Exploiting the Security Weaknesses of the Gnutella Protocol *

Demetrios Zeinalipour-Yazti
Department of Computer Science
University of California
Riverside, CA 92507, USA
csyiazti@cs.ucr.edu

Abstract. Peer-to-Peer (P2P) file-sharing systems such as Gnutella, Morpheus and Freenet have recently attracted a lot of interest from the internet community because they realized a distributed infrastructure for sharing files. Such systems have shifted the Web's Client-Server model paradigm into a Client-Client model. The tremendous success of such systems has proven that purely distributed search systems are feasible and that they may change the way we interact on the Internet. Beside the several advantages that have been uncovered by P2P systems, such as robustness, scalability and high fault tolerance various other questions and issues arise in the context of Security. Many P2P protocols are bundled along with an adequate amount of security mechanisms but are proprietary which makes their analysis difficult. The Gnutella Protocol on the other hand is an open protocol, which doesn't highlight security in the sake of its simplicity. Most security weaknesses of the Gnutella Protocol could be avoided if the protocol was taking into account that Peers may misbehave.
In this paper we provide an overview of the Gnutella Protocol Specification, describe several of its weaknesses and show how they can be turned into *Distributed Denial of Service Attacks, User's Privacy Violation and IP Harvesting*. We present the weaknesses with experimental attacks that we have performed on the Gnutella Network. We finally evaluate how these attacks could be avoided and suggest in some cases improvements on the protocol.

1 Introduction

The Gnutella community is increasing day by day at extremely high rates. Clip2 [19] shows that the typical number of peers found in the Gnutella [12] network during a weekday is $43,546$ peers sharing $1,843,549$ files. By the time of this writing another huge network of peers, namely Morpheus [18], which was previously operating over the Kazaa [22] protocol, joined the Gnutella Network. Cnet.com [25] has logged $78,629,070$ downloads of the popular Morpheus Gnutella beta client in a period of 12 days making it the most popular download on the internet.

Beside the tremendous success that these systems meet, they are also facing many security threats. Although we won't consider in this paper attacks or flaws which are not directly related to the Gnutella protocol we mention below a few other "side-effects" of using P2P clients. Recently the Morpheus network suspended its operation after Kazaa, which used to be Morpheus protocol vendor, disabled the protocol from machines that were using the Morpheus Client [12]. After this attack, Morpheus immediately switched its protocol to Gnutella to prevent similar attacks in the future. *Spyware* programs are also a huge problem in the P2P community. A spyware is a program that is usually distributed along with the P2P client and which sends out personal user information. Cydoor [23] is an example of such a program which is distributed along with the some of the most popular P2P clients. *Viruses* of course are another potential thread for the P2P community. P2P systems are just another medium over which viruses can spread efficiently, although

* Course Project for "Seminar in Computer Security" at University of California - Riverside, Department of Computer Science, March 2002.

it is estimated [13] that they are not as dangerous as viruses which are spread over the email. Most Antivirus vendors such as Symantec have already released protection updates for the notorious Trojan horses *VBS.Gnutella* and *W32.Gnuman.Worm*. Finally, *poorly written P2P Clients* are another problem in these networks, since they expose their users to all the security flaws of the particular P2P client. For example, the actual transfer of files in Gnutella is done with the HTTP protocol. This means that each Gnutella Client instantiates a "mini" web-server which is capable of serving other Gnutella Clients after their query was performed over the Gnutella network. These "mini" web-servers are sometimes poorly written making them vulnerable to attacks.

Although all the pre-mentioned threads affect users that use P2P clients they are not a result of the Gnutella Protocol itself. The Gnutella Protocol itself though presents another set of security problems which can jeopardize the normal operation of this community. Security was never a design issues for Gnutella. Gnutella's security is sacrificed in the sake of simplicity and performance. Messages sent across to other peers are sent in plain text and are readable and modifiable by anyone. This results in serious problems such as spamming, where a peer pretends to have "anything" we ask for. We will exploit this weakness in section 4.2 and turn it into an active Distributed Denial of Service Attack.

Encryption is also orthogonal to the way the Gnutella Protocol operates. Since each peer must be able to decrypt the sent message in a reasonably small time it would make no sense to embed encryption in the protocol. This means that we would probably never be capable of protecting a Gnutella user from some malicious user which tries to spy someone else's activity, violating personal privacy in this way.

We believe that most of the threats could be eliminated if the Gnutella protocol was taking into account security. By this we mean that the Gnutella Protocol should enforce the implementation of several features, by stating them clearly in the protocol specification, instead of doing "recommendations". A different design of the protocol could enforce critical, to our opinion, features that this particular P2P protocol should meet.

The focus of our work is to identify security weaknesses of the protocol and to exploit them by setting up proof of concept attacks. The remaining of this paper is organized as follows: Section 2 presents Related Work in this area, Section 3 an overview of the Gnutella Protocol for readers who are not familiar with the Protocol details. Section 4 presents a taxonomy of the Gnutella weaknesses and evaluates how these attacks could be avoided by suggesting in some cases improvements on the protocol. Finally we provide our conclusions and make a reference to future work in Section 5.

2 Related Work.

Although Gnutella has attracted a lot of interest from both the industry and the academic environment regarding its communication and data model [4][5][6], it has not received the adequate interest regarding its security model. To our knowledge, no previous work has been published regarding the security aspects of this Protocol. Some presentations and newsgroup postings are interesting and are hence described in brief below.

S. Bellovin analyzes and compares in his 9th Usenix Security symposium presentation [1], the Napster and Gnutella protocols. His work is mainly focusing on issues such as possible new attacks, traceability of behavior, and privacy but does not provides an insight of how dangerous these issues may become.

Parashar et al. are evaluating in [3] security mechanisms in Peer-to-Peer Applications. This work is not directly related to the Gnutella Protocol but it rather defines a general framework for deploying security mechanisms in P2P networks. They start out by categorizing P2P applications into three categories a) Distributed File Sharing Applications, such as Gnutella, b) Real-time Communications applications, such as Instant Messengers, and c) Distributed Computing applications such as the SETI@Home project [9].

They then discuss how three different Security Enabling Technologies, namely Cryptography, Smart Cards and Steganography can be used to make P2P more secure.

Endeavors Technology's Magi Enterprise software [21], is one of the first highly secure file sharing and communication capable system, which can be as secure as software used across banks and other financial institutions. The company has implemented a full-featured X.509 Public Key Infrastructure (PKI) over a Secure Sockets Layer (SSL) network backbone, which in combination provides the cryptographic standard for Internet e-commerce. The use of X.509 PKI authentication allows security certificates from Endeavors, or from any other recognized X.509 certificate authority, to be used to establish the true identity of any Magi-enabled peer device when it comes on-line and the use of SSL point-to-point security encryption enables each pair of peers that communicate with each other to have a unique key for that pairing. The problem with this technology is that it is proprietary and discovery of security flaws can only be based on reverse engineering techniques, such as the Napster case [14]. Some questions regarding the scalability of the system also arise.

Seth McGann describes in a posting to the Security Focus Online, BugTraq Archive [8], how somebody can spread *Self Replicating Servents*, how to perform *Man in the Middle Attacks* and finally how to implement a *Gnutella Port Scanner*.

Our Contribution In this paper we identify several security weaknesses of the Gnutella Protocol, that were not discussed to our knowledge in any other publication, validate them with proof-of-concept attacks and propose methods of improving the Security aspect of the Protocol.

3 The Gnutella Protocol at a Glance

Gnutella's distributed search protocol [24] allows a set of peers, servents or clients, to perform filename searches over other clients without the need of an intermediate Index Server. The Gnutella network topology is a pure Ad-Hoc topology where Clients may join or leave the network at any time without affecting the rest topology in any sense. The Protocol hence, is designed in a highly fault-tolerant fashion with a quite big overhead [4] of synchronization messages traveling through its network. All searches are performed over the Gnutella network while all file downloads are done offline. In this way every servent that needs to serve a file launches a mini HTTP 1.1 web server and communicates with the interested servents with `HTTP` commands.

The core of the protocol consists of a set of descriptors which are used for communication between servents and also sets rules for the inter-servents descriptors exchange. These descriptors are *Ping, Pong, Query, QueryHit and Push. Ping* messages are sent by a servent that needs to discover hosts that are currently active on the network. *Pong* messages on the other hand are responses to *Ping* requests every time a host wants to come in communication with the peer that requests to join the network (Figure 1, step 3). Although the protocol doesn't set any bound on the amount of incoming or outcoming connections from a particular host, most Gnutella clients come with a default value for these parameters, usually 10, but which are adjustable by the user. A client joining the Gnutella network for the first time may of course not have any clue regarding the current topology or his neighboring peers making the connection to the network impossible. For this reason most client vendors have setup *Host-Caches* Servers which serve clients with IP addresses of servents currently connected to the network (step 1). Another mechanism which is widely deployed, but is not part of the protocol, is the use of local caches of IP addresses from previous connections. In this way a servent doesn't need to connect to an IP acquisition server (i.e. a host-cache) but can rather try to establish a connection with one of its past peers.

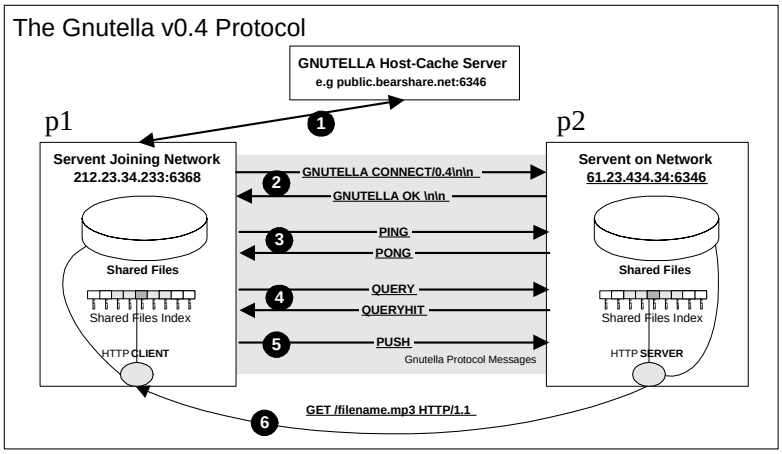

Fig. 1. Gnutella v0.4 Protocol.

Right after a peer has obtained a valid *IP address* and *socket port* of another servent it may perform several queries by sending *Query* descriptors and receive asynchronously results in *QueryHit* descriptors (step 4). Step 6 shows how a servent p_1 may download a file from another servent p_2. This procedure is done offline with the HTTP protocol. If the servent p_2 is firewalled, then p_1 may request from p_2, with a *Push* descriptor (step 5), to "push" the file.

4 Taxonomy of Gnutella Protocol Weaknesses

In this section we will present several different kind of attacks that we have implemented using the Jtella API [10]. Jtella is an open source Gnutella Application Protocol Interface written in Java, which allows the creation of Java based Gnutella Servants with custom behavior (i.e. not according to the Gnutella Specification). We have to mention that most of the commercial Gnutella Clients are vulnerable to our attacks, which can unsettle the regular operation of a big part of the Gnutella network. This happens since the Gnutella security relies to much on the behavior of the participating clients instead of build-in mechanisms which could prevent such attacks.

4.1 Basic Notation

In the rest of the paper, we will make use of the following conventions:
Gnutella Network, denoted as "G", consists of an arbitrary number of inter-connected *Gnutella Peers*, forming a random graph, which might be *connected* or not.
A **Gnutella-peer** or **Servent** denoted as "p", is a regular Gnutella client which has no malicious intention and participates in the ad-hoc topology in order to search and download files.
A **Malicious-peer**, denoted as "Mp", is a modified Gnutella client which instead of behaving according to the recommendations of the Gnutella Protocol performs activity which harms other peers p. We denote that a Mp, "advertises" itself in the network with the best parameters *(i.e. high bandwidth, large number of shared files)*. The obvious reason of doing such as thing is to attract as many as possible "victims" p.
A Host under attack, denoted as "A", might be either another p or any other Internet Service, such as a popular web site (e.g. yahoo.com [17]), which is the target of an attack.
Both Mp and p connect to the Gnutella network, as described in section 3, by exchanging a number of *Ping*

and *Pong* messages. Every p connects to a variable number of k other peers $\{p_0, p_1, ..., p_k\}$, where k depends on the preference of the p client. A typical number for k in Gnutella is 10. Right after a connection to the Network G is achieved a peer might start accepting *Query* messages and respond to them with *QueryHit* messages if it meets the requirements of the given query.

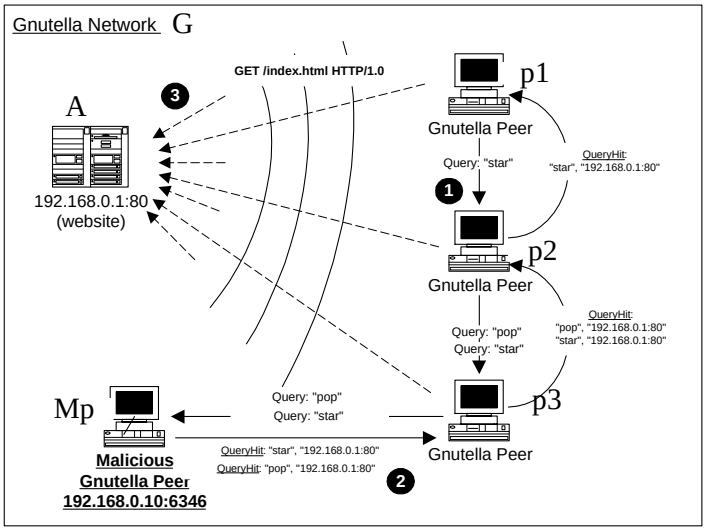

Fig. 2. Performing a Distributed Denial Service on a public webserver (e.g. yahoo.com).

4.2 Distributed Denial of Service Attack through Spamming

Spamming is well known in the Email World. We all receive unsolicited emails from time to time, with advertisements or sometimes even viruses. Although some user may easily trash a spammed email message; and there would be no other consequence for him, the same wouldn't happen with a spammed query result in the Gnutella Network. If we accept to download a spammed query result we might become an active part of a *Distributed Denial of Service (DDoS) Attack* against some Internet Host.

Gnutella has no provisions to thwart Distributed Denial of Service Attacks. DDoS attacks are probably the first thing that comes in somebody's mind when we think of a huge number of "uncontrolled" participating hosts. Unfortunately no provision has been taken to avoid such attacks, neither in the protocol nor in the majority of the Gnutella clients. A DDoS attack can easily be achieved in Gnutella since a Mp may "lie" by answering positively to any query it receives and urge p to download a resource from A, which is the host under attack. A Denial of service threat, according to E. Amoroso [2], arises whenever access to some computer system resource is intentionally blocked as a result of malicious action taken by another user. Although this definition is not exactly the case in Gnutella, since a p which is participating in the attack is not intentionally creating the load on A, but it simply tries to download the file it was said to be there, this kind of attack is still considered a DoS. Such attacks are also extremely difficult to identify, on A behalf since requests are launched from many different IPs and domains.

For the purpose of the experiment we have configured an Apache Webserver [15] on a Pentium III, 900MHz Windows host with 100Mb LAN connectivity. The Web-server A will be the target of the attack performed by Mp. We have also modified the default `LogFormat` attribute in the configuration file of apache, *conf/http.conf*, in such a way that it will log the `User-Agent` of the requesting Peer. This will help us to

list the Gnutella Clients that are actually vulnerable to this attack. A detailed overview of the topology is shown in Figure 2. As you may see p_1 and p_2 are both performing a *Query* with different keywords in *step 1*. p_3 is simply forwarding the message to Mp which is the malicious host. M_P will reply to both messages, in *step 2* with a *QueryHit* response urging them to download the file from A. We have to recall that Mp *QueryHit* message is very "attractive"; since it indicates that the file is located on a server with a LAN connection. It moreover tries to make the filename as much similar, to the initial query, as possible. In our experiment we have chosen to reply to all messages with a "dressed" filename that consists of the initial keywords along with a random extension of popular media files, such as *("mp3", "mpeg", "asf", "wma")*. We have also chosen a random filesize which was in the range $1Mb - 10Mb$ in order to make the results as attractive as possible. Gnutella clients such as LimeWire would rank our result as a top quality result, since the filename contains all the query keywords.

The generated load on A could be so high that A would be unable to accept additional connections or even bring it down in some cases. In our case, we early realized that our Apache Server was not able to respond to the generated load. That happens since our *fake responses* may propagate to many p , which subsequently may try to download the file from A. A must allocate some local resources in order to service the fake request. In our experiment, as we will show later on, we were able to respond to $7,344,838$ *Query* requests in an 8 hour period. Of course all $7,344,838$ replies may not reach the designated users since many messages can be discarded, due to the adhoc nature of the network. Moreover all remaining users may not rush to download the file Mp is pretending to be on A. Although not related to the attack. It is also interesting to mention an approximation of the overhead generated by each *QueryHit*. If we assume that each p is connecting to averagely 5 other peers (which is usually much higher) and will forward a message only in a depth of 7 (i.e. $TTL=7$), which is also typical for a Gnutella client and assuming that each peer is using *message flooding*, which means that Mp fake responses will be sent to all of its peers, we will averagely reach $4^6 * 5 = 20,480$ other p. What we actually want to see in this experiment is how dangerous this attack can turn.

Maximizing the Number of Polluted Peers *p*. An extension of this idea with which Mp can maximize the number of p it pollutes, with spammed results, is by invoking a very simple technique which is also shown in Figure 3. Suppose that Mp connects to a Gnutella IP Host-Cache Server and obtains randomly a number of k hosts which are connected to the network. Immediately after that it establishes a connection with the k peers, it pollutes them with spammed *QueryHit* messages for a short period and disconnects from them. Although Mp may have cancelled its operation, any p which received a spammed message will probably rush to download the non-existent file. Mp can immediately re-connect to the Gnutella Host-Cache Server, obtain another random number of k hosts and repeat the same procedure. In this way a Mp may be able to pollute many different sub-graphs of the Gnutella Network G at a given time.

Another important aspect of this attack is that most of the p will insist on trying to download the file from A on a virtually endless basis. We have observed peers trying to download the file from the Web-Server (after Mp was shutdown), well after a period of 20 hours. This happens because most Gnutella clients are designed in such a way that when a download fails, the client will automatically try to download the file again after a given interval. We have observed that this interval is sometimes very small (e.g. 2 seconds) which subsequently means that the cheated p will automatically try to re-establish a connection with A after 2 seconds. That means that the attack will be re-launched automatically from the participating cheated hosts on a clock-wise basis or until the user of p chooses to manually cancel the download. Most of the times, this is not the case because many users choose to download a file and then may leave the system running for several hours or until their download is satisfied.

We finally mention that some unknown sources are already taking advantage of this security flaw, with the difference that they don't intend to create a DDoS attack but they want to create some extra traffic

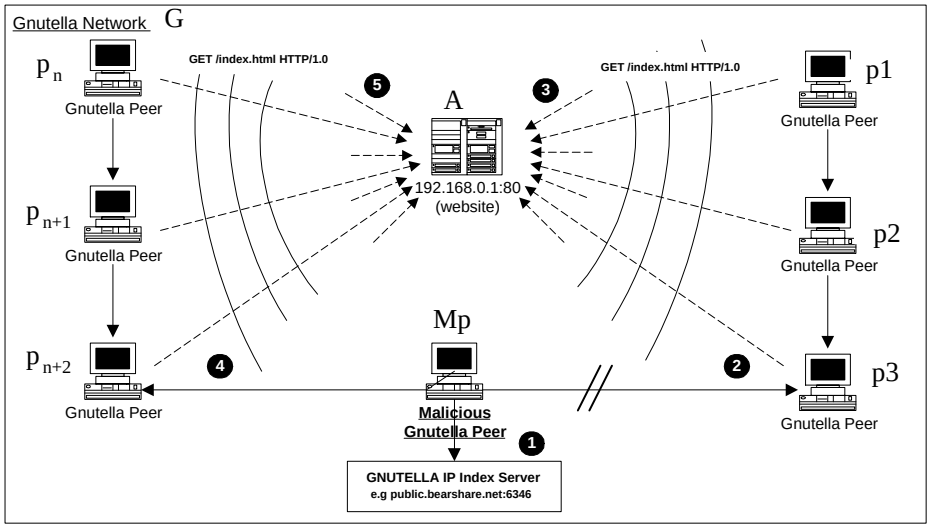

Fig. 3. Maximizing the number of p that Mp pollutes with Spammed results.

on some web sites for commercial purposes. More specifically a malicious Mp replied to any *Query* with a *QueryHit* which prompts a Gnutella users to download an HTML page. As soon as the user downloads and launches the specific page, he is automatically redirected to a commercial web site.

Table 1 presents some of the Gnutella Clients which are vulnerable to the DDoS through Spamming Attack. We can see that the most of the commercial Clients are affected by this attack. We need to mention that most of the requests made to our Apache Server had no User-Agent field in the HTTP request, which indicates that many other unlisted Clients are also affected by this attack

User-agent	Version
BearShare	2.3.0, 2.4.2, 2.4.4, 2.5.0b12
Gnotella	1.0.5
Gnucleus	1.6.0.0, 1.6.1.0
LimeWire	1.6d, 1.7c, 1.8, 2.1.3, 2.2.1
Napshare	1.0
Mutella	
Oasis	2.0.0
Qtella	0.4.0

Table 1. User-Agents of Gnutella Clients which are vulnerable to the DDoS Attack.

How far can the Spammed QueryHit Messages Spread? Estimating how far our spammed messages can spread is probably a difficult question to answer since it depends on the number of p that are connected to the Gnutella network, the random topology at the time of attack (i.e. is the network a connected graph), the number of hosts to which some peer connects, the maximum TTL to which a peer is set and finally the percentage of messages lost due to disconnections or failures.

J.Riiter, one of the founding developers of Napster [20] presents in [11] an interesting set of mathematical

equations which aim to calculate the reachability, capacity, and bandwidth throughput of the Gnutella Network. His assumptions are too broad, which makes it difficult to validate the correctness of his equations. We consequently performed an experiment with 30 misbehaving peers $\{Mp_0,\ Mp_1, ..., Mp_{29}\}$ distributed on 3 hosts, in order to get a better understanding. Each host launched 10 Mp, each of which connected to a Gnutella Host-Cache Server to obtain a list of peers p currently connected to the Gnutella Network. After a period of 2, 4, 6, 8 hours we connected to the Gnutella Network with a commercial Gnutella client, namely Morpheus, and performed several queries to see if we can actually receive an answer from one of the Mp. In all cases we were able to receive a spammed message from one or more of the 30 Mp. After shutting down the 30 Mp we observed that the misbehaving peers answered to $7,344,838$ *Query* requests in the 8 hour period. In most occasions the spammed message indicated that the resource is available 7 hops away, which is typically the maximum TTL in Gnutella. This experiment proves that spammed messages can affect a large number of Gnutella users p and spread confusion all over the Gnutella Network.

Attack Results. Although the Apache server helped us to figure out which clients were vulnerable to the DDoS attack it couldn't provide us an insight of the actual load since it simply rejected most of the requests; which were subsequently not logged, or collapsed in some cases. The Apache Server was operating without any serious problems at the 14 *requests/second* rate. At higher rates though, it was not behaving appropriately. For this reason we decided to shield the web server with a software firewall (see figure 4), namely ZoneAlarm [16], which would have a smaller overhead of logging the attempted TCP/IP connections. Interestingly, we found that even with the firewall was not capable of gathering some concrete results, since it also collapsed after the request's rate increased. More specifically, we mention that the firewall "froze" after the 80 *requests/second* rate.

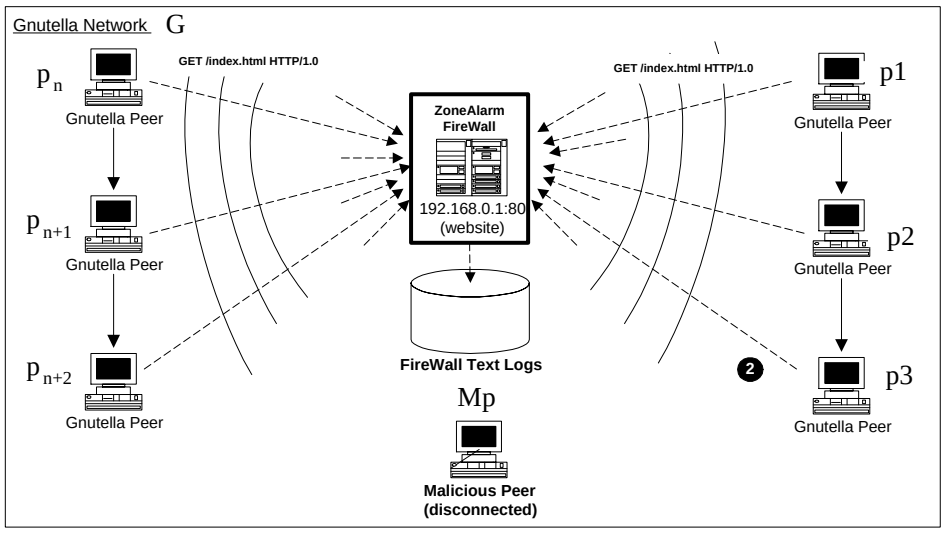

Fig. 4. Logging p_i requests in the ZoneAlarm firewall log, while Mp has disconnected.

In order to gather a $1,41$ MB log file which consists of $17,299$ logged requests, we restarted the firewall several times. After the rate increased beyond 100 *requests/second* we were unfortunately not able to bring the firewall up since it collapsed on initialization. Many system crashes also occurred so we were not able to gather concrete results. We believe that if we were using a different platform and a different

set of tools we would be able to gather better results. Although these problems, we extracted some useful statistics which are not affected by the fact that the requests have not been taken contiguously.

Description	#
Total Number of Logged Requests.	17,299 *requests*
Size of Log File.	1,41 *MB*
Unique # of IP Addresses found in log.	6,544 *addresses*
Maximum # of requests done by 1 IP address.	318 *requests*
Maximum # of requests per second.	104 *requests/second*

Table 2. Statistics gathered from ZoneAlarm Firewall Log File.

Table 2 provides a set of statistics gathered from the ZoneAlarm Firewall Log File. In order to extract and manipulate the data we wrote a custom *ZoneAlarm Log Analyzer*. Although the system was running for 1 hour it basically collected data only for 19 minutes due to the system crashes that were described before. As we can see from the table, the Unique number of IP addresses that did a request from our Webserver was 6,544, which implies that a host is requesting a particular document several times, since the total number of Logged Requests is 17,299. This validates our observation that a Gnutella Client will insist on downloading a particular spammed *QueryHit*. Supportive to this fact is also the Maximum number of requests done by 1 IP address; which is 318 and which again shows that a cheated Gnutella client will return after some interval to reclaim the file. The table also shows that the generated load is considerably high (104 requests/second), taking into account that this number could be much larger if we were able to log all the requests.

Traceability. The Gnutella Protocol does not define or does not even encourage the deployment of any auditing system. The fourth fundamental requirement in the Orange Book is accountability: "Audit information must be selectively kept and protected so that actions affecting security can be traced to the responsible party" [7]. In the case of a DDoS Attack against a commercial host on the internet, there will be no evidence of who launched the particular attack. An even more scaring scenario is that somebody performs several attacks on many different Internet hosts and there will be no way to prevent or eliminate his attacks; since shutting down the network is theoretically not feasible.

Shielding Peers from the DDoS Attack through Spamming It is obvious that spamming is probably one of the most difficult things to solve in P2P file-sharing systems such as Gnutella. This happens because a client has no liability against the queries it receives. Although the protocol specification states clearly that "a peer should only reply to a *Query* with a *QueryHit* if it contains data that strictly meets the Query Search Criteria", that is simply not enough. We instead believe that if the protocol was slidly changed then a download activity of some peer p would not turn into a DDoS attack against some web server A. We propose that p should upon receiving a *QueryHit* from Mp, try to validate that Mp really holds the file that it claims to have. This could be achieved by an extra *Query/QueryHit* message exchange between p and the host A. In order to achieve this p should try to connect to A (with the *Ping/Pong* pair) and consequently try to exchange the *Query/QueryHit* pair. Since A doesn't know anything about the particular file and may even not know anything about the Gnutella protocol (e.g. it is a web server), it will simply reject the request. p will know from that point that is was misdirected and cancel any further activity. This change on the protocol has a relatively small overhead since it results in the exchange of

one extra *Ping/Pong* and one *Query/QueryHit* pair but it would save p from passively participating in a DDoS attack. It would also save many resources from p since it would not need to re-establish a download session with A every 30 seconds.

4.3 The *Pong* attack.

In this subsection we are going to present another type of DDoS attack that can easily be achieved with the Gnutella Protocol. Although this attack is much weaker than the attack presented in the previous subsection, and should probably not be considered as important, it proves that the protocol design is inadequate. As we have already described in section 3, after a peer p_1 sent out a *Ping* to another peer Mp, Mp will respond with a *Pong* message. A typical *Pong* message, which is shown in Figure 5, contains the IP address and the port of Mp.

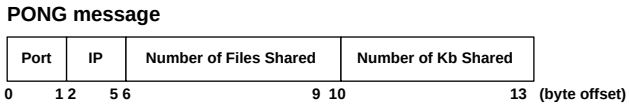

Fig. 5. A Gnutella *Pong* message Peers IP and Port can be set to any number.

Since Mp is a misbehaving peer it will respond with a *Pong* which contains the IP and the PORT of another host A which is the host under attack. p_1 would subsequently think that it established a connection with A, cache $A's$ address instead of $Mp's$ and forward all *Query* messages to A. In figure 6 we can see analytically the 3 steps that were described above.

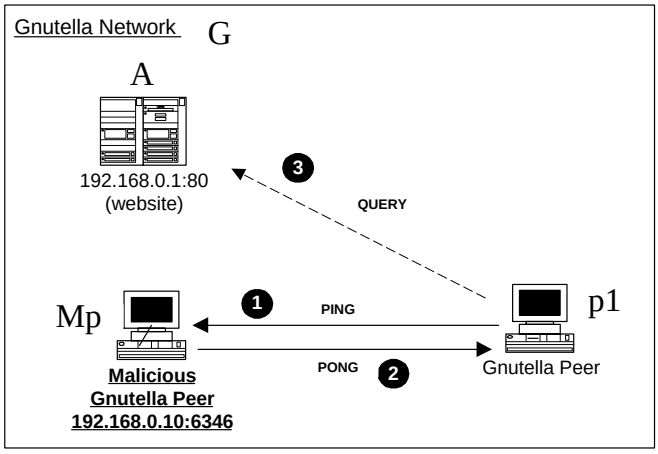

Fig. 6. p thinks that it is connected with A, instead of Mp, and forwards to A all *Query* Messages.

As we mention in the beginning of this subsection, this attack is not comparable to the DDoS through Spamming because this attack will last only for a short period. This happens because p_1 will after a short interval re-send a *Ping* message to all the hosts in its cache in order to re-discover the topology. Since A is not really a Gnutella client and is not able to reply with a *Pong* message to $p1's$ request, p_1 will simply remove A from its cache and the attack will terminate.

4.4 IP Harvesting.

A typical internet user has an IP address which is never broadcast and is not generally known to anyone else other than the Internet server on which this user connects. The nature of P2P networks though requires a user to advertise his address to other Peers in order to achieve connectivity with them. It is well known that malicious hackers are constantly scanning the Internet for IP addresses and that they try to break into these systems if the systems are not secured adequately. These hackers usually ping random IPs with the hope that some IP might be up and running a specific service, such as a Webserver, or are targeting high-visibility Internet Servers that have a well known address. The Gnutella network might facilitate the activities of these persons since harvesting IP addresses in the Gnutella network is very efficient. Another interesting issue is that users of the Gnutella Protocol are usually sharing some common behavior, which could be translated in terms of Operating System they run, servers that are running on their hosts (i.e. their Gnutella Client along with its mini Web Server) as well as their Security Ignorance. A Malicious hacker which identifies a security flaw in one or more of the poorly written Gnutella Clients could harvest for IP addresses and then launch a massive attack against the Gnutella users. This kind of attack is obviously easier for a hacker than trying to hack into a high-visibility Internet Server which is usually shielded with the latest protective software, such as firewalls. The IP addresses that somebody can discover are usually not very helpful after some time since most users of such systems are either using dial-up connections; hence their IP address changes over time, or are assigned dynamic addresses with protocols such as DHCP. The problem is bigger though for users which have static IP address and might run the Gnutella Clients around the clock. An example of such users, are the users of the ".edu" domain, which are estimated [26] to contribute with 10% of the Gnutella volume.

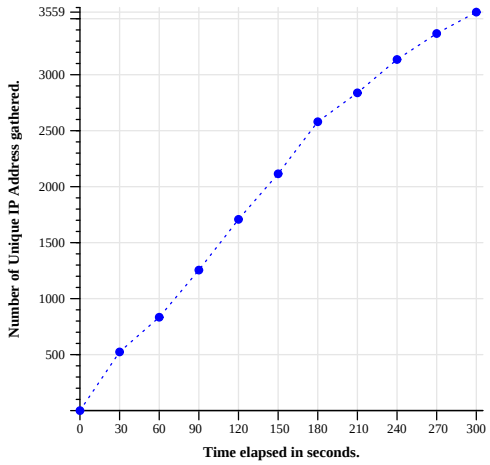

Fig. 7. Discovering IP Addresses from the Gnutella Network can be done efficiently.

In order to demonstrate how many IP addresses somebody can actually obtain from the Gnutella Network we have written an IP-Harvesting Engine. Given that each host can arbitrary disconnect from the network we need to be able to discover their IP addresses in a short time. For the purpose of the experiment, we have implemented, again with the Jtella API, a multi-threaded IP-Harvesting Engine. The engine connects to a set of pre-specified Gnutella Host's Caches, obtains an initial set of peers which are active in the network and tries to connect with them collecting each time their IP address. In figure 7, we presents how we have discovered $3,559$ unique IP address in a period of 5 minutes. IP addresses are

tailored along with the port numbers of the corresponding Gnutella clients. This information could facilitate malicious hackers which are seeking for "fresh" IP addresses.

4.5 Other Minor Weaknesses.

Injecting Viruses through the *Push* leak. In Gnutella a peer p_1 may send a Push descriptor, shown in Figure 8, if it receives a *QueryHit* descriptor from another peer p_2 that doesn't support incoming connections. This might happen when p_2 is behind a firewall or when p_2 claims to have a file that doesn't exist on its mini web-server.

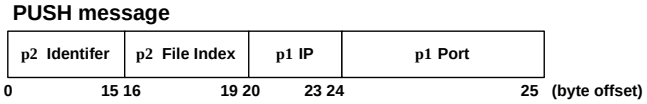

Fig. 8. A Gnutella *Push* descriptor sent by p_1 and received by p_2.

Upon receiving the *Push* descriptor, p_2 should try to establish a TCP/IP connection with p_1 on the given IP and PORT address and after exchanging another pair of descriptors, which are not significant in our context, transmit p_2 file's contents. The problem that arises with the *Push* message is that p_2 may inject viruses to p_1 at any time since p_1 blindly accepts whatever is sent by p_2. The only thing that needs to be done from p_2 behalf is to initially "lie" to any *Query* it receives by replying with a *QueryHit* that contains a file that doesn't exist. We have not validated the hazard of this attack but we need to mention that some Gnutella Clients, such as LimeWire, will filter out "dangerous files" (e.g. files having a **.vbs* or **.exe* extension).

Violating User's Privacy through GUID Tracing. GUID, which stands for Gnutella Unique Identifier, is a 16-byte string "uniquely" identifying a descriptor *(Ping, Pong, Push, Query and QueryHit)* that is sent along the Gnutella network. Each peer upon receiving a descriptor from the network should log the descriptor's GUID in order to use it for future purpose. For example if peer p_k forwards a *Query* message to a number of other peers it needs to log the GUID string in order to be able to decide, in the future, if it should forward the *QueryHit* message it receives. The Gnutella protocol strongly encourages implementer's of Gnutella Clients to discard responses, such as *QueryHit*, messages that were not seen before. GUIDs are generated locally by each peer, since there is no central authority that could distribute GUIDs. Hence it relies on the Gnutella Client implementer of how these unique strings are generated. Many implementations, such as the Jtella API, use random GUID generators which have no security risk. S. Bellovin though mentions in [1] that GUID in Windows 95/98/NT clients contain the hardware MAC address, which is constant over time and which subsequently could be used to link requests over time. This phenomenon is apparently dangerous since it might enable a malicious hacker to track user's queries and downloads violating in that way the anonymity or privacy of the particular user.

5 Conclusions & Future Work

Although Gnutella has attracted a lot of interest from both the industry and the academic environment regarding its communication and data model, it has not received the adequate interest regarding its security model. It thus presents several security weaknesses which can easily be exploited. Most security

weaknesses of the Gnutella Protocol could be avoided if the protocol was taking into account that Peers may misbehave.

In this paper we provided an overview of the Gnutella Protocol Specification, described several of its weaknesses and showed how they can be turned into *Distributed Denial of Service Attacks, User's Privacy Violation and IP Harvesting*. We presented the weaknesses with experimental attacks that we have performed on the Gnutella Network. We finally evaluated how these attacks could be avoided and suggest in some cases improvements on the protocol.

We are currently working on extending our experiments in order to cover flaws that were not validated with experimentation, such as the *injection of viruses through the* Push *leak* as well as *User's privacy violation through GUID tracing*. We also want to perform the DDoS denial attack on a higher capacity experimental server together with some more sophisticated log analyzing tools in order to gather some more concrete results.

References

1. Steven M. Bellovin, "Security Aspects of Napster and Gnutella", 9th Usenix Security Symposium Presentation, Denver, Colorado, August 2000.
2. Edward G. Amoroso, "Fundamentals of Computer Security Technology", ISBN 0-13-108929-3, Prentice-Hall 1994.
3. M. Parashar, M. Agarwal, S. Arbeeny, V. Bhat and R. Chowdhury, "Evaluating Security Mechanisms in Peer-to-Peer Applications", Department of Electrical and Computer Engineering, Rutgers University., 2001
4. Beverly Yang and Hector Garcia-Molina, "Comparing hybrid peer-to-peer systems", Proc. 27th Int. Conf. on VLDB, Rome, 2001.
5. H. Stoica, I. Morris, R. Karger, D. Frans Kaashoek, M. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications", SIGCOMM 2001.
6. S. Gribble, A. Halevy, Z. Ives, M. Rodrig, D. Suciu. "What Can Databases do for Peer-to-Peer? WebDB Workshop on Databases and the Web, June 2001.
7. National computer Security Center. "Department of Defense Trusted Computer System Evaluation Criteria", 1985.
8. Seth McGann, "Self-Replication Using Gnutella", Security Focus Online, BugTraq Archive, May 2000, *http://online.securityfocus.com/archive/1/59387*.
9. "The SETI@home (Search for Extraterrestrial Intelligence at Home) Project", UC Berkeley, *http://setiathome.ssl.berkeley.edu/*.
10. Ken Mccrary, "The JTella Java API for the Gnutella network", October 2000, *http://www.kenmccrary.com/jtella/*.
11. Jordan Ritter, "Why Gnutella Can't Scale. No, Really.", February 2001. *http://www.darkridge.com/~jpr5/doc/gnutella.html*
12. John Borland, "Morpheus' downfall: Bills weren't paid", News.com Article, March 2002, *http://news.com.com/2100-1023-851330.html*.
13. John Borland, "Gnutella viruses weaker than email bugs, experts say", News.com Article, June 2000, *http://news.com.com/2100-1023-241440.html?legacy=cnet*.
14. "Reverse-engineered Napster Protocol Specification"., SourceForge, *http://opennap.sourceforge.net/napster.txt*.
15. "The Apache HTTP Server Project", Apache.org, *http://httpd.apache.org/*.

16. ZoneAlarm Firewall, Zone Labs Inc., *http://www.zonelabs.com/*.
17. Yahoo!, Yahoo.com., *http://www.yahoo.com/*.
18. Morpheus. , MusicCity.com, *http://www.musiccity.com/*.
19. Clip2. , Clip2.com, *http://www.clip2.com/*.
20. Napster, Napster.com, *http://www.napster.com/*.
21. Magi Enterprise, Endeavors Technology, *http://www.endeavors.com/*.
22. Kazaa, Kazaa.com, *http://www.kazaa.com/*.
23. Cydoor Spyware, Cydoor Technologies, Inc. *http://www.cydoor.com/*.
24. Gnutelliums, Gnutella, *http://www.gnutelliums.com/*.
25. Cnet.com, *http://www.cnet.com/*.
26. The GnutellaMeter, *http://www.gnutellameter.com/gnutella-hosts.html*.