# Misusing Unstructured P2P Systems to Perform DoS Attacks: The Network that Never Forgets

Elias Athanasopoulos and Evangelos P. Markatos

Institute of Computer Science (ICS)
Foundation for Research & Technology Hellas (FORTH)
{elathan,markatos}@ics.forth.gr

Technical Report 370
ICS-FORTH, February 2006

**Abstract**

Unstructured P2P systems have gained great popularity in recent years and are currently used by millions of users. One fundamental property of these systems is the lack of structure, which allows decentralized operation and makes it easy for new users to join and participate in the system. However, the lack of structure can also be abused by malicious users. We explore one such attack, that enables malicious users to use unstructured P2P systems to perform Denial of Service (DoS) attacks to third parties. Specifically, we show that a malicious node can coerce a large number of peers to perform requests to a target host that may not even be part of the P2P network, including downloading unwanted files from a target Web Server. This is a classic form of denial-of-service which also has two interesting characteristics: (a) it is hard to identify the originator of the attack, (b) it is even harder to stop the attack. The second property comes from the fact that certain unstructured P2P systems seem to have a kind of "memory", retaining knowledge about (potentially false) queries for many days. In this paper we present real-world experiments of Gnutella-based DoS attacks to Web Servers. We explore the magnitude of the problem and present a solution to protect innocent victims against this attack.

## 1 Introduction

With the explosion of file sharing applications and their adoption by large numbers of users, P2P systems exposed a wealth of interesting design problems regarding scalability and security.

In this paper we examine one such problem that may arise from the implicit trust that is inherent in current unstructured P2P systems. We show how a malicious user could launch massive DoS attacks by instructing peers to respond positively to all queries received, pretend to provide all possible files, and pretend to share everything. Such peers usually respond to all queries in order to trick ordinary users to download files which contain garbage, advertisements, or even malware. In this paper we show that the responses provided by such bogus peers may result in DoS attacks to unsuspected victims, which may not even be part of the P2P network. Indeed, with modest effort we have managed to develop techniques, which, if adopted by bogus peers, can result in DoS attacks to third parties by redirecting a large number of peers to a single target host. In a nutshell, whenever they receive a query, these bogus peers respond by saying that the victim computer has a file that matches the query. As a result, a large number of peers may try to download files from the unsuspected victim, increasing its load significantly. Furthermore, we have developed mechanisms which *trick* this large number of peers to actually download files from the unsuspected victim. To make matters worse, in our methods, the victim does not even need to be part of the P2P network but could also be an ordinary Web Server. Therefore, it is possible for a significant number of peers attempt downloading files from a Web Server, increasing its load and performing the equivalent of a DoS attack.

The rest of this paper is organized as follows. Section 2 presents the architecture of the Gnutella P2P system focusing on the lookup and data transfer process. Section 3 illustrates the techniques we developed to perform DoS attacks by misusing the Gnutella system, and Section 4 presents experiments for the measurement of the effectiveness of the DoS attacks. Section 5 presents an algorithm to protect third parties from Gnutella based DoS attacks. Section 6 provides an overview of related work and Section 7 summarizes our findings and presents directions for further work.

## 2 Gnutella Architecture

The Gnutella system is an open, decentralized and unstructured P2P system. This section describes the architecture of the Gnutella system and highlights the basic components that are used as part of the attack.

### 2.1 Query-QueryHit exchange mechanism

Information lookup in the Gnutella system is performed using Query flooding or controlled Query flooding, known as Dynamic Querying. In both cases, nodes broadcast a Query packet, which embeds the search criteria, to some or all of their first-hop neighbors. The Query packet is forwarded to the system until its TTL becomes zero. In each forwarding step the TTL of the Query packet TTL is decremented by one and a "HOPs" counter is incremented. Along the paths on which the Query propagates, every node of the system is free to answer by issuing a QueryHit packet. A QueryHit packet travels back to the originator of the Query following the same path of the Query packet. It is important to note that there is no central mechanism to confirm whether peers generating QueryHit packets actually hold a file that matches the search criteria of the original Query.

A QueryHit packet consists of a standard Gnutella header describing the TTL and HOPs of the packet and the actual QueryHit payload. Among other fields, the QueryHit payload specifies the IP address and the port number of the node holding the requested data file and a list of entries matching the search criteria of the Query. Each entry is formed by the file name of the object, its local index and sometimes a SHA1 hash, to assist in the parallel download of a file from multiple locations (e.g., *swarming*). Upon receiving a QueryHit the node that issued the query can directly connect to the host listed in the QueryHit packet and try to perform the download. It is important to note that there is no central authority to verify that the IP address and the port number listed in the QueryHit packet match the IP address and the port number of the node issued the QueryHit packet. In addition, a peer may generate QueryHit messages with a spoofed "HOPs" field, to imitate QueryHits that have been generated by another peer.

### 2.2 Data transfer protocol

The actual data transfer among two Gnutella peers is performed using an HTTP based request/response mechanism. Specifically, when a servent receives a QueryHit and is willing to download the data file, it connects to the IP address and port number listed in the QueryHit packet and issues a request that has the following form:

```
GET /get/<File Index>/<File Name> HTTP/1.1\r\n
User-Agent: Gnutella\r\n
Host: 123.123.123.123:6346\r\n
Connection: Keep-Alive\r\n
Range: bytes=0-\r\n
\r\n
```

On the receiving end, the servent generates a response in the following form:

```
HTTP/1.1 200 OK\r\n
Server: Gnutella\r\n
Content-type: application/binary\r\n
Content-length: 4356789\r\n
\r\n
```

It is important to note that the data transfer process is an HTTP-like transaction that is almost identical to an HTTP transaction between Web browsers and Web Servers. One small difference is the "`/get/<File Index>`" part of the initial GET request, which is entered only by Gnutella servents and not by Web browsers.

## 3 Exploiting Gnutella

The content lookup process and the data transfer mechanism can lead to serious attacks against the system itself but also against third parties. Malicious users can exploit the absence of a mechanism for verifying the integrity of the information exchanged among peers and pollute the system with fake information. In this Section we explore techniques that can lead to DoS attacks against any machine connected to the Internet and to degradation of the Gnutella system itself.

### 3.1 Exploiting the Query-QueryHit mechanism

As already noted, there is no central mechanism to verify that a node which replies with a QueryHit to a Query is trustworthy. That is, malicious nodes can reply to any Query they receive with a QueryHit which embeds the IP address and the Port number of any remote server. For example, a malicious node can reply to every Query it receives and redirect peers to another Gnutella peer. In the general case, a QueryHit can embed the IP address and the Port number of any computer machine connected to the Internet, including Web Servers. This may lead a large number of Gnutella peers to connect and try downloading a non-existent data file from the Web server. The Web server may respond with an HTTP 404, meaning that it was unable to locate the requested data file. As we will show later in our experiments, Gnutella peers will persistently try to to download the data file from the Web server, even though they have received an HTTP-level failure message.

### 3.2 Exploiting the HTTP protocol

A large number of HTTP requests that result in a 404 response may not be difficult to handle for a Web Server. The attack can be more efficient if we can force the Gnutella peers to perform an actual download from the Web server. The download may not even be relevant to their search criteria server. This can be achieved by embedding a specifically constructed file name in the QueryHit packet. For example, consider that a Query with search criteria "foo bar" is received. The file name:

```
../../live HTTP/1.0\r\n\r\nfoo bar.mp3
```

will be displayed to the user's client as:

```
../../live HTTP/1.0____foo bar.mp3
```

If the user decides to download the above data file, the targeted Web server will receive a request[1]:

```
GET /get/1/../../live HTTP/1.0\r\n\r\nfoo bar.mp3 HTTP/1.1\r\n
User-Agent: Gnutella\r\n
Host: 123.123.123.123:6346\r\n
Connection: Keep-Alive\r\n
Range: bytes=0-\r\n
\r\n
```

Since any Web server will try to process the request as soon as it parses the `\r\n\r\n` sequence, the above request is equivalent to:

```
GET /get/1/../../live HTTP/1.0\r\n\r\n
```

which, in turn, is equal to:

```
GET /live HTTP/1.0\r\n\r\n
```

Assuming there is a data file with the filename "live" in the Document Root of the targeted Web Server, the tricked Gnutella peers will be forced to download it. If the file is large, the Web Server may be unable to cope with the incoming malicious requests.

[1] Note that we have constructed an HTTP 1.0 request, which is accepted by the majority of current Web Server software.

### 3.3 Attacking Gnutella

Malicious peers that use this attack are hard to detect by the Gnutella system. A malicious peer can spoof the HOPs field of a Gnutella message and thus hide its origin. Consider that peer A constructs a QueryHit message and sends it to its neighbors, but instead of inserting a HOPs=0 field, it inserts a HOPS=s field. The neighbors of A, upon receiving the QueryHit (which in reality is constructed by A), will think that another peer has constructed the QueryHit and forwarded it to A.

A peer which receives fake QueryHits may consider the node which constructed these QueryHits as a spammer. Unfortunately, the identity which is embedded in the QueryHit message is not the identity of the peer which constructed the QueryHit message. That is, Gnutella peers believe that it is the victim peer that produces spam messages.

The problem of isolating spam nodes is actively investigated by the developers of most Gnutella clients. To the best of our knowledge, the identification of spammers is currently still a manual process. If some IP addresses of a subnet qualify as belonging to nodes generating spam, the whole subnet is isolated from the system by entering its range to a “black list” used by the Gnutella client. Although this is a slow process requiring human intervention, a few misbehaving nodes can lead to isolation of large subnets, something that an attacker might exploit. This means that an attack to the Gnutella itself can be constructed by producing fake QueryHits, which embed IP addresses of active Gnutella peers that belong to known large subnets. These IP addresses are easy to be collected by crawling the Gnutella system[8].

## 4 Experimental Results

We performed experiments in order to measure the effect of a DoS attack produced by the Gnutella system to Web Servers of our lab. The Web Server software which we used was the standard pre-configured Apache[2] of the Debian GNU/Linux Operating System[3]. We modified a well-known Gnutella servent[4] to respond to every Query it receives with one and only QueryHit. This policy was one of the fundamental choices that we made, since we wanted to perform an attack with minimal effort. The Web Server was installed on the same host with the Gnutella servent. The Gnutella servent was trying to maintain from 80 up to 100 simultaneous connections to the Gnutella system and besides issuing QueryHits it operated at the ultrapeer level implementing the whole Gnutella protocol as is. In the rest of this section we present the results of our experiments.

### 4.1 Simple Query-QueryHit Exploitation

In the first experiment, our malicious servent was replying to every Query it received with one and only QueryHit which embedded a file with advertised size of 3,240,000 bytes. The filename provided in the QueryHit was the product of

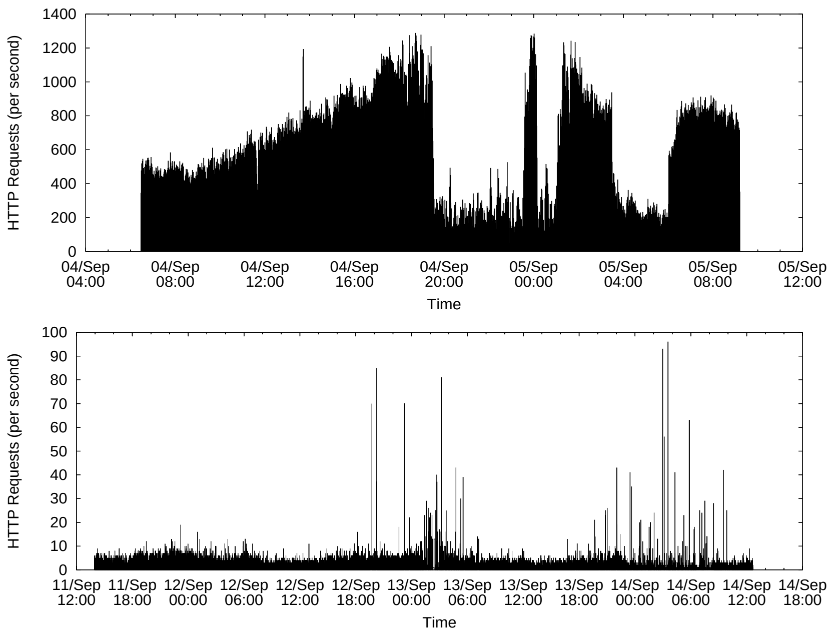

**Fig. 1.** The rate of HTTP Requests per second that our Web Server recorded. The first graph presents the period that our malicious client was connected to Gnutella. The second one presents the period after over 10 days of the experiment.

| Duration | 21 mins | 1h 9 mins | 4 h |
|---|---|---|---|
| QueryHits Generated | 10,000 | 100,000 | 1,000.000 |
| Downloads Recorded | 696 | 1,017 | 6,061 |
| Unique IP Addresses | 30 | 332 | 1,988 |

**Table 1.** Statistics collected for the experiment, in which our Web Server actually serves a file with 3,240,000 bytes.

the concatenation of the search criteria with the ".mp3" extension. That is, a Query for "foo" had as a result a QueryHit for "foo.mp3". Again, following the "least effort" principle, we chose not to create a sophisticated engine that will eventually understand the semantics of the Query, such as the format and size of the file the remote user issuing the Query wants to download. Instead, we chose to have the malicious servent generate QueryHits in the most obvious and naive way.

We connected our malicious servent to Gnutella and let it to answer every incoming Query for a period of two days. Figure 1 presents the requests which were logged in the Apache log file by our Web Server. After careful examination of the Apache log file, we found out that our Web Server had also recorded some

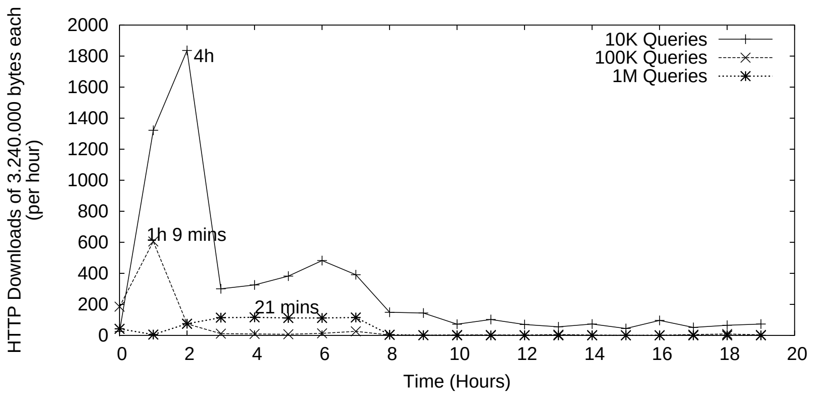

**Fig. 2.** Download rate (per hour) of a file of 3,240,000 bytes from the Web Server. Each curve has a caption noting the time needed for the malicious client to answer the amount of Queries.

| Duration | 10 mins | 1h 10 mins | 4 h 30 mins |
|---|---|---|---|
| QueryHits Generated | 10,000 | 100,000 | 1,000,000 |
| Downloads Recorded | 133 | 10,639 | 258,769 |
| Unique IP Addresses | 10 | 192 | 2,698 |

**Table 2.** Statistics collected for the experiment, in which our Web Server actually serves a file with 0 bytes. Denote that Duration is related to the time period needed by the malicious client to serve the amount of Queries specified in the 2nd row. The last two rows represent information collected for the whole experiment.

Gnutella Handshake requests in addition to HTTP GET requests. That is, our Web Server was considered as a good peer to connect to by other peers of the system, since it was advertising that it had a lot of content to provide.

It is important to note that the second graph in Figure 1 depicts the HTTP requests which were logged by our Web Server 10 days after the end of the experiment. We observe that peers that could not receive the content they were looking for kept on trying for many days. It seems that Gnutella has a kind of "memory", with the information contained in QueryHits having a long lifetime inside the system. It appears that a DoS attack based on generating malicious QueryHit packets is hard to stop, since the Gnutella system will continuously try to access the victim machine. This means that even if the original attacker is discovered and shut down, the attack may still go on.

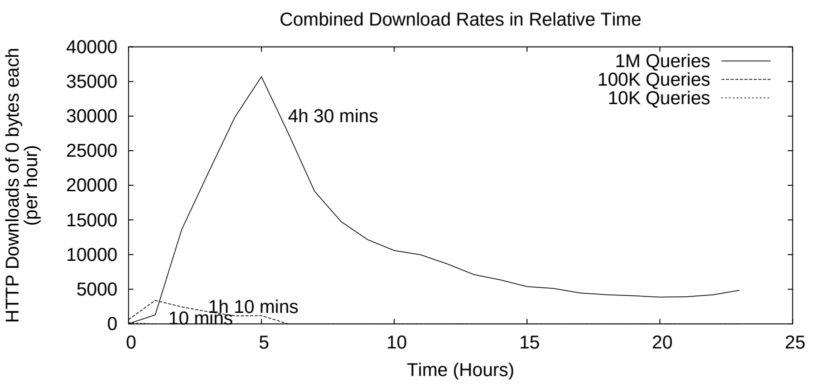

**Fig. 3.** Download rate (per hour) of a file of 0 bytes from the Web Server. Each curve has a caption noting the time needed for the malicious client to answer the amount of Queries.

## 4.2 Adding HTTP Exploitation

In the next experiment we want to experiment with a DoS attack to a Web Server using a single servent connected to the Gnutella system. We modified our client to create QueryHits that carried filenames constructed in the fashion we explained in Section 3.2. For each incoming Query we constructed a QueryHit with the filename:

```
../../high_quality HTTP/1.0\r\n\r\n search_criteria.mp3
```

Our Web Server had a file with filename “high_quality” with an actual size of 3,240,000 bytes in its Document Root directory. That is, every request performed by a Gnutella peer had as a result an actual file download of 3,240,000 bytes.

We performed the experiments for 10K Queries, 100K Queries and 1M Queries respectively. That is, our malicious servent was generating a fixed amount of QueryHit packets in each experiment. The attacked Web Server was instantiated on a new port before the beginning of each experiment. Note that our Web Server was isolated from all other traffic, since it was always listening to non-standard port numbers. The download requests per hour recorded by our Web server are presented in Figure 2.

We observe that in contrast with the previous experiment, the request rate per hour is quite low. This is obvious, since the Web Server is quickly saturated and thus unable to serve all incoming requests. That is, many requests are not recorded because they never manage to complete the TCP/IP handshake with the Web Server.

One could argue that the decrease of the request rate is due to our HTTP exploitation trick. However, we have found that is not the case. We repeated the experiment but instead of using a file of 3,240,000 bytes we used a file with the filename identical but empty (e.g., zero-size). The file size was again advertised

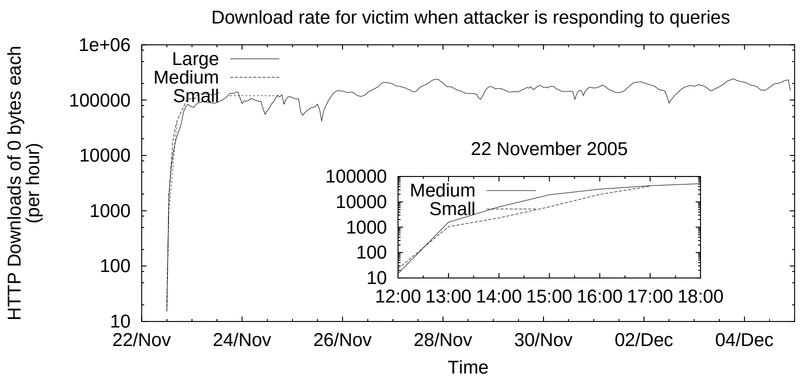

**Fig. 4.** Download rate per hour of a file of 0 bytes from the Web Servers during a simultaneous DoS attack, while each of the malicious client is connected to the Gnutella system. Y axis is on logarithmic scale.

as 3,240,000 bytes in our QueryHits. The results of this experiment are presented in Figure 3.

As we can see the request rate is quite high. This confirms the observation that our Web Server was under a DoS attack during the first experiment, since a lot of its incoming requests were never recorded in the log file.

In Tables 1 and 2 we present the results for both experiments. (These Tables actually contain the aggregate numbers used in Figures 2 and 3.) At first, we observe that in the case where the Web Server actually serves a file with a size of 3,240,000 bytes, more than 5,000 complete downloads have taken place in a few hours. This corresponds to more than 15 GB of data and represents the amount of data transmitted from our attacked Web Server during the experiment. Furthermore, we observe that in the second experiment where our Web Server responds with an empty file, we recorded downloads for more than one quarter of the amount of QueryHits produced by our malicious servent. Of course, this number does not represent unique requests, since the unique IPs which were logged were less than 3,000. On the other hand, because of the existence of NAT and Proxy gateway configurations, it is quite likely that less than 3,000 unique IPs map to a larger number of unique users.

It is interesting to observe that users seem to download files with obscured filenames. We believe that besides naive users that download everything, some automated clients must exist that pre-configured to download everything in batch mode. This suspicion is supported by our logfiles, which contained records of download entries with names like “foo.mpg.mp3”. That is, due to the naive way that our malicious servent respond to incoming Queries, it generated completely bogus QueryHits, and, surprisingly, some of them were actually selected for downloading.

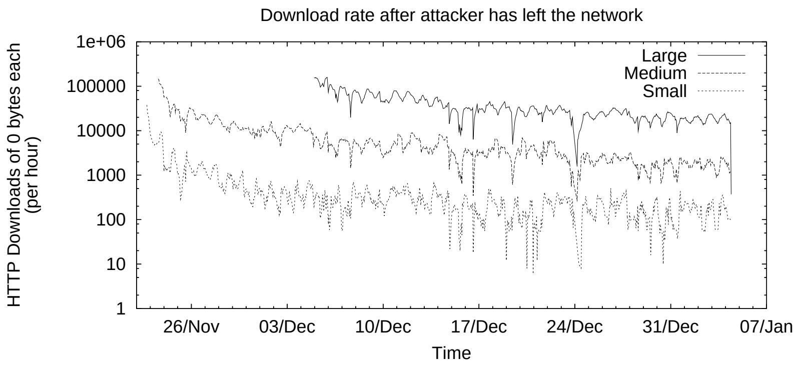

**Fig. 5.** Download rate per hour of a file of 0 bytes from the Web Servers during a simultaneous DoS attack, after the time each of the malicious client got disconnected from the Gnutella system. Y axis is on logarithmic scale.

| Duration | 303 mins | 1,529 mins | 18,273 mins |
|---|---|---|---|
| QueryHits Generated | 1,000,000 | 10,000.000 | 100,000,000 |
| Downloads Recorded | 731,625 | 10,161,472 | 70,952,339 |
| Unique IP Addresses | 5,272 | 52,473 | 421,217 |

**Table 3.** Statistics collected for the final experiment, in which we issued a simultaneous DoS attack in three distinct Web Servers. Denote that Duration is related to the time period needed by the malicious client to serve the amount of Queries specified in the 2nd row. The last two rows represent information collected for the whole experiment.

## 4.3 Measurements of a simultaneous DoS attack

Since we were convinced by the previous experiments that our Web Server was persistently under a DoS attack, we wanted to study the nature of the attack in more detail. We set up a new experiment with five malicious clients acting simultaneously against five distinct Web Servers. The malicious clients were configured to serve 10K, 100K, 1M, 10M and 100M Queries using HTTP-exploitation. Each of the clients was running in a dedicated machine. The targets were five distinct Apache processes running on a dedicated server, isolated from other incoming and outgoing traffic, except for Web and SSH. We decided to discard the traces from the first two clients (the ones that served 10K and 100K Queries) since the generated download rate was quite low compared to the rate produced by the other three clients. For the rest of this section we will refer to the traffic produced by the three malicious clients that served 1M, 10M and 100M Queries as Small, Medium and Large. Notice that the major difference between this experiment and the one in the previous section is that the malicious clients run simultaneously and all the DoS attacks are taking place at the same time.

Table 3 presents the results of our last experiment. In addition, in Figures 4 and 5 we present the request rate recorded by our Web Servers while the malicious clients were serving Queries and after the time they stopped. Observe that the attack does not stop at the time the malicious clients end their action, but continues for many days.

It is very interesting to observe the fluctuations of the curves for specific daily time periods. We believe that this effect relates to non-business hours and holiday periods in different locations, e.g., when users are more likely to be using their Gnutella clients or more likely to be engaged in other activities. For example, notice in Figure 5 that on December 24 and at the end of December (e.g., on Christmas eve and around New Year's day) the request rate is quite low.

## 5 Proposed Countermeasures

As demonstrated in the previous section, a DoS attack can be launched by malicious peers that answer all Queries received and hereby direct unsuspected Gnutella peers to request a non-existing file from a third party such as a Web server. Furthermore, by embedding specific file names in the QueryHit packet, malicious peers can force ordinary peers to download an existing file from a Web server. If a great amount of ordinary peers is tricked to download a large file from a Web server, the server will soon be unable to serve its ordinary requests, since its available capacity will be exhausted by the requests performed by the tricked Gnutella peers.

One could argue that the existing Gnutella software can be changed to detect HTTP-exploitable filenames in QueryHit packets, but this would not cover attacks against Gnutella peers. Another practical solution would be to prevent URL escaping in HTTP GET requests, in a fashion similar to Web Browsers. Network-level intrusion detection and prevention systems could also be used to filter Gnutella traffic from the traffic a Web Server receives.

However, we believe that it is worth examining whether it is possible to tackle the problem "head-on" rather than relying on partial fixes or workarounds such as the ones presented above. We next describe an algorithm that aims to detect and mitigate the impact of DoS attacks to non-Gnutella participants and present a preliminary evaluation.

### 5.1 Short Term Safe Listing: the SEALING algorithm

Our algorithm mainly focuses on protecting innocent victims such as non-Gnutella participants from DoS attacks originated from Gnutella. We consider a non-Gnutella participant as any host advertised to Gnutella (i.e. with an IP address and port number delivered in a Gnutella QueryHit) that does not support the

Gnutella protocol. That is, the following Validation Criterion is used to distinguish between third parties that are potential victims of a DoS attack from normal Gnutella peers:

**SEALING Validation Criterion:***Any host advertised in a Gnutella QueryHit packet which can not respond correctly to a Gnutella Handshake process is considered as a non-Gnutella participant and a potential victim for a Gnutella-based DoS attack.*

The SEALING algorithm is shown in Figure 6. The goal of the algorithm is to place potential DoS victims in a Safe List based on the SEALING Validation Criterion. This Safe List keeps track of machines that should not be contacted for downloads. Each Gnutella node keeps a Safe List and periodically updates its records. Each record has a lifetime of a fixed time interval. For the purposes of our evaluation, we used a fixed time interval of 30 minutes.

```
SafeListLifeTime := 30 mins;
if (GnutellaPacket(pkt) == QueryHitPacket) {
    GnutellaExtractNode(pkt, &GnutellaNode);
     if (SafeListContains(GnutellaNode)) {
        if (CurrentTime() -
            SafeListGetTimeOfNode(GnutellaNode) <
            SafeListLifeTime)
            GnutellaDropPacket(pkt);
    }
    else
          GnutellaParseHits(pkt);
}
...
onDownloadAttempt(node, file) {
    if (GnutellaHandShake(node))
        GnutellaDownload(node, file);
    else
        SafeListAdd(node);
}
...
```

**Fig. 6.** SEALING Algorithm.

### 5.2 SEALING Evaluation

We attempt to evaluate the SEALING algorithm using the trace collected from the Middle DoS attack. We group the download requests by the IP address

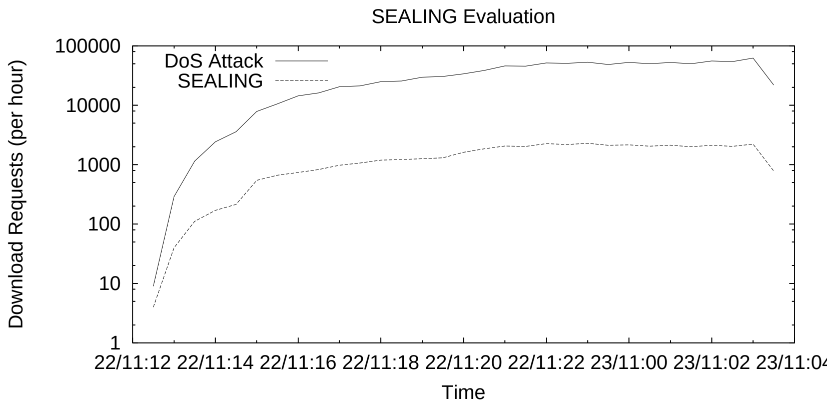

**Fig. 7.** The evaluation graph of the SEALING algorithm. The solid curve represents the amount of download requests during a DoS attack using Gnutella. The dashed curve represents the amount of download requests that will be eventually exposed to a Web Server, if Gnutella nodes utilize the SEALING algorithm.

recorded by the Web Server during the attack. We consider the first download request as a download attempt that, according to SEALING, will fail since the Web Server will not respond correctly to the Gnutella Handshake. Based on SEALING, all the download requests following the first download and for the next 30 minutes will be filtered out by the Gnutella peer and eventually will not make it to the Web Server. That is, we assume that the Gnutella peer that received the QueryHit, will add the Web Server to its Safe List after failing to Handshake with it.

For every download request we find in the trace we compare its timestamp with the first one encountered in the trace, which serves as the time offset of the SEALING algorithm. If the timestamp of a download request is found to be over 30 minutes after the time offset, then we consider that the download request serves as a new Handshake, which will also eventually fail. Again, we filter out the next download requests we encounter in the trace that have relative time difference less than 30 minutes with the new time offset. The results of the evaluation, as shown in Figure 7, indicate that SEALING reduces the effectiveness of the DoS by roughly two orders of magnitude in terms of the number of download requests to the victim site. We believe that this is sufficient to downgrade the threat of Gnutella-based DoS attacks to the level of mere nuisance for the majority of potential victims.

## 6 Related Work

There are many studies on security issues of unstructured P2P systems such as Gnutella. Daswani end Garcia-Molina[5] propose a number of strategies for limiting Query flooding attacks through Query traffic balancing an the Ultrapeer level. Mishra[6] describes extensively a number of existing attacks in P2P systems and proposes a new protocol called Cascade. One of the main features of Cascade is iterative search. In iterative search, a peer controls the Query flow. In contrast with pure flooding, iterative search forwards the Query to a peer's neighbors and requests the neighbors of each neighbor. Then, it proceeds on connecting to them and performing the Query recursively.

Zeinalipour-Yazti[10] considers the spam generation problem in Gnutella and proposes for each peer to perform a direct connection to the peer it wants to download from, using the system protocol and the download protocol, re-querying the peer and then performing the download.

AusCERT[1] has published an anonymous article which presents a traffic analysis from Gnutella traces. The analysis discusses IP addresses and Port numbers in PONG and QueryHit messages that are not Gnutella peers, implying that DoS attacks via Gnutella may have already been performed.

Paxson[7] has studied the problem of reflectors in DoS attacks, where Gnutella is also listed as a major threat. According to Paxson, a Gnutella network can be used as a reflector in a DoS attack by generating fake PUSH messages. A PUSH message is sent to a firewalled peer which can not accept incoming connections, so as to initiate the connection for a data transfer.

Finally, some proposed enhancements to Gnutella may further amplify the attack presented in this paper. For instance, in an attempt to address to the freeriders problem Sun and Garcia-Molina have proposed SLIC[9], a technique that rewards data share holders and isolates freeriders. Because it does so based on the number of Queries and QueryHits forwarded, SLIC is likely to be a prosperous environment for the attack presented in this paper, since it promotes peers that have seemingly great answering power.

## 7 Concluding Remarks

We have demonstrated how unstructured P2P systems can be misused for launching DoS attacks against third parties. We have developed an attack that exploits a number of weaknesses of unstructured P2P systems and manages to instruct innocent Gnutella peers to generate a significant amount of traffic to a victim host. The victim can be another Gnutella peer, but also a host outside the Gnutella system, such as a Web Server.

Although the basic attack relies primarily on the ability to spoof QueryHit responses, we also took advantage of the HTTP protocol used by Gnutella peers for data transfers. This allowed us to construct malicious QueryHits that result to a download of an arbitrary file from any Web Server. An interesting observation is that the use of HTTP in this case allowed the attack to "leak" to other systems as well.

Finally, we have developed SEALING, an algorithm which aims on keeping a local Safe List on each peer containing IP addresses and port numbers of hosts that have been characterized as non-Gnutella participants. Our algorithm assumes that any connection from Gnutella participants to non-Gnutella participants is a possible DoS attack.

### 7.1 Future work

To ensure prompt mitigation of Gnutella-based DoS attacks we believe it is necessary to further strengthen our defenses. The SEALING algorithm presented in this paper is sufficient, but only if it is adopted by a large fraction of Gnutella users as its effect is proportional to the fraction of nodes that support it. Until most nodes implement SEALING, it may be worth considering countermeasures that can be effective even if only deployed on a smaller fraction of nodes, such as superpeers. One solution that we are currently exploring is probabilistic validation of QueryHits on superpeers.

Another direction worth exploring is how the basic attack can be used against third parties other than Web servers and Gnutella peers, and for launching attacks other than DoS. For example, it may be possible to embed buffer-overflow URLs in QueryHit responses, so that Gnutella peers unintentionally assist in the dissemination of malware-carrying exploit code to victim servers. Determining the feasibility and effectiveness of such an attack requires further investigation and experimental analysis.

## 8 Aknowledgements

We sincerely thank Kostas G. Anagnostakis (Internet Security Lab, Institute for Infocomm Research, Singapore) for his valuable comments regarding the set up of the experiments and the overall understanding of the Gnutella based DoS attack's nature. We also thank, in alphabetical order, the following members of the Distributed Computing Systems Laboratory (ICS, FORTH) for their valuable remarks during a series of meetings regarding the material presented in this paper: Akritidis Periklis, Antonatos Spiros, Antoniades Demetres, Athanatos Manos, Koukis Demetres, Papadakis Charalambos, Polychronakis Michalis and Fragopoulou Vivi.

## References

1. Anonymously Launching a DDoS Attack via the Gnutella Network. http://www.auscert.org.au/render.html?it=2404.
2. Apache web server. http://www.apache.org/.
3. Debian gnu/linux os. http://www.debian.org/.
4. Gtk-gnutella servent. http://gtk-gnutella.sourceforge.net.
5. N. Daswani and H. Garcia-Molina. Query-flood dos attacks in gnutella networks. In *ACM Conference on Computer and Communications Security*, 2002.
6. M. Mishra. Cascade: an attack resistant peer-to-peer system. In *the 3rd New York Metro Area Networking Workshop*, 2003.

7. Vern Paxson. An analysis of using reflectors for distributed denial-of-service attacks. *SIGCOMM Comput. Commun. Rev.*, 31(3):38–47, 2001.
8. Daniel Stutzbach and Reza Rejaie. Characterizing the two-tier gnutella topology. *SIGMETRICS Perform. Eval. Rev.*, 33(1):402–403, 2005.
9. Qixiang Sun and Hector Garcia-Molina. Slic: A selfish link-based incentive mechanism for unstructured peer-to-peer networks. In *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 506–515, Washington, DC, USA, 2004. IEEE Computer Society.
10. D. Zeinalipour-Yazti. Exploiting the security weaknesses of the gnutella protocol. Technical Report CS260-2, Department of Computer Science, University of California, 2001.