a communication channel to a server, and then download the required information through that channel, Gnutella servants open numerous channels simultaneously, and pass data on all of them. In addition, it is not only the data generated by your client that is chewing up your bandwidth, but also the forwarded traffic from throughout the network that is passed on by your servent on its way to its destination. Then, once the desired files have been located, the remaining bandwidth is consumed with downloading executables, MP3s, picture files, and so forth, all of which commonly have larger than average file sizes. The drain on your bandwidth will cost the organisation in terms of both money and network performance.

Additional resources that will fall prey to this type of network use are data storage space, possibly also impacting on backup media, and loss of productivity.

5.5 Denial of Service

If you are unfortunate enough to have one of the slower connections on the block, then you may quickly find that the amount of traffic generated on the Gnutella network is sufficient to cause a self-inflicted denial of service. This may particularly be true if the next servent in the chain has a much higher bandwidth, and can generate a lot more traffic than your networks can handle.

Whilst this may occur due to high load on the Gnutella servents, it could equally be caused, or at least accelerated by a malicious servent with a high-speed connection to the network. Since it is not possible to verify the integrity of a packet (which will be discussed in the next section), a malicious servent could easily spoof large numbers of search queries, in an attempt to slow down or stop a section of the network [19]. Spoofed messages of this nature would be difficult to detect, and almost impossible to trace.

Integrity

5.6 Data Integrity

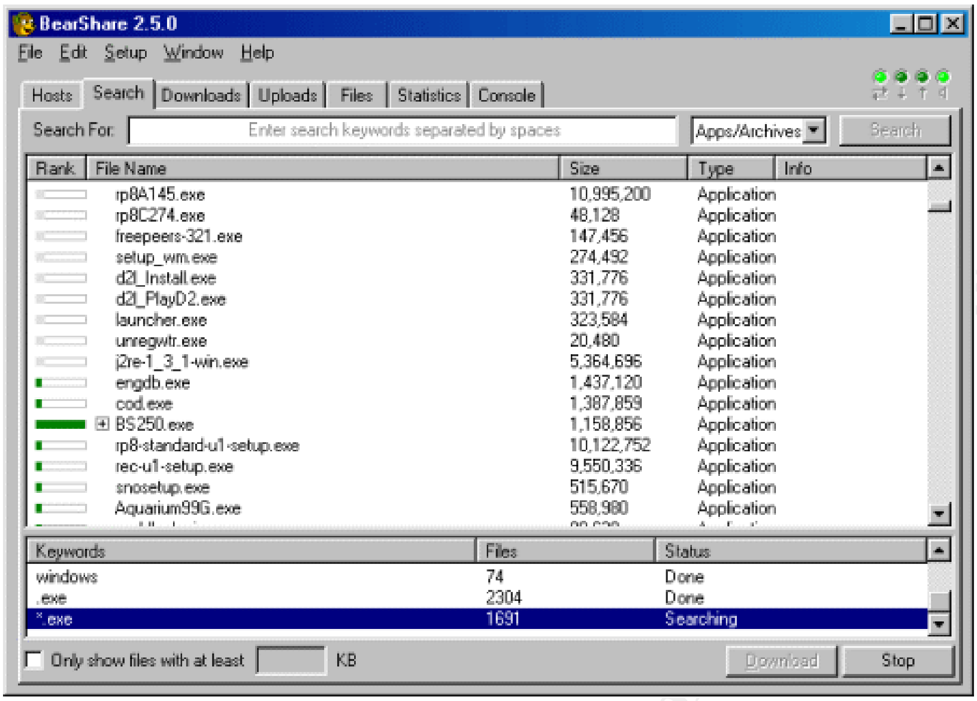
One of the other main causes of vulnerabilities in the Gnutella network that leaves it open to abuse is the openness of the protocol itself. The protocol relies on a trust model, in which legitimate users host valid files from their servents, and servents routing packets within the network adhere to the rules for changing some fields in the headers, such as the TTL and Hop Count, but do not change any other data in the packets. There are two major flaws in this model: there is no way to verify the integrity of the data flying around the network, and the protocol is completely open, so that a malicious user that wishes to alter data has all the information he needs to do so; and the built-in anonymity in the system makes it almost impossible to single out those individuals that choose to abuse the rules, so there is no simple way to prevent them from doing so. In fact, it could be argued that trust and anonymity should be mutually exclusive, and yet the Gnutella network continues to operate very effectively, which seems to disprove this theory.

Any servent within the Gnutella network could change the data in the packets that it handles. This could, for example, enable a malicious user to respond to all search requests with a positive match for the file the requestor is looking for, and then when the requestor connects to download the file, generate a trojaned file that matches the request, but with malicious content. A variation on this would be for the malicious servent to intercept a QueryHit packet, replace the IP address with its own, and send it on its way. It could then download the target file, alter it to contain some malicious code, and then serve it to the requestor when they connected to retrieve the file [20]. Methods similar to these can be used to spread both malicious software and advertisements, neither of which are wanted by the requestors.

5.7 Malicious Software

There are many ways that viruses and trojans can spread through the Gnutella network. Once again, this is a deficiency in the trust model adopted by peer-to-peer networks. Gnutella is more prone to the spread of malware than other networks, such as Napster, because the files that can be shared are not limited to certain types, such as ".mp3" files, but can have any extension.

Figure 5.3 below shows a screenshot from BearShare, with a small set of the results to a search for files containing the string ".exe":


BearShare 2.5.0
File Edit Setup Window Help
Hosts Search Downloads Uploads Files Statistics Console
Search For:
Enter search keywords separated by spaces
Apps/Archives
Search
☐ Only show files with at least KB
Download
Stop

Rank	File Name	Size	Type	Info
	rp8A145.exe	10,995,200	Application	
	rp8C274.exe	48,128	Application	
	freepeers-321.exe	147,456	Application	
	setup_wm.exe	274,492	Application	
	d2l_Install.exe	331,776	Application	
	d2l_PlayD2.exe	331,776	Application	
	launcher.exe	323,584	Application	
	unregwtr.exe	20,480	Application	
	j2re-1_3_1-win.exe	5,364,696	Application	
	engdb.exe	1,437,120	Application	
	cod.exe	1,387,859	Application	
	⊞ BS250.exe	1,158,856	Application	
	rp8-standard-u1-setup.exe	10,122,752	Application	
	rec-u1-setup.exe	9,550,336	Application	
	snosetup.exe	515,670	Application	
	Aquarium99G.exe	558,980	Application	

Keywords	Files	Status
windows	74	Done
.exe	2304	Done
*.exe	1691	Searching

Figure 5.3 - Executable files listed by BearShare

As the figure shows, there are a huge number of executable files being shared on the Gnutella network, and a huge variety of different files, including the installation files for the servent software itself. This small subset was returned after only a few seconds of searching. Any one of these files could be a virus or trojan, just waiting for a user to download and run it before it delivers its payload. Due to the nature of the network, it is very difficult for antivirus software to effectively prevent infection, especially when the user is actively seeking out these files to download and execute. The range of file types that can be infected is limited only by the types of file that can deliver their payload on execution. This includes executables, batch files, visual basic scripts, documents with malicious macros included, media files that can exploit a buffer overflow in the player, and any other type of file that has been found to be, or will be found to be, susceptible.

Because the Gnutella protocol is completely open, anyone can write their own servent software, and could easily include some malicious content in the code. Also, as we saw above, the servant applications are often made available through the network, and could be infected with a trojan. The widespread use of Gnutella servents makes them an ideal medium for the spread of malicious software, as they are designed to be installed throughout the Internet, and to communicate freely with each other. The idea has been proposed [20] that a network of malicious servents could be established, which could be controlled via a covert communication channel enabled by using specially crafted search strings, or additional data embedded in the unique servent identifier fields of the descriptors. In much the same way that recent malicious software agents have communicated via IRC chat channels, these servents could be used to launch synchronised attacks across the Internet, similar to the recent DDoS attacks. An example of unwanted code being installed with the servent application is the DlDer trojan, a spyware application that was installed with one of the servents and reported user information to an external website. DlDer is described below in the section on specific vulnerability examples.

5.8 Specific Vulnerability Examples

Malware

Name: VBS/GWV.a [21]
Alternate names: Gnutella.worm, VBS/Gnu, VBS/Gnutella.worm
Discovery date: May 30, 2000
Platform affected: Windows
Description: The VBS/GWV worm is a Visual Basic script that is designed to propogate through the Gnutella network. When run, the worm writes a copy of itself to the local hard drive using a randomly chosen name from a list of file names that are likely to meet common search criteria (see the McAfee information page for the full list [21]), deletes the original file, and makes two changes to the Gnutella.ini file: it adds ";vbs" to the value "extlist=" and "C:\Program Files\gnutella" to the value "databasepath=". These changes allow the servent to share VBS scripts, and makes sure that the Gnutella directory is in the list of shared folders. It then also creates a ZIP file in the shared folder, which allows the author to track its spread. Note that the worm still requires another Gnutella user to download and run it to propogate. There is no malicious content.

The VBS script includes the following fields [22]:
ProgramName = "Gnutella Worm v1.2 By LeGaLiZeBuDz"
ProgramDate = "2000 May 28. The first v1.2 Gnutella Worm."

Name: W32/Gnuman.worm [23]
Alternate names: GnutellaMandragore, Gspot trojan, Mandragore, TROJ_MANDRAGORE, W32.Gspot.Worm, W32/GnutellaMan, W95/Gnuman.A
Discovery date: February 26, 2001
Platform affected: Windows
Description: The W32/Gnuman worm is an executable file, with a “.exe” extension, that is also designed to spread through the Gnutella network. When run, the file writes a copy of itself to the StartUp folder as a hidden file, with the

name GSPOT.EXE, so that it will be reloaded each time the system boots. The worm will then respond to each Query packet that comes to the infected servent with a file that exactly matches the search string, with a ".exe" extension [24]. A copy of the worm will be served to the Query sender, if it is requested, with this filename and with a file size of 8,192 bytes [25]. Note that the worm still requires another Gnutella user to download and run it to propogate. There is no malicious content.

Name: DlDer [26]
Alternate names: Trojan.Win32.DlDer, Troj_DlDer
Discovery date: December 28, 2001
Platform affected: Windows
Description: The DlDer trojan is an executable file, with a ".exe" extension, that is installed with the LimeWire, Kazaa and Grokster peer-to-peer servents, among other packages. It is part of an adware component of the software, but it was classified as a trojan on account of its behavior. Users could elect not to install additional components during installation, but even if they chose not to, the component was installed anyway. DlDer.exe is installed in c:\windows, and after installation it runs and downloads another executable file, Explorer.exe, to c:\windows\explorer, and creates two registry keys, *HKLM\Software\games\clicktilluwin* and *HKLM\Software\Microsoft\CurrentVersion\Run\dlder*, to reload the components at system startup [27]. At next system startup, Explorer.exe is executed, connects to an external website, and sends user information and web-browsing history. This trojan is not malicious, and does not propogate, but it is spyware, reporting personal information to an external entity. Current versions of the listed software do not include this spyware.

Vulnerability

Name: Gnut Gnutella Client Arbitrary Script Code Execution Vulnerability [28]
CVE name: CAN-2001-1004 (under review) [29]
Discovery date: August 30, 2001
Platform affected: Windows and Linux
Version affected: Gnut, version 0.00.4.27 and earlier
Description: Cross-site scripting (CSS) vulnerability in gnut Gnutella client before 0.4.27 allows remote attackers to execute arbitrary script on other clients by sharing a file whose

name contains the script tags [29].

Gnut is a console-based Gnutella servent, with a web interface for executing searches and viewing results. When the search results are returned, the web interface does not strip *html* tags from the filenames. An attacker could craft a filename that included script code, that would then be run by the web interface application, and may be able to gain unauthorised access to resources on the local machine [28]. There are no publicly known exploits for this vulnerability, and the vulnerability was fixed in version 0.00.4.28.

6. Threats to Corporate Networks

The Gnutella protocol has become widespread in recent months, and we have seen that many organisations have been producing and providing servent applications to the public. A huge number of Gnutella servents are busy exchanging files across the Internet, and for this reason it should be little wonder that a large number of requests to the Gnutella default ports are being picked up on intrusion detection systems around the world.

From examining the protocol and the way that servents interract with each other, we have seen that legitimate hosts only connect to known servents, using known or cached address and port information. Many home users have experienced the problem of being plagued by connection requests from remote hosts trying to connect to Gnutella ports, when they have not installed servent software on their machines nor had any connection with the Gnutella network. This is often due to the user being connected to the Internet via a dial-up modem, which uses a dynamically assigned IP address each time it connects to the ISP's servers. When the modem disconnects, that IP address becomes available to be assigned to other dial-up users. If a previous user of an address was using Gnutella software, then that IP address will be stored in other servents' host caches, and those servents are likely to attempt to connect to that address.

However, corporate networks are likely to be responsible for a large proportion of the firewall and intrusion detection data sent to the Internet Storm Centre, yet corporate networks are unlikely to be connecting to the Internet through dynamically assigned IP addresses. If these organisations are also seeing large numbers of probes against these ports, then this raises the question of how an organisation's static IP addresses have found their way onto cached host lists.

It is reasonably unlikely that attackers, or Gnutella users, are probing port 6346 on a large scale, since there is little to be gained in doing so. The only service likely to be listening on that port is the Gnutella servent service, and that could easily be listening to any other port, and if it is running it will be actively advertising the fact to the network. The only advantage in probing would be the remote possibility of breaking into a private Gnutella network, which the users have decided not to connect to the wider Gnutella network.

It appears reasonably likely, therefore, that if an organisation's perimeter defences are seeing a lot of traffic on port 6346, then at some time (probably recently) those networks have been attached to the Gnutella network, which suggests that somebody in the organisation has installed servent software. We know from the previous discussions that corporate data and resources are at risk from this, if it is not correctly configured, and that Gnutella can bypass firewalls, so we must consider this to be a fairly serious breach of security.

6.1 Insider Threat

A number of recent surveys have suggested that by far the biggest threat to corporate networks comes not from anonymous hackers somewhere outside the network, but from insiders - legitimate users within the network. Those attackers can take a number of different forms, such as a disgruntled employee that intentionally wants to cause harm to the organisation, or a naive or careless employee that accidentally damages or endangers the networks. Either of these could threaten the organisation's resources through the misuse of a Gnutella servent.

The organisation is particularly at risk if it does not have a clear policy, either on the specific use of peer-to-peer clients, or generally on the installation of unknown software. Unless a policy strictly forbids installation of such software, the disgruntled employee or the innocent user may feel that they are within their rights to install the software on corporate systems, and they may be correct. If peer-to-peer clients are allowed on the corporate networks, then a clear policy needs to be in place to govern its configuration and operation, as there are few other methods available to prevent misuse.

Once the software is installed, the greatest risk to the organisation comes from the way that it is set up and used. Probably the biggest risk comes from the choice of which files and folders are shared. The malicious user will probably wish to share as many sensitive corporate files as possible, in an attempt to embarass or discredit the corporation, or destroy their competitive advantage. The naive user may just want to share as many files as possible in order maximise their access to other servents' files, or just because they believe in sharing information. This could expose corporate data, cookies with sensitive passwords, or other information useful to potential attackers.

The threats are not only from data leaving the organisation. If the users choose to download copyrighted or objectionable material to the corporate networks, then the corporation may again be embarassed or discredited, or even find themselves the target of legal action. Costs can be incurred through inappropriate use of bandwidth and other resources, both in terms of money and lost productivity if those resources were needed elsewhere. Again, a naive user may inadvertantly incur these costs, or not realise the significance, whilst a malicious user could intentionally cause damage by running up costs, overloading resources, or introducing malicious software to the networks.

These threats all have one thing in common; they rely on having an insider take the actions that leave the networks vulnerable. The motivation of the attacker may be different in each case, but the consequences are often the same.

7. Exploit Scenario

In this section we will examine an example of one way in which a malicious attacker could exploit the described vulnerabilities of the Gnutella protocol to attack a target system.

In this example it is assumed that the attacker has specifically targetted a particular corporate systemwhich is protected behind a firewall. It is also assumed that a legitimate inside user within the target network has installed a Gnutella servent in order to share files, without malicious intent. The installation of the servent package may be in breach of the corporate policy, allowed by the corporate policy, or simply not covered sufficiently by the policy, but it is assumed that management have not specifically authorised this application, and that administrative staff are not monitoring the traffic.

7.1 User Actions Leading to Compromise

For the attacker to be able to make use of the Gnutella protocol as the attack vector in this example, the first requirement is that there is a Gnutella servent installed within the target network, and that the servent has opened a connection to another Gnutella servent outside the corporate network. In this example, we are assuming that the legitimate network user has installed the servent software and intentionally connected to another servent in order to share files.

The corporate firewall in this example is confugured to not allow traffic through in either direction using the default ports of 6346 and 6347, although it does allow unrestricted access to the Internet from the internal network on ports 25 for e-mail, and port 80 for Web browsing, neither of which use a proxy in this case. This is certainly not an unusual situation in corporate networks.

In order to be able to communicate with the external Gnutella network the user has configured his Gnutella servent to communicate using port 80 to another servent on the Internet that is listening on this port. The user must supply the address of this next node manually, as connection to most servents on cached address lists requires access to the default ports. The user could supply a single address to connect to, or a number of addresses of servents listening on this port. Either option would give him access to the network. Having only a single connection to the network does not affect the performance greatly, but simply reduces the number of hosts that see each query. This will only have a significant effect for small TTL values.

As we have discussed previously, the Gnutella protocol is designed to work effectively in this manner, passing data through a firewall without affecting connectivity to the network. When the inside servent attempts to connect to an outside host using port 80 the firewall will see the traffic as a standard *http* connection request, and will allow the connection. Once the connection is established it will remain open for the duration of the operation of the servent (assuming that it is not terminated by the other host or by network problems), and all traffic between the two hosts will be passed using this open connection. When outside hosts wish to download a file from the servent behind the firewall, however, they will be unable to open a new connection to the host, as it will be blocked by the firewall, and will instead have to send a Push request. In this example, for the Push to be successful, the requesting host will have to be listening on ports 25 or 80 to allow the connection to again pass through the firewall.

The other action that the user must take before he can connect to the Gnutella network is to choose which files he will share with the Gnutella network. This step has been discussed in some detail in this document, and is the primary cause of sensitive data becoming vulnerable to theft from networks through the use of Gnutella. However, the malicious attacker may not have to rely entirely on the unsuspecting user to provide him with the desired content; the VBS/GWV.a worm, described above, proved that an executable file (in this case a VBS script) could easily make changes to the configuration files for the Gnutella servent, including changing or adding to the shared folders. The attacker could write some code that added the desired directories to the list of shared folders in the user's configuration files, and then append the code as a trojan to an executable file that the user was attempting to download, perhaps using the methods employed by some of the malware examples given above. In order to do this, the attacker is likely to need some knowledge of the platform being targetted and the servent software being used, as the configuration files are different for each. For example, BearShare stores configuration data in the text file “FreePeers.ini” in the BearShare installation directory, and Limewire stores configuration data in the text file “Limewire.props” in the “2.3.1” directory under the installation directory. Each can be changed by simply adding a directory name in the correct place in the file.

However it happened, we can assume in this example that the user's Gnutella servent is sharing some sensitive corporate files with the network, and that a malicious attacker will wish to target those files.

Once the user has completed these basic tasks, he can connect to the Gnutella network and begin to share files. Once he has connected to the network and established a connection through the firewall, any other Gnutella user on the network can search for and download files from his shared directories.

7.2 Finding the Target

The opportunist attacker now has everything that he requires to be able to steal corporate information from the target computer. Any Gnutella user that is connected to the same Gnutella network as the corporate user will be able to receive search responses from him, provided that the files on the shared folders match the search criteria, and that the connection between the user and attacker goes through less nodes than the attacker's initial TTL value. The opportunist attacker might do a search for any files in the network that match a generic criteria that might reveal sensitive information, such as files with the extension ".doc" or, as we have seen demonstrated, searching for files with a ".txt" extension to reveal information such as cookies. Responses to such queries might come back to the attacker from anywhere in the network, including the target machine behind the firewall. This type of query cannot be targetted at a particular host, however.

It is more difficult for an attacker to target a particular servent in the network, as the protocol does maintain a degree of anonymity for intermediate nodes, and does not allow a user to specify which nodes to query for information. The attacker will have to perform some reconnaissance of the surrounding Gnutella network in order to find the target servent that he is looking for.

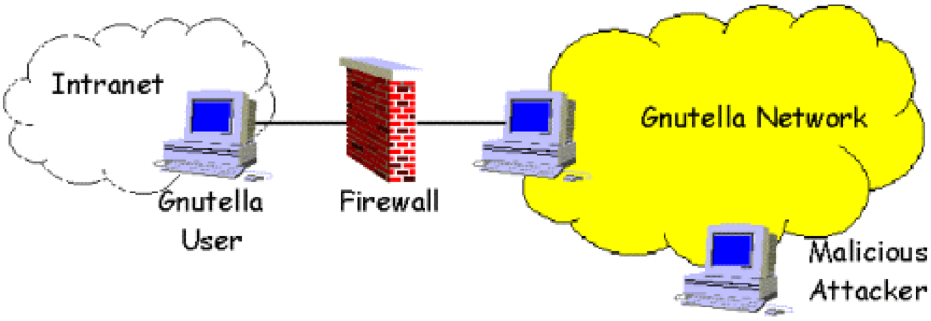


Figure 7.1 - Malicious attacker connected to host through Gnutella network

The Pong descriptor and the QueryHit descriptor both return IP address and port information from the responding servents. The attacker can use this information to map the nodes around him, filtering the traffic to look for the target's IP address. Ping requests could be used to initially identify the target and confirm that it is active on the network, although this could be prone to errors as many of the implementations of the servent software rely on the use of cached address lists rather than a Pong reply from every node that is within reach of the Ping. Another way would be to issue a Query packet which contains a very generic search request that is likely to be matched by most of the servents that receive it. The resulting QueryHit packets will contain the IP address and port information for the replying servents, along with the number of hops made to send the reply. This information could be used to map the surrounding network, although it would not provide information on the routing of descriptors through the network.

One way for the attacker to then scan the target host for files would be issue generic Query packets in this manner, and then to accept all the incoming replies and filter out the responses that came from the desired target host. The main disadvantage of this approach is that it is likely to illicit a huge number of responses, which might overload the attackers systems, especially if the target host is a large number of hops away.

A more effective means of attack would be for the attacker to attempt to map the route between himself and the target host, in order to connect to a servent much closer to the target. This enables him to send out Query packets with a much smaller TTL value so that fewer Gnutella servents need to be queried to ensure that the target receives each Query sent. This could be achieved by the attacker first issuing a very generic search request with a high TTL, in order to identify the target host by filtering the responses based on IP address. The QueryHit packet received from the target will tell the attacker how many hops away the target is, as well as possibly telling him some of the filenames that the target has shared. The attacker could then disconnect from the network and reconnect to one of the servents that was one hop away, and send a Query for a filename that is known to be hosted by the target. If the servent is in the route to the target, then the returned QueryHit packet will have a lower hop count than before, which puts the attacker one hop closer to the target. By repeating this process the attacker could eventually connect his Gnutella servent to the servent that is immediately outside the target's firewall. This last intermediate servent is directly connected to the target via the port 80 connection through the firewall. All queries sent to this servent will still be sent to the whole Gnutella network, but the attacker now has the advantage of being able to use a TTL of just 2 on all Query packets, and can be sure that they will reach the target, whilst reaching a minimal number of other hosts. The other great advantage that this type of connection has is that, as long as the attacker configures his system not to respond to Ping or Query packets, then the target system will not have any record of the attackers IP address, as all packets are forwarded by the intermediate servent.

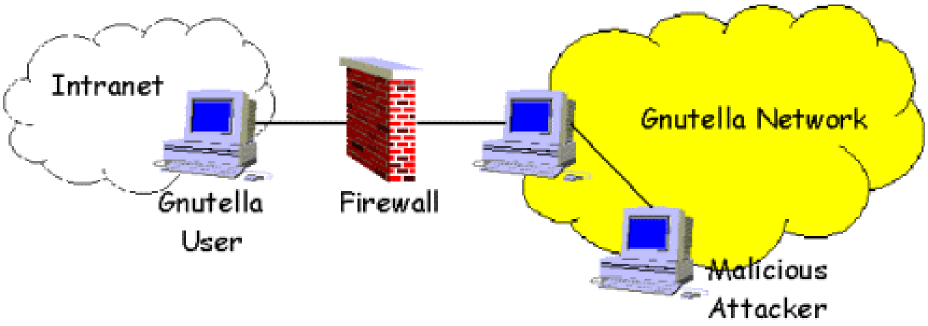


Figure 7.2 - Minimising the number of hops between attacker and victim

7.3 Searching For Data

Once the attacker has established a connection that is acceptably close to the target, he can begin scanning the target servent for files. It is not possibly to request a full file listing from another servent, but it is a trivial process to generate queries that will match with the maximum number of files. Simple queries that match with known file extensions are one way to achieve this. Most servents are configured by default to ignore queries with less than 3 characters in the search criteria, which leaves 3 character file extensions as an easy option to work on. The specific file extensions that the attacker might target will depend greatly on the type of data that he is looking to steal.

Another tool that the attacker has is the Pong descriptor. The attacker can Ping the network, and will get a Pong response from each servent that the Ping reaches which includes, among other information, the number of files and the total quantity of data shared. This lets the attacker know how hard to keep looking for files.

Once again, it is important to note that each Query sent by the attacker will illicit responses from a number of servents that can match the search, but by keeping the TTL value in the outgoing queries as low as possible the attacker is able to minimise the amount of traffic returning.

7.4 Downloading Files

Due to the nature of the Gnutella protocol, gaining access to another machine is achieved simply by being part of the network. The actions discussed so far will allow the attacker to list the files that the user is sharing with the Gnutella network. The next stage is to download the desired files.

The attacker will be unable to connect directly to the target servent due to the firewall which is protecting the target's network. The attacker will, therefore, have to send a Push descriptor, in order the request that the target computer opens a channel through the firewall from the inside for the file to be downloaded. The Push descriptor contains the IP address and port of the requesting machine, so that the target servent knows where to connect to.

This will be the first time that the attacker has had to send an IP address to the target machine, and he may wish to maintain his anonymity, so that his IP address does not get entered into the firewall logs when the inside servent tries to connect. This could perhaps be done by installing a Netcat relay on another Internet machine, which listens on the configured port, issues the "GET" request and relays the file to the attacker.

7.5 Tools Required by Attacker

All of the attack methods mentioned here can be performed using standard Gnutella commands, with a standard servent application. The attacker is relying on the nature of the protocol, and not special software or source code. Most configuration changes that an attacker might make to the servent software, such as varying TTL values, can be done manually through the configuration files. The protocol is platform independent, and servent applications exist on many different operating system platforms.

The open nature of the protocol does allow anybody to write servent software, however, and many servent applications have been made available through the open-source community. One example of this is Gnut, a command line Gnutella tool that is available as source code, to be compiled on many different platforms. An attacker could make use of this source code to modify the software to meet his needs. The source code for Gnut is freely available from Gnutelliums [31].

8. Network Defences

Due to the nature of the Gnutella protocol, its flexibility with respect to which ports it can use, and its ability to be able to pass through firewall defences, preventing a determined user from using the Gnutella protocol is not easy. Clearly, blocking ports 6346 and 6347 at the perimeter is not going to be enough to keep Gnutella out of your networks.

The two most effective tools available to system and security administrators to mitigate the risks presented by peer-to-peer networks such as Gnutella are awareness and security policy.

Of course, organisations may decide that peer-to-peer file sharing fulfills a business need, or that users should be allowed the freedom to choose which software is right for them. It is the responsibility of individual organisations to weigh up the risks involved, and to then define a security policy that suits their needs.

8.1 Maintaining a Gnutella-free network

If your organisation decides that the risks are too high, or there is simply no need to have it, then a sensible policy is to keep Gnutella out of the networks altogether. There are various measures that an administrator can take to enforce this policy, and a few suggestions are given here:

Make sure that the policy is clearly stated, and make sure that users are aware of the policy. A policy that disallows installing any executable files on corporate systems, unless done by an administrator, helps prevent servents being installed and also helps prevent virus propogation.

Make sure that users are aware of the risks.

Block all unnecessary ports at the firewall, both incoming and outgoing. Install a proxy firewall if possible for those services that are needed.

Occasionally audit workstations to check for new applications.

Configure your IDS to trigger an alert if it detects packets with "GNUTELLA CONNECT" or "GNUTELLA OK" in the payload. An example of a pair of Snort IDS rules that could achieve this might be:

```
        alert tcp any any <> $local_net any (content: "GNUTELLA
CONNECT"; \
                msg: "Gnutella connection attampt";)
        alert tcp any any <> $local_net any (content: "GNUTELLA OK"; \
                msg: "Gnutella connection attampt";)
```

The IDS could also be configured to alert if it detects packets coming into the network with "GET /get" in the payload, provided there are no Web servers in the intranet to create false positives. An example Snort rule might be:

```
        alert tcp any any -> $local_net any (content: "GET /get/"; \
                msg: "Inbound GET request";)
```

Monitor traffic passing through the firewall, and look for odd patterns. Examples are large numbers of simultaneous connects (>10) from a single host, or long-lasting (hours?) continuous connections on ports 80 or 25.

8.2 Reducing the Risks of Using Gnutella

If the decision is made that Gnutella is required, or desirable, in the corporate networks, then a number of steps need to be taken to reduce the risks. A few suggestions are given here:

Again, state the policy very clearly, and make sure users are aware.

Only allow approved clients to be installed.

Enforce clear separation between sensitive files and shared files.

Only allow experienced users or administrators to configure the shared folders.

Develop a strong policy on misuse.

If you have a closed Gnutella peer group within the network, make sure users are aware that a single peer connected to an outside network opens the whole network to the Internet.

Provide a pool of non-sensitive public files to share, so that users are not tempted to share additional company files.

Audit writing of files into the shared folder, and hold users accountable.
Make sure that current antivirus software is activated on all machines involved in the peer group.
If using the Gnut command line servent application, ensure that the version installed is 0.00.4.28 or later.
Audit the servent configurations to check for changes.
Restrict access to the software to the minimum number of hosts possible, and then consider separating these hosts from the main network.

9. Conclusions

Peer-to-peer networking protocols, such as Gnutella, have gained a great deal of public favour recently, and it shows no sign of diminishing. Unfortunately, though, the openness that makes it so successful also introduces a number of security vulnerabilities.

Users and administrators need to be aware of how Gnutella works, and the risks that they are exposing themselves to by running these file-sharing applications on their systems. In particular, the Gnutella protocol has been designed to allow it to operate despite normal network protections, so firewalls alone are not a sufficient measure to prevent Gnutella from crossing the network perimeter.

It is beyond the scope of this assignment to decide whether the Gnutella protocol and associated applications are a good or a bad thing. That is a decision that must be made by the appropriate responsible people in each organisation. However, I have attempted to present sufficient information to assist those making that decision, and provided suggestions for ways of mitigating the risks. The best defences against the vulnerabilities are awareness and a really good security policy.

References

All the references given are active Internet links at the time of writing.

[1] The SANS Institute, *Cyber Defense Initiative*
- http://www.sans.org/CDI.htm

[2] Incidents.org home page
- http://www.incidents.org

[3] Incidents.org, *Letter from the Director*
- http://www.incidents.org/mission.php

[4] DShield.org home page
- http://www.dshield.org/

[5] DShield.org, statistics for Australasia, March 8, 2002
- http://www1.dshield.org/country_list.php?continent=AU

[6] Internet Assigned Numbers Authority, *Port Numbers*, March 16, 2002
- http://www.iana.org/assignments/port-numbers

[7] Rob Harmer Consulting Services Pty Ltd, *Have you been Napstered...........?*, AuditNet.org, March 4, 2001
- http://www.auditnet.org/articles/have_you_been_napstered.htm

[8] Peer-to-Peer Working Group, *What is peer-to-peer?*
- http://www.peer-to-peerwg.org/whatis/index.html

[9] Billy Evans, *Peer-to-Peer Networking*, SANS Reading Room, Oct 29, 2001
- http://rr.sans.org/threats/peer2.php

[10] Gnutella News, *What is Gnutella?*
- http://www.gnutellanews.com/information/what_is_gnutella.shtml

[11] Clip2, *The Gnutella Protocol Specification v0.4, Document Revision 1.2*
- http://www.clip2.com/GnutellaProtocol04.pdf

[12] HostsCache.com
- http://www.hostscache.com

[13] Oliver Kuhl, *Re: Gnutella exposure/evaluation: Request for comments, brickbats, etc*, Netsys.com Firewall Mailing List Archives, May 25, 2000
- http://www.netsys.com/firewalls/firewalls-2000-05/msg00528.html

[14] Gnutelliums home page
- http://www.gnutelliums.com

[15] stavrosthewonderchicken, MetaFilter Community 'blog, January 19, 2002
- http://www.metafilter.com/mefi/14009

[16] mr_crash_davis, MetaFilter Community 'blog, January 19, 2002
- http://www.metafilter.com/mefi/14009

[17] Eytan Adar and Bernardo Huberman, *Free Riding on Gnutella*, First Monday, volume 5, number 10, October 2000
- http://firstmonday.org/issues/issue5_10/adar/index.html

[18] Gibson Research Corporation, *Internet Security Issues for Napster and Gnutella Users*, January 31, 2002
- http://grc.com/su/peertopeer.htm

[19] *Knowbuddy's Gnutella FAQ*
- http://www.rixsoft.com/Knowbuddy/gnutellafaq.html

[20] Seth McGann, *Gnutella could cause allergic reaction*, May 11, 2000
- http://interrorem.com/news/newsbody.php3?parentid=0&newsid=39

[21] McAfee Security, *VBS/GWV.a*, May 30, 2000
- http://vil.nai.com/vil/content/v_98666.htm

[22] Global Incident Analysis Centre, *Detects Analyzed 8/18/00*, August 18, 2000
- http://www.sans.org/y2k/081800.htm

[23] McAfee Security, *W32/Gnuman.worm*, February 28, 2001
- http://vil.nai.com/vil/content/v_99024.htm

[24] IDGNet, *Worms worms worms and important Outlook e-mail updates*, March 2, 2001
- http://www.idg.net.nz/webhome.nsf/ArchiveVirus/ <link split for clarity>
084A6748666E7A82CC256A02007055C1!OpenDocument

[25] Robert Lemos, *Gnutella worm finds new way to squirm into PCs*, CNet News.com, February 26, 2001
- http://news.com.com/2009-1001-253185.html?legacy=cnet&tag=mn_hd

[26] F-Secure Virus Descriptions, *DlDer*, January 7, 2002
- http://www.europe.f-secure.com/v-descs/dlder.shtml

[27] ByteRage, *Vulnerability Development mailing list web archive Re: Clicktilluwin DLDER Trojan*, January 3, 2002
- http://lists.insecure.org/vuln-dev/2002/Jan/0023.html

[28] SecurityFocus, *Gnut Gnutella Client Arbitrary Script Code Execution Vulnerability*, August 30, 2001
- http://online.securityfocus.com/bid/3267

[29] Common Vulnerabilities and Exposures, *CAN-2001-1004 (under review)*, Jan 31, 2002
- http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN

[30] R. Fielding et al., *RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1*, June 1999
- http://www.ietf.org/rfc/rfc2616.txt

[31] Gnutelliums, *Gnut - console Gnutella client for Linux and Windows*
- http://www.gnutelliums.com/linux_unix/gnut/