Attacking P2P Networks

ECS 235 — Hao Chen - Fall 2005

Andre L. Nash

December 16, 2005

1 Introduction

Peer-to-peer networks enable sharing of files amongst users in such a way that the computing power and network bandwidth required to share these files are shifted from relatively few servers to the users which request them. While peer-to-peer networks traditionally have been linked to illegal activity - unauthorized sharing of copyrighted music, movies and applications - the fact that these networks shift huge bandwidth and computing power requirements onto the users make them an attractive option for businesses, musicians, and artists that desire to promote their products via legal means.

Peer-to-peer file sharing networks are constantly under attack. The following list is not comprehensive, but provides a good sample of the types of attacks these networks face:

- *Poisoning*, where a client can provide content that does not match the description. For example, a client, A, can broadcast a message saying it needs file X. A malicious client can send a message back to A saying it has file X, then send it file Y.
- *Denial of service* attacks that decrease or cease total capable network activity. As we will see, denial of service attacks can be byproducts of other attacks.
- *Defection* attacks, which allow a client to participate on the network with a very low upload-to-download ratio. Defection attacks can be subtle - one can pretend to have a very high upload-to-download ratio by *credit shaping*.
- *Virus* attacks, where a malicious client can add viruses into files shared on the network. This means that our definition of a malicious client is one that may be unaware of their activities.
- *Malware* attacks, where the peer-to-peer software contains spyware, or propagates files which contain spyware.
- *Filtering* attacks, were "network operators may attempt to prevent peer-to-peer network data from being carried" [1].
- *Identity* attacks, where client anonymity is not protected.
- *Spamming* attacks where unwanted information is sent across the network.

This paper attempts to examine and provide practical solutions to most of these attacks[1].

Options available for securing the following peer-to-peer networks, subject to the given attacks, will be examined throughout:

- eMule (derived from and backwards compatible with eDonkey 2000)
- BitTorrent

[1]Specifically, there is a lack of good information on malware, filtering, and spamming attacks for the peer-to-peer networks examined in this paper. Malware appears in several clients (even official clients) across a variety of peer-to-peer networks.

2 Poisoning and Spoofing Attacks

2.1 Piece/Chunk Identification

2.1.1 eMule

Each file is split into chunks of at most 9.28 MB, each chunk of which is split into 180 KB blocks. Each chunk is hashed using the MD4 algorithm, then a *file hash* is computed by string concatenating each chunk hashes, then taking the MD4 hash of that. After a chunk has been downloaded by a peer (successfully or not), it computes the *part hash* and compares it to the part hash sent by the peer. In the event of a part hash mismatch, recovery is done on a block-by-block basis. Note that several MD4 collision generators exist, and the task of generating a colliding chunk (a chunk that has the same MD4 hash as the original chunk) is trivial. Such colliding chunks can poison files in eMule networks without detection.

A secondary, *root hash* is computed for each part by using the SHA1 algorithm, again with 180 KB blocks.

2.1.2 BitTorrent

Unlike eMule, the size of each piece (equivalent to a chunk in eMule terminology) is determined by at torrent creation time (using the make-torrent application). The sizes allowed for pieces are: 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, 1.0 MB, and 2.0 MB. *The reason for variable sized pieces is unstated by the author.*

Each file piece is hashed using the SHA1 algorithm. These piece hashes are stored in a torrent file, which is then placed on a web site. The entire file that is available on the network is successfully received by a client if the piecewise SHA1 hashing of the file matches what is stored in the torrent file.

2.1.3 Optimal Piece/Chunk Size Criteria

To prevent poisoning attacks, we prefer to have chunk sizes as small as possible. In the case of fixed chunk sizes, or chosen large variable chunk sizes (over 25 KB), we prefer to use some type of incremental signing or hashing technique to continuously verify that the pieces/chunks are on the path to correctness. Note that in the case of eDonkey 2000, the peers are all assumed to be extremely trusted, since each client receives part hashes from other clients on the network. This varies from the BitTorrent approach, where each client assumes that the record of reference (the torrent file itself) is trusted. This means that if you can obtain the torrent from a trusted source, you can remove (to some extent) the peers from the trusted computing base (TCB).

2.2 Communication Protocols

2.2.1 eMule

eMule is a hybrid peer-to-peer network. In this hybrid network, a client connects to a eMule server. The server creates a ”Client ID” and sends it back to the client. There is nothing in this packet which prevents tampering - such as a signature. One easy way to prevent impersonation is to assume the servers are trusted, and have a server sign the client ID (perhaps via group signature). This will allow all trusted servers to verify that the signature is correct, and prevent client ID forging attacks.

eMule supports ”secure” user identification using the following protocol. Optional packets represent backward compatibility with eDonkey 2000 (from which eMule is derived). Secure user identification is an eMule extension, and is optional.

1. Initial Handshake:
   (a) Open TCP Connection
   (b) Send **hello** packet
   (c) (Optional) Receive eMule info packet
   (d) Receive **hello** answer
   (e) (Optional) Send eMule info packet

2. Secure User Idenfitication:
    (a) Send **secure identification** packet:
        i. This packet will contain whether or not the public key of the target client needs to be sent. If so, then we first receive the RSA public key of the target.
    (b) Receive **signature** packet

There are some interesting security ramifications being explored with this type of "secure" user identification. In particular, it appears to be client-specific as to whether or not they will allow multiple connections per IP address. If so, suppose that client A has an open, secured connection with client B. If there is nothing that will prevent a malicious client, C, from opening a second, unsecured connection with B pretending to be an eDonkey 2000 client (this can be accomplished by not sending the optional "eMule info answer" packet to B). Nothing in the protocol prevents a malicious client from impersonating other clients in an unsecured manner and cancelling their requests (the cancel message doesn't include what connection-specific information that the client can use to determine what to cancel, so it must cancel all file transfers). Such an attack amounts to an identity/denial of service combination attack, and can be used to gain position in another client's upload queue. The eMule specification[2] specifies, however, that "the typical scenario in which this (cancel) message is sent is when a downloading's has reached the top of the queue but it has already downloaded the file from another source."

We are also investigating how we can break the secure user identification, so that we can impersonate someone else in order to gain credit advantage (where we steal impersonate someone else in order to use their accumulated credits to our advantage).

2.2.2 eMule Malware and Virus Attacks

Each client can request "part hashes" from other clients. In this case, it is possible for a malicious client, C, to impersonate a file transfer and sending fake chunk hashes. In this case, the client will not know that chunks are incorrect, unless they collaborate with other non-malicious clients to find the correct chunk hash.

Consider a malicious client that impersonates other clients and sends out fake chunk hashes. Assuming non-collaborative hash checking, such a client can embed malware and/or viruses into shared files[2].Note that affected files are **not** limited to only executables - several exploits have been performed on applications which play music (mpg123 and winamp, for example) and display jpeg files, for example [9, 10].

2.2.3 BitTorrent

BitTorrent was, until (September 2005) a hybrid peer-to-peer network. With the ability for torrents to be created in *trackerless* form (version 4.1.x), BitTorrent now has the ability to run as either a pure or hybrid mode. There are several protocols.

When a connection is opened with another peer, it is sent a handshake. The handshake consists of:

1. **handshake:** pstrlen pstr reserved infohash peerid

Of note, the peerid is chosen by the client, and it is allowed to be any value that the user wishes (as long as it is unique among all clients in a given tracker). BitTorrent assumes that all incoming connections are from valid peers. If Client B receives a handshake from Client A, Client B simply replies to Client A with a handshake of its own. Note that there is no receipt from client B that the handshake was received, and neither client are assured that their handshake was received correctly. This type of handshaking protocol could lead to man-in-the-middle attacks. Client A believes it is interacting with client B, when infact, it is interacting with client C (one could cross-reference the peerid with the tracker, but a client could send phony information to a tracker in an attempt to get a list of *assumed* valid peerid/ip address combinations).

The request, piece and cancel messages, initiated between clients and after a handshake has occured, have the following forms:

[2]Infact, peer-to-peer viruses have their own class, called P2P-Worms. Most such worms create fake files in shared directories of clients for networks such as Kazaa, Morpheus, Gnutella and eMule.

1. **request:** len=0013 id=6 index beginoffset length
2. **piece:** len=0009+X id=7 index beginoffset block
3. **cancel:** len=0013 id=8 index beginoffset length

This appears to lead to a potential denial-of-service attack, because malicious clients can request blocks on the behalf of other clients, as well as send other clients pieces that they neither need nor requested. Blocks that are sent can be anything. The official BitTorrent client makes no provisions for ensuring that a client doesn't receive a packet they didn't request, because peers talk directly to peers (even in the presence of the tracker)[3]. The later issue leads to a poisoning attack - nothing guarantees that the piece being sent matches what is supposed to be sent.

HBO used this fact to their advantage; they intentionally flooded BitTorrent networks with malicious clients that claimed to have valid pieces[6]. Since these piece hashes are stored in the torrent file, anyone can claim to have a valid file piece. Once a client downloads an entire piece, they hash what they have downloaded and compare it to the hash in the torrent file. If the hashes do not match, the client redownloads the piece in its entirety.

Finally, if the flag 'one connection per ip' is false, a client can spoof an IP address and create a connection between another client with a fake peerid (as mentioned above). Since the traffic is sent unencrypted, an adversary with the ability to sniff a client's packets can also send arbitrary *cancel* messages on behalf of that client.

As will be suggested later, given the presence of a *secure torrent file*, a secure BitTorrent client can enforce downloads in a specific order such that each piece can signed and verified on the fly, and either a) enforce a 'one connection per ip' requirement, or b) for a given IP address, for the first connection, establish a session key.

2.3 Defection Attacks

2.3.1 Client Interaction

For both networks, clients interact in swarms (clients that work with each other to receive files they desire). For example, if client A, needing a file F, is sharing a file G and client B has file F and needs file G, the two will work together to share files. In the case of eMule, there is considerable flexibility, as multiple files can be used to determine a peer cooperation level. With BitTorrent each file consists of multiple blocks, and peers cooperate to exchange file blocks as if they were multiple files. In other words, the two systems differ only in their view of files.

2.3.2 eMule

eMule is an open-source P2P program - as such, several modified clients[4] ("eMule mods") are available for a variety of operating systems. Many of these modified clients provide an enhanced GUI, and support peer banning, anti-identity stealing, increased download throughput, 'anti-leech' capabilities, unicode and intelligent chunk selection.

eMule attempts to deal with defection attacks by use of a credit system[3], where users are rewarded for uploading to the network. These credits are used by uploaders to implement a priority queue, as the more a user uploads, the higher he will be placed in the download queue for a given file. Credits are not global, however, they're maintained between peers (in particular, each peer locally saves the number of credits owed to another peer). As we will see, the credit system is flawed.

Modified *leecher* clients attack the network by defection. The main features of such clients are:

- Support stealing of other users identities that are downloading a particular file (of interest to the user). Once an identity has been spoofed, the client attempts to assume another user's position in a download queue.

[3]This claim is supported by a code inspection on the BitTorrent client, v4.1.6.
[4]eMule Plus, Sivka, Pawcio, Xtreme and MaTriK, to name a few.

- Uploading bogus data. Assume that filename F is a popular file on the network. The client, A, can create a file (say, a file consisting of a random bitstring), assign F as the filename, and report to an eMule server that they are sharing a file named F. Very likely, client A receives credit for an upload whenever a second client, B, downloads it, because B will not immediately verify that it is the correct file.
- Credit shaping. This is the practice of trying to join in a file sharing swarm by tailoring shared files in such a way that downloaders will accept you. This can be manually performed, but leecher clients automate this process. Since credits are maintained between peers, peers can pair up to exchange files of mutual interest in such a way that the upload-to-download ratio is high, but the number of files being shared is low.

2.3.3 BitTorrent and "Choking"

Choking is the "temporary refusal to upload" to a particular peer/client. [8] states that choking is used for two reasons:

- TCP congestion control behaves poorly when uploading over many connections simultaneously
- Choking allows each peer to get a consistent, optimal download rate

BitTorrent peers always unchoke a fixed number of peers (the specification uses four, but this is variable for the client) and use choking to ensure that they don't receive file pieces from clients that give them poor download speeds[5]. Note that choking does not imply that a client must stop downloading; in a non-malicious client, it allows simply allows peers to maximize their own download rates. When a client acts as a seed (where it has the complete file and only needs to upload), the client then tries to maximize its *upload* speed when deciding which peers to unchoke.

Since clients operate in a tit-for-tat mode, the theory is that all downloading peers (other than seeds) are required to upload most of the time, otherwise they'll be choked (because they will not provide either a consistent or optimal download rate for any client).

Combined with peer spoofing, one could complete a denial-of-service/defection attack on a BitTorrent simply by sending unsolicited 'choke' messages to clients (in order to maximize their download rates). While peers are optimistically unchoked[5] every 30 seconds, broadcasting this message frequently enough could effectively deny service to other peers, while providing a maximal download rate for the malicious client.

Hales and Patarin provide a variant of this attack, where a malicious client fakes the identity of several clients in an attempt to gain download advantage[12]. They also note two key items: 1) the lack of strict online identity checking (based on IP address and port) enables this attack, and 2) that such attacks are rare simply because of a type of *tribal dynamics* in torrent swarms. The dynamics they discuss are as follows: because trackers are not linked, users must **actively** and **manually** search for trackers which give them good performance. When freeloaders that engage in defection join a tracker/swarm, performance decreases; As a result, non-malicious peers tend to migrate towards better performing trackers/swarms. While this is true, a defection attack might, in the short term of a swarm's life cycle, be effective.

3 Identity Attacks and Anonymous P2P

As previously stated, an identity attack on a peer-to-peer network removes client anonymity.

For the BitTorrent network, clients participating in a swarm are public knowledge; One can simply query a tracker for a list of all clients that are using that tracker for the purpose of sharing a given file. eMule networks, by contrast, works via *callback requests*. A client, A, identifies peers by first initiating a search request to an eMule server. The search request results in A receiving a list of clients (a set of {client ID, client port} pairs). A chooses which of these clients he wishes to engage in sharing with by sending the eMule server a *callback request* from {client ID, client port}. Once the target client receives this message, the client

[5]Optimistic unchoking means that a peer is unchoked regardless of the current download rate from that peer. The peer chosen for the optimistic unchoke is rotated every third rechoke cycle. This helps increase a peers individual download rate, by actively searching for peers that can give them better download rates.

has the option of acknowledging the callback request. Generally a client responds to all such messages since it will only benefit him (giving him additional resources to get data from). Clearly, neither network does enough to preserve client anonymity against identity attacks.

One way to help guard against such attacks, at the expense of reduced throughput, is by introducing a network layer which, with high probability, protects the clients. Such a layer, combined with a peer-to-peer network, is called *anonymous P2P*[7]. These networks operate on the principle of *security by obscurity.* Each client acts as a universal sender or receiver in the sense that any client can request data on behalf of any other client on the network. This obfuscates the flow of data, and provides a high level of anonymity.

The three main anonymous P2P network layers are Tor (onion routing), Freenet, and I2P [6]. At the present time, Tor and I2P are currently being used as layers for anonymous P2P in BitTorrent, and in theory any of these networks can be used on top of any of these layers.

Further work would include an analysis of anonymous networking layers used for peer-to-peer networks (I2P-BT, for example).

4 Future Work

The version of BitTorrent examined was v4.1.6 (beta) of the official client, available at http://www.bittorrent.com. It is written in the Python programming language, and relies on 11 shared libraries (for a windows build): GTK 2.4 runtime, pygtk 2.4, pycrypto 2.0, python-win32gui, twisted 2.0.1, mfc71.dll, pysystray, GNUWin32 Gettext, py2exe, NSIS (Nullsoft Scriptable Install System)[7]. Additionally, NSIS requires three plugins - InstallOptionsEx, Processes, and KillProcDLL. For a *nix build, BitTorrent is ran directly using Python (no executable is made).

The application source code, comprised of 57 files totaling approximately 500 KB in size, is poorly commented and lacks any type of high level documentation. Assuming that this was a due to the fact that a beta client was being examined, we reviewed a few previous versions, including v4.0.3 and v4.0.4, and found the same issues. A full source code analysis is recommended.

A quick search on some of these libraries reveals a Windows-specific remote buffer overflow vulnerability in the mfc71.dll shared library caused by the presence of a 'jmp esp' instruction[4]. Further research must be done in order to ensure that this vulnerability does not pose any harm for users running this client.

Most of the attacks that can be performed on peer-to-peer networks are likely due to a lack of secure client (and client/server) protocols. With BitTorrent, communication protocols could be rewritten to provide secure handshaking, so that clients can be reasonably certain of other clients identities (without impersonation). No information regarding the swarm could be obtained unless a person authenticated themselves accordingly. In combination with secure tracker communication, one could effectively deny spoofing and identity attacks.

In regards to a secure tracker, one technique that could be used to better identify users is to have the tracker establish a peer id for each client, similar to how eMule creates a high/low client ID (recall, the current specification allows the client to specify their own 'unique' peer id).

On the poisoning attacks front, one solution would be to provide incremental part hashes for each chunk. This would be particularly effective for networks such as eMule - where the chunk sizes are fixed. One could publish a supplementary incremental hash file, so that clients can terminate invalid chunks early. Since clients can request any portion of a chunk from another peer, this would be a simple modification. In BitTorrent, the current PyCrypto library is used to generate the chunk hashes (these are stored along with metadata in the '.torrent' file), and can be used to generate backward-compatible incremental hash torrent files. We have currently modified the v4.1.6 make-torrent application so that it generates incremental signatures, as extended meta data for a torrent. Remaining tasks are to have the client recognize incremental hash metadata, and perform the appropriate incremental checks.

Given the state of the BitTorrent specification, a minimalist client can be developed which eliminates the shared library dependencies. This would clearly reduce the amount of possible vulnerabilities, with proper care and use of static analysis tools. Note that BitTorrent was designed for TCP/IP, not for an anonymous

[6]Other projects are Entropy, GNUnet and Nodezilla

[7]py2exe and NSIS are required to build the application and create an installer. It is possible to run this client without building an installer.

layer. Should a client be developed specifically for the layer, there might be vulnerabilities which both close and open.

5 Conclusion

If other peer-to-peer networks approach the issue of security in similar fashion to eMule and BitTorrent, we would be hard pressed to recommend their use in all but the most limited of contexts. Current security in eMule and BitTorrent is poor - out of the attacks examined, there are, at best, poor mechanisms in place which ensure protection against any of them.

An interesting approach for improving the state of identity attacks involves the use of anonymous networking layers, such as I2P and Tor. These layers potentially improve anonymity, but at what expense? A full analysis should be made, as it is possible that improved anonymity opens a client up for other types of attacks.

Since BitTorrent is both open source AND widely used, it makes a perfect candidate for adoption of several of the proposed fixes. Future work can be done to ensure that BitTorrent networks can provide reasonable security guarantees.

References

[1] Peer-to-peer. (n.d.). Retrieved December 15th, 2005, from `http://en.wikipedia.org/wiki/Peer-to-peer`

[2] D. Bickson, Y. Kulbak. The eMule Protocol Specification, January 17th, 2005.

[3] eMule. (n.d.). Retrieved December 15th, 2005, from `http://en.wikipedia.org/wiki/EMule`

[4] eSignal Remote Buffer Overflow. March 30th, 2004. Retrieved December 15th, 2005, from `http://www.securiteam.com/windowsntfocus/5DP0X00CAY.html`

[5] B. Cohen. Incentives Build Robustness in BitTorrent, May 2003.

[6] HBO Attacking BitTorrent. October 4th, 2005. Retrieved December 15th, 2005, from `http://radar.oreilly.com/archives/2005/10/hbo_attacking_bittorrent.html`

[7] Anonymous P2P. (n.d.). Retrieved December 15th, 2005, from `http://en.wikipedia.org/wiki/Anonymous_P2P`

[8] Bittorrent Protocol Specification v1.0. December 13th, 2005. Retrieved December 15th, 2005, from `http://wiki.theory.org/BitTorrentSpecification`

[9] Exploit-JBellz. January 15th, 2003. Retrieved December 15th, 2005, from `http://vil.nai.com/vil/content/v_99963.htm`

[10] Microsoft Security Bulletin MS04-028. December 14th, 2004. Retrieved December 15th, 2005, from `http://www.microsoft.com/technet/security/bulletin/MS04-028.mspx`

[11] eMule. (n.d.). Retrieved December 15th, 2005, from `http://www.answers.com/topic/emule`

[12] D. Hales, S. Patarin. How to cheat BitTorrent and why nobody does. May 2005. Retrieved December 15th, 2005, from `http://p2pnet.net/story/5172`