Global Information Assurance Certification Paper

GCIH Practical

By: Brent Deterding

For Online Beta

NAPSTER ISSUES

(and the alternatives)

This practical is a hybrid between option #1 (Illustrate an Incident) and option #2 (Document and exploit, vulnerability, or malicious program). SANS/GIAC permitted this hybrid (Eric Cole). **The focus is on the evaluation of Napster (option #2 part).**

Executive Summary

This incident occurred at the University of Missouri at Columbia. A single student in a residence hall was using Napster to an excessive degree, which led to a partial denial-of-service for other students. The use was discovered and terminated. However, problems continued to arise with regards to Napster and a more permanent solution had to be devised.

Peer-to-Peer File Sharing

Peer-to-Peer (P2P) is not an unknown concept. In fact, it has been around for quite some time. HTTP (web browsing) is P2P, as is FTP. In the absence of restrictions, such as firewalls, anyone could be a client or a server. When used in a manner such as Napster, P2Ppresents some difficulties. The fact that everyone can be a server and the fact that everyone wants what someone else has (or vice-versa), coupled with the fact that the traffic is not easily identifiable and cannot be shaped easily make P2P a problem when it comes to Napster and Napster-like clones.

Napster

Napster is a P2P client/server that is widely known and widely used that is used to share music files in the MP3 format. MP3 allows files of relatively small size with CD-quality audio. Napster specifically presents its own unique problems. Napster is almost always used as a conduit for copyrighted material, it consumes huge amounts of bandwidth, it has its own protocol, and it has many legal issues surrounding it.

University of Missouri Background

The University of Missouri is not unlike many other institutions of higher learning in their policies and practices. Although serving anything out falls under the guidelines of the acceptable use policy (AUP), the enforcement of that guideline was tread upon lightly. The AUP covered some of the more basic principles. Illegal activity was not allowed, nor was money-making activities, nor was interfering with others through. At that time, no perimeter protection whatsoever was in place. Given the nature of college students; this was probably a good idea. Students can be very vocal and can organize quickly with arguments relating to first-amendment rights. With Napster not clearly being defined as illegal administration and legal was not willing to block it on principle. This specific incident would raise a new issue, however; availability and distribution of resources.

The University of Missouri's network had no perimeter protection of any kind in the notion of an open environment. Students is residence halls have 10/100 ports in their rooms and a newly-designed gigabit network to use. The Internet link was 45 Mbit. The general path from a residence hall room to the Internet was as follows:

<switch->1150->1200->1200->I here>

As you can see, students had it nice. Unrestricted, fast Internet access to do with as they so chose and mom and dad got the bill. Metrics for the core network and down to the building level on incoming and outgoing traffic is kept using MRTG. Using this, unusual spikes can be seen and addressed quickly, and the overall level of traffic can be monitored.

Beginning in August 1999 upon resumption of classes for the fall semester, we began to see an increase in the amount of traffic. Associating this to the normal tendency for use to grow with time, as well as to several thousand new users, we dismissed it as normal. As use continued to escalate we became more concerned. Being a student myself I was well-aware of the popularity of Napster and its associated bandwidth consumption. Placing a counting filter on the Internet router to count the number of packets going to of coming from the Napster Inc. block of IP addresses revealed an astounding amount of traffic.

The Incident

The incident occurred on Saturday afternoon during January, 2000. The Help Desk was called with a complaint of slowness in a residence hall network. This particular residence hall always was and always had been the leading producer and consumer of bandwidth at the University, so high traffics volume was not abnormal. The Help Desk looked at the MRTG graphs and saw an abnormally high amount of traffic. From this on residence hall, 35 Mbit was originating - 78% of the total Internet link of the University. The Core Networking on-call person was called. He called me, as I had lived there a year previous and had gotten my start in networking through trouble with bandwidth and MP3s.

After meeting at the Network Management Center and seeing the traffic graphs, we headed over to the residence hall with a sniffer. Immediately everyone knew we were there, although for what they did not know. A few probing questions were asked, although we did not answer. When we saw the volume of traffic we too were stunned; 35 Mbit is a lot of packets! We grabbed a few seconds worth and began to analyze them. They were mostly packets with PSH and ACK set containing 1448 bytes of data; the rest were ACKs to those packets. We noticed this traffic as file transfer due to the size and pattern of the PSH/ACKs and ACKs. We filtered out large packets and sniffed once again. It did not take long to notice large amounts of traffic to and from IPs we recognized as Napster IPs.

Knowing that politically we would be committing suicide if we blocked those IPs entirely, we were at a loss as to what to do. We couldn't block it entirely, and we knew that if we tried it would be circumvented shortly. We couldn't turn off the entire building, as that would not be acceptable. However, we could stop the worst offenders for utilizing too much bandwidth and using a server. We monitored for 15 minutes using iptraf (<insert link here>) and noticed that one person was sending out nearly 17 Mbit. That is over 2 MB *per second* and was completely unacceptable. The rest of the traffic was spread out fairly evenly though. In the meantime we had contacted our boss, who was now on the phone with us about the time we found the single user. He instructed us to take whatever measures we could to block Napster in that building and to unplug the user from the network. He told us that he would take any heat resulting from blocking Napster at that building, but that given the circumstances he wasn't worried. He knew the methods of circumventing Napster blocking and also was aware of the political ramifications. We unplugged the high-usage user and blocked the default data ports of 6699 and 6688 and went home after seeing traffic nearly stop.

Monday morning we looked at the maps in the weekly staff meeting and showed the drastic drop-off and what we did to cause it. As if to mock us, the graph, which was updated every 5 minutes, began to steadily rise throughout the meeting, until it had resumed nearly 20 Mbit of traffic flow. I came up with the idea to block the Napster IPs, as well as the default server port of 8888. We (the on-call person and I) went over to the residence hall again and began to sniff again. Everyone had simply changed the default port to random ports. We blocked the Napster IPs and waited. Traffic dropped off as expected. Within 20-30 minutes it had begun to rise once again. The traffic now was to random IPs on random ports but was most definetely Napster traffic. We were at a loss as to the course of action to be taken.

We removed the filters so as to avoid questions and researched the topic. Traffic volume decreased with time and as long as it was below 30 Mbit normal traffic flowed well enough. By and large it was forgotten about after the two attempts at blocking it failed.

Preparation

What could the University have done to prepare for students and their bandwidth consumption? Several options come to mind, including perimeter protection, Intrusion Detection capability, buying more bandwidth, or web caching. However, these do not deal with bandwidth consumption specifically by Napster. Napster file transfers can function if one end of the cummunication is behind a firewall, although not if both ends are firewalled. An Intrusion Detection System (IDS) could not, at that point, detect Napster traffic accurately. More bandwidth was not an option for multiple reasons, and would have done nothing for the underlying problem anyway. Web

caching, while helping conserving the bandwidth leaving and coming into the University, would hvave done nothing for Napster traffic. So what was the University to do? Port blocks and IP blocks were circumvented, and no one had any other options. It appeared that the University was stuck.

Identification

The initial indication came from a call to the help desk regarding slowness in a residence hall. We would have noticed on Monday, as we kept an eye on that particular residence hall. However, this occurred on a Saturday afternoon. As was mentioned above; the network was sniffed to determine what was consuming the bandwidth. But how did we know that it was Napster traffic from the trace? First, let's look at tcpdump notation:

tcpdump format:
[timestamp] [interface] [src ip] > [dst ip]: [protocol*] (frag: []) [message*] [tos*]
[hex data]

* denotes that the field may or may not be present
** more fragments bit - may or may not be present

Now; what kind of packets were we seeing? Packets strikingly similar to these were showing up in mass quantities and very fast . . .

```
13:36:56.153619 < h0010b58ceb5d.ne.mediaone.net.6700 > 192.168.1.101.3256: . 0:0(0) ack
95569 win 65280 <nop,nop,timestamp 684514 109183785> (DF)

13:36:56.153690 < 192.168.1.101.3256 > h0010b58ceb5d.ne.mediaone.net.6700: P
110049:111497(1448) ack 0 win 32120 <nop,nop,timestamp 109183913 684514> (DF)

13:36:56.153715 < 192.168.1.101.3256 > h0010b58ceb5d.ne.mediaone.net.6700: P
111497:112945(1448) ack 0 win 32120 <nop,nop,timestamp 109183913 684514> (DF)
```

What does this tell us? The timing is very quick; indicating automated (non-interactive) traffic. An emphemeral port sending data to another emphemeral port indicates data transfer as well. An internal address is sending data out to another machine with the PSH and ACK flags set. PSH and ACK together can typically indicate the transfer of data. The maximum allowed MTU is being sent as well, further pointing to file transfer. Also corroborating evidence is a 32K window size.

When we see packets like this it indicates that our internal machine is transfering data to an external machine via some form of file transfer. It could be ftp, it could be http. We saw this pattern going to multiple external machines. So how to nail it as Napster? We looked for port 8888 traffic with Napster IP addresses as src and dst addresses, of course! We also looked for the default data ports of 6688 and 6699 and found it in abundance. We had positively identified the traffic as excessive Napster traffic. At the time, the only thing that we could have done to identify the traffic as Napster-like would have been to catch the first packet in a transfer and look at the ASCII; it would have contained the title of the song being transferred.

Containment

So how was the problem contained? The long-term solution will be discussed later, but the short term solution was not really a solution. As mentioned, the solution was to first block the default data ports, then the Napster IPs, then the default Napster server port. Each of these failed in short order. The most flagrant offender was denied access. However, due to the widespread nature of Napster we were unable to isolate very many individuals as the biggest offenders. The question as to why our measures didn't work arose. Were the students just that good? After all, we had never had this problem with ftp sites; shut them down and the problem went away. Napster's default data port of 6688 or 6699 is very easy to circumvent; it simply requires changing a field in the options. Blocking the server port of 8888 and Napster IPs was circumvented in the same way; proxies. There are many free, public proxies available (http://proxys4all.cgi.net) that were used to effectively bypass our blocks. One user was contained through unplugging his network; everyone else was free to use Napster with a

small amount of client reconfiguration. Again, a better method was not available at the time and the University found itself stuck.

Eradication
Napster did not go away until much later and still plagues many Universities and businesses.

Recovery
Because we were unable to contain or eradicate napster, recovery was difficult. The immediate problem was averted by unplugging the most flagrant violator; which reovered some of the bandwidth claimed by Napster. The network continued to be under heavy load but survived the ensuing traffic without major harm or delay. Traffic gradually increased as usage became more widespread to the point where the University of Missouri had a steady 30 Mbit stream outbound from Napster traffic. A more permanent solution was forthcoming.

Follow-Up/Lessons Learned
So what was the University to do? What was any University or business to do with the issue of Napster? It seemed that it couldn't effectivcely be blocked! In the face of an impossibility a possibility arises many times . . .

Exploit Details:

Name: Napster Protocol (Napster)
Variants:
amster -- amiga napster client
BeNapster -- BeOS napster client
BitchX -- IRC chat client with napster plugin
TekNap -- (formerly called BWap) standalone console unix client based on bx-nap plugin for BitchX
crapster -- BeOS napster client
gnap -- gnome napster client
gnapster -- gnome napster client
gnome-napster -- gnome napster client
gtk napster -- gtk napster client
hackster -- visual basic napster client
iNapster -- WWW interface to napster
java napster -- java napster client
jnap -- java napster client
jnapster -- java napster client
Knapster -- KDE napster client
Lopster -- gtk/unix napster client
MacStar -- Mac Napster Client
MyNapster Webclient
nap -- linux/bsd command line client
NapAmp -- Napster plugin for WinAmp
Napster for BeOS
Napster for MacOSX
TkNap -- Tcl/Tk napster client
Riscster -- napster client for RiscOS
snap -- perl napster client
webnap -- PHP napster client
XmNap -- motif Napster client

Operating System: Nearly any operating system is able to use a Napster client in some way
Brief Description: Napster is a file-sharing protocol based on Peer-to-Peer file sharing, ala HTTP and FTP. It has gained much notoriety for the widespread usage of the Napster Inc. client and the legal troubles that surround it.

Protocol Description:

A word about Napster Inc. versus the Napster protocol. The Napster protocol would more aptly be termed the OpenNap protocol. There exists a company Napster Inc. who first released a Peer-to-Peer (P2P) client called Napster. The protocol it used was later termed OpenNap and is freely available. References to the Napster protocol, or to Napster, within this document indicate the OpenNap protocol. When Napster Inc. is the intention it will be denoted as such. The best protocol description can be found here: http://opennap.sourcceforge.net/napster.txt

In general, the Napster protocol is fairly simple. It operates over TCP through a series of messages. Typically Napster servers use port 6666, 7777, 8888, and 9999 to communicate. The format of the packets is <length><type><data>. Length is 2 bytes long and specifies the length of the data portion of the packet. Type is also two bytes long and specifies the message type. Data is the data accompanying a given message type, such as the login name of a login message type. Although there are mnay message types, we will focus on some of the more important ones here. Some of the more popular message types and their corresponding data options are:

2 login [CLIENT]

Format: <nick> <password> <port> "<client-info>" <link-type> [<num>]

<port> is the port the client is listening on for data transfer. If this value is 0, it means that the client is behind a firewall and can only push files outward. it is expected that requests for downloads be made using the 500 message (see below)

<client-info> is a string containing the client version info

<link-type> is an integer indicating the client's bandwidth

0 unknown
1 14.4 kbps
2 28.8 kpbs
3 33.6 kbps
4 56.7 kbps
5 64K ISDN
6 128K ISDN
7 Cable
8 DSL
9 T1
10 T3 or greater

<num> build number for the windows client [optional]

Example: foo badpass 6699 "nap v0.8" 3

6 new user login [CLIENT]

Format: <nick> <pass> <port> "<client-info>" <speed> <email-address>

This message is used when logging in for the first time to register a new nickname. The client normally checks for an unused nick using the "check nickname" (7) message and upon receipt of a "nickname not registered" (8) from the server will proceceed to log in with this command.

note: this message is similar to the 0x02 message, with the addition of <email-address> on the end

Example: foo foo 6699 "nap v0.8" 3 email@here.com

7 nick check [CLIENT]

Format: <nick>

Queries the server to see if <nick> is already registered. This message is typically used prior to logging in for the first time to
check for a valid nickname.

Response to this message is one of 8, 9 or 10

203 (0xcb) download request [CLIENT]

Format: <nick> "<filename>"

client requests to download <filename> from <nick>. client expects to make an outgoing connection to <nick> on their specified data port.

Example: mred "C:\Program Files\Napster\generic cowboy song.mp3"
SEE ALSO: 500 alternate download request

500 (0x1f4) alternate download request [CLIENT]

<nick> "<filename>"

requests that <nick> make an outgoing connection to the requesters client and send <filename>. this message is for use when the person sharing the file can only make an outgoing tcp connection because of firewalls blocking incoming messages. this message should be used to request files from users who have specified their data port as 0 in their login message

607 (0x25f) upload request [SERVER]

<nick> "<filename>" <speed>

this message is used to notify the client that user <nick> has requested upload of <filename>

Example: lefty "generic band - generic song.mp3" 7

Description of Variants

Programs using the Napster protocol were listed previously. They all indicate that they follow the Napster protocol. Differences may include how they send message length, type, and data. For example, the Napster Inc. Windows client sends the length, type, and data in one single packet whereas Gnapster, a common Linux client, sends the length and type in one packet and the data in a second packet. The protocol is still followed, however. Clients exist for the Amiga, Macintosh, RISC processors, BeOS, UNIX, Linux, Java, and almost any other conceivable platform can, in some manner, access and use a Napster client.

How the Exploit Works

We will use the simple topology of two clients (#1 and #2) and a server in a typical (simplified) session.

Client #1 is an existing user and logs in to the server (msg 2).
Client #2 is a new user and logs in to the server (msg 7).
The discepancy here will be discussed later.

Client #1 transmits their shared-file list for each song (msg 100).
Client #2 transmits their shared-file list for each song (msg 100).

Client #1 searches for a song (msg 200) that client #2 has.
Client #1 receives a search response (msg 201) from the server indicating that client #2 has the song.
The response is finished when a search response ending is sent (msg 202).

Client #1 sends a download request (msg 203) to the server.
Server sends an upload request (msg 607) to client #2.
Client #2 sends an upload request accepted (msg 608) to the server.
Server sends a download ack to client #1 (msg 204).

Client #1 sends a downloading file status (msg 218) to the server.
Client #2 sends a uploading file status (msg 220) to the server.

<Data transfer initiation and completion>

Client #1 sends a downloading complete status (msg 219) to the server.
Client #2 sends a uploading complete status (msg 221) to the server.

For the purposes of identifying a blocking Napster much of the above is not necessary. To that end, emphasis is not placed on message types not detailed above in the protocol description. A word about the most used function of Napster; file transfer.

File transfer in Napster is peer-to-peer, meaning that it occurs directly between clients using tcp. Although any port can be specified, the default ports are usually 6688 or 6699. There are four modes of file transfer; normal upload, normal download, firewalled upload, and firewalled download. Firewalls can be configured in a myriad of manners and means. However, it is safe to say that most firewalls do not allow indiscriminate (i.e. - ephemeral) connections to internal hosts from external hosts. Therefore, if a client is firewalled and doesn't have the ability to accept incoming connections, standard downloading is not possible. To solve this problem, a push request is sent through the existing client->server connection. If both clients are behind a firewall file transfer is not possible. A normal transfer works by the client wanting a file initiating the connection to the client holding the file. A firewalled download operates in the reverse fashion; the client holding the file initiates a connection to the client wanting the file.

A download always begins by one client searching or browsing (msgs 200 or 211). A client wishing to download a file issues a get (msg 203) to the server, which responds with a get ack (msg 204). A get ack contains the data port of the client who holds the file. If the get ack indicates a data port of 0 for the holding client's file, the requesting client issues an alternate download request (msg 500). This requests that the holding client initiate a connection to the client requesting the file. If, however, the get ack contains a nonzero data port, the requesting client connects to the given port of the holding client. First, a 1 (ASCII 49) is returned. The requesting client then sends a get message (msg 203) to the holding client. After this is complete, for either firewalled or non-firewalled downloads an error message may be returned or the data transfer may be initiated. Server-side statistics are kept on the number of concurrent uploads and download using messages 218 - 221.

The following were all captured using tcpdump.

A Napster logon looks like this on the network:

```
11/10-00:38:19.255526 128.206.250.209:2365 -> 64.124.41.236:8888
TCP TTL:64 TOS:0x0 ID:36727  DF
*****PA* Seq: 0xB2ABBD80   Ack: 0x7C04D1FA   Win: 0x7D78
TCP Options => NOP NOP TS: 9668023 7890647
2B 00 02 00                                        +...
```

A Napster download request looks like this on the network:

```
11/10-00:38:34.529297 128.206.250.209:2365 -> 64.124.41.236:8888
TCP TTL:64 TOS:0x0 ID:36746  DF
*****PA* Seq: 0xB2ABE1EB   Ack: 0x7C04D78E   Win: 0x7D78
TCP Options => NOP NOP TS: 9669550 7890819
46 00 CB 00                                        F...
```

Notice the boldfaced payload bytes - these are the message types corresponding to login and download request. These are both from gnapster, as only the length and field are in this packet. A Windows clients packet would be much larger as other information would follow this payload.

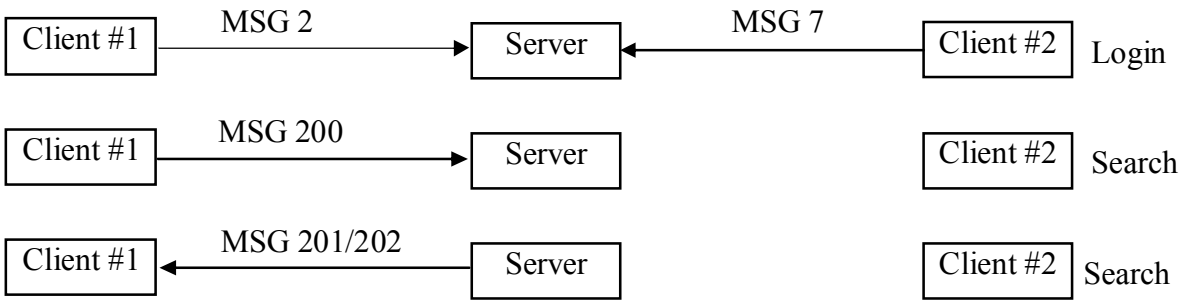
Here is an upload request from a Windows client:

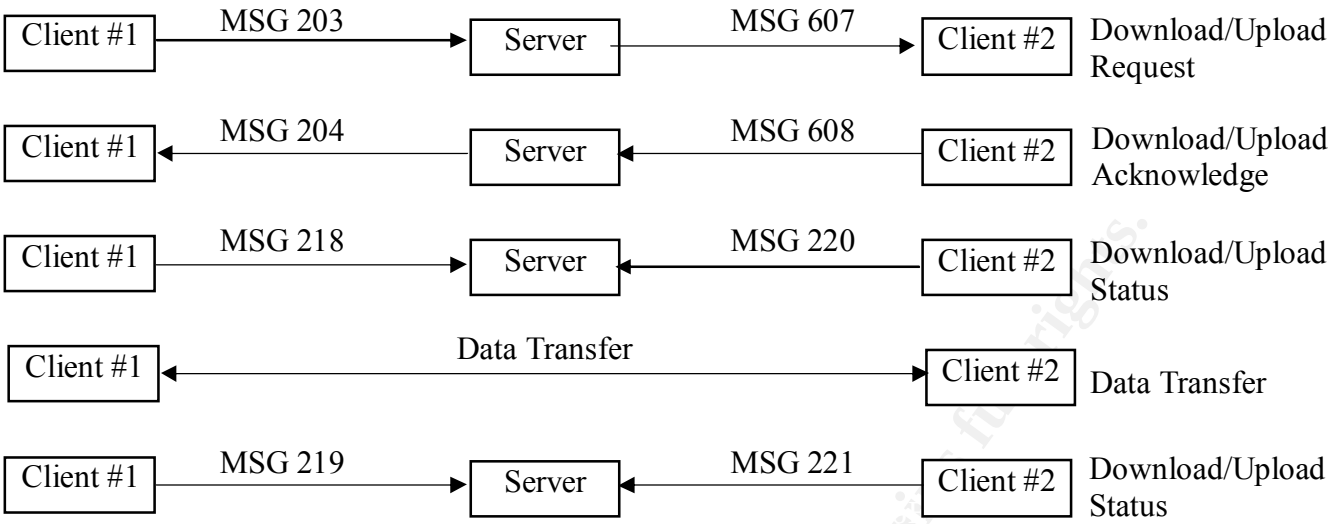
```
11/10-00:55:26.398731 64.124.41.236:8888 -> 128.206.250.209:2365
TCP TTL:48 TOS:0x0 ID:51735  DF
*****PA* Seq: 0x7C04D975   Ack: 0xB2ABE287   Win: 0x7C70
TCP Options => NOP NOP TS: 7993346 9770262
3C 00 5F 02 62 6F 64 69 65 33 20 22 5C 68 6F 6D  <._.bodie3 "\hom
65 32 5C 6E 65 77 6D 70 33 5C 43 4F 4D 45 44 59  e2\newmp3\COMEDY
5C 52 6F 62 69 6E 20 57 69 6C 6C 69 61 6D 73 20  \Robin Williams
2D 20 41 6C 63 6F 68 6F 6C 2E 6D 70 33 22 20 38  - Alcohol.mp3" 8
```

Following is a Napster nick check. I have found several inconsistencies with the definition of the protocol - this being one of them. The specification states that the following happens for a new user: nick check (type 7), nick ack (type 8), new user login. Instead, the new user login data is truncated onto a nick check, which isn't supposed to happen according to the specification. Here is the nick check under gnapster:

```
11/10-09:57:16.428100 128.206.250.209:2611 -> 207.195.111.2:8888
TCP TTL:64 TOS:0x0 ID:56387  DF
*****PA* Seq: 0xF0ECA3AF   Ack: 0xCADF78BA   Win: 0x7D78
TCP Options => NOP NOP TS: 13021740 3549238
08 00 07 00                                        ....
```

Diagram





How to use the exploit

Whichever client is used, installation is typically fairly painless and typical. The Windows client uses an installation wizard to install the client and make registry changes as appropriate. Once installed, use is very straightforward, as everything is handled through the gui.

Signature of the attack

I am presenting on the topic of both identifying and blocking Napster at SANS 2001 in Baltimore in May, 2001. I'm including my presentation. Any questions not covered in this practical are covered in the presentation and I do not want to duplicate the entire contants of the presentation.

The Napster protocol can be identified on the network, just like FTP or HTTP. Something within the protocol must be identifyable and unique, however. The message type provides this information. By using an Intrusion Detection System such as Snort (http://www.snort.org) one can now identify Napster traffic with rules that I wrote.

The "old method" for identifying Napster traffic was to match on port numbers or IP ranges. As discussed previously, port numbers can be changed or proxied and OpenNap provides hundreds more Napster servers that are not Napster Inc. owned.

The "new method" I came up with was to key in on the message type in the packets. When I came up with the idea I was not aware of anyone who had done this. Snort had rules to identify Napster traffic, but they were based on port numbers, or the default Windows install. They were:

```
alert TCP any any <> any 6699 (msg:"Napster Client Data"; flags: PA; content: ".mp3"; nocase; )
alert TCP any any <> any 8888 (msg:"Napster 8888 Data"; flags: PA; content: ".mp3"; nocase; )
alert TCP any any <> any 7777 (msg:"Napster 7777 Data"; flags: PA; content: ".mp3"; nocase; )
alert TCP any any <> any 6666 (msg:"Napster 6666 Data"; flags: PA; content: ".mp3"; nocase; )
alert TCP any any <> any 5555 (msg:"Napster 5555 Data"; flags: PA; content: ".mp3"; nocase; )
alert TCP any any <> any 4444 (msg:"Napster 4444 Data"; flags: PA; content: ".mp3"; nocase; )
alert TCP any any <> any 8875 (msg:"Napster Server Login"; flags: PA; content: "anon@napster.com"; )
```

I came up with the following rules, which are in the most current snort ruleset:

```
alert tcp $HOME_NET !80 -> $EXTERNAL_NET 8888 (msg:"INFO napster login"; flags: A+; content:"|00
0200|"; offset: 1; depth: 3; )
```

alert tcp $HOME_NET !80 -> $EXTERNAL_NET 8888 (msg:"INFO napster new user login"; flags: A+; content: "|00 0600|"; offset: 1; depth: 3;)

alert tcp $EXTERNAL_NET any -> $HOME_NET 8888 (msg:"INFO napster download attempt"; flags: A+; content: "|00 cb00|"; offset: 1; depth: 3;)

alert tcp $EXTERNAL_NET 8888 -> $HOME_NET any (msg:"INFO napster upload request"; flags: A+; content: "|00 5f02|"; offset: 1; depth: 3;)

alert tcp $HOME_NET any <> $EXTERNAL_NET 8875 (msg:"INFO Napster Server Login"; flags: A+; content:"anon@napster.com";)

As you can see, I look at a specific offset for a specific number of bytes for a specific pattern to identify the packet as Napster. My rules demonstrate that Napster, no matter the platform, the version, the port, or the proxy, can be detected successfully. There are some false positives associated with these rules. Some NetBIOS traffic and some web traffic (I cannot duplicate this) are incorrectly identified as Napster traffic. I reccommendeliminating any reference to port numbers to decrease the potential of bypassing these rules.

How to Protect Against It

Snort has "flexible response" built into it, meaning that it can "session-snipe" at connections to kill them. By appending "resp: rst_all" to these rules, snort will kill any and all Napster traffic that it sees. In testing on a small (25 node) network, this has proved 100% effective.

Another option I recently discovered was a product by the name of packethound. This product is in use at the University of Missouri and blocks Napster uploads effectively on a very busy network. It works for hundreds of other protocols as well. It can be found at (http://www.packethound.com).

Source Code/Pseudo code

Napster client source code can be found from each client's web page indicated previously.
Napster server source code can be found at http://download.sourceforge.net/opennap/opennap-0.40.tar.gz

Additional Information

I encourage you to look my presentation on the subject matter over. It is enclosed. Thank You!