Gnutella Peer-to-Peer Security (April 2007)

Gurpreet Dosanjh, Brody Lodmell, Alexander Van Der Star, Shane Wang

***Abstract*— The open source Gnutella protocol is an ubiquitous distributed file-sharing protocol implemented by many file-sharing clients. After various vulnerabilities were discovered with the previous 0.4 version of the protocol that opened the network up to distributed denial of service attacks and malicious file injections, the Gnutella team has managed to patch these vulnerabilities in the protocol's latest 0.6 rendition. However, after extensive research detailed in this report, vulnerabilities with the protocol still exist and are easily implemented through any of the open source Gnutella clients. One particular vulnerability opens the Gnutella network up to a flood of un-regulated files which could easily be viruses or Trojans posing as searched files of other end users. Our recommendation involves a voting system be incorporated in the next protocol version to identify malicious files or users and remove them from the network.**

***Index Terms*— Distributed Computing, Computer Security, Computer Network Security**

I. Introduction

Gnutella is an open source peer-to-peer protocol that is the basis for many popular file sharing applications today, including BearShare, LimeWire, and Phex. During our experiments we found the Gnutella network had 600 000 users connected just to our subnet. Since any vulnerability in the Gnutella protocol would make all machines on the network vulnerable to an attack, it is important that Gnutella be designed with security in mind.

Gnutella's current official protocol is the 0.4 protocol, which was last modified in 2003 [3]. There have been at least two security vulnerabilities in this protocol that were identified back in 2002, and can be used for a variety of malicious purposes. However, there is a newer version of the protocol, version 0.6, which is currently in testing [4]. Version 0.6 enhances the security of the Gnutella protocol, and makes some of the version 0.4 exploits obsolete. While this protocol is not the official protocol yet (It will be in September), the majority of major applications that utilize the Gnutella protocol have already switched to use 0.6.

This paper aims to educate the reader about the exploits that already exist in the official protocol version, and then talk about the changes that version 0.6 will bring. From there, we will outline our attempts to find and exploit a vulnerability in the 0.6 version of the Gnutella protocol, and discuss any improvements that could be made to prevent these exploits from occurring.

II. Gnutella 0.4 Protocol

The current Gnutella protocol, version 0.4, operates very differently than the newer 0.4 protocol. The 0.4 protocol has no handshaking, and operates as a decentralized network of peers who communicate directly with each other (See Figure 1).

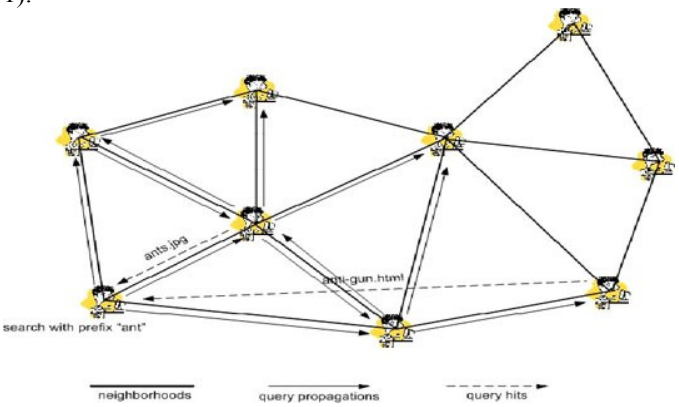

Fig. 1 – The network topography of the 0.4 protocol for Gnutella [2]

Peers on the Gnutella network are normally called 'servents' as they act partly as a server and partly as a client. When a servent first connects to a network, it will send a PING message to any immediate neighbour, who will send the PING along to all of its neighbours. This process is repeated up to 7 times. Any peer who receives the PING will respond with a PONG message to let the newly connected servent know that it is responding. From there, the servent can send out QUERY messages, which are basically search strings for files that the servent is trying to locate, in the same manner, and it will receive QUERYHIT messages from peers that have a file matching the query. After a servent receives a QUERYHIT, that servent can immediately start a TCP connection on the specified port for the file with the peer that said it had the file.

While this style of network layout is very useful for exchanging information in a purely P2P manner, it has not been designed with security in mind. Firstly, there are no controls on flow of traffic, or verification of identity. The lack of both of these makes the network susceptible to being used as an agent for a Denial-of-Service attack. Also, since

there is no manner of verification that the correct file is being sent once a QUERYHIT is received, there is a possibility that a client might be sent malicious files when behaving correctly. All the previous work we found on Gnutella security in 0.4 dealt with these two vulnerabilities.

III. Past Work

Only one previous published attempt to analyze Gnutella's network security was found in our research. It was a paper released in 2002, which highlighted a couple weaknesses in the 0.4 protocol. There were two attacks outlined:

DDoS attack using modified QUERYHITs.

Virus Injection using push.

The DDoS attack in this paper was verified experimentally, while the virus injection attack was only described theoretically [1]. We used these two attacks as the basis for our analysis of the 0.6 protocol of Gnutella, so it is important to understand how these attacks work.

IV. DDoS

The DDoS attack outlined in [1] uses the QUERY/QUERYHIT exchange in order to perform a denial of service attack on a victim machine. This is done by replying to any query that it receives with a modified QUERYHIT. This modified QUERYHIT message is the same as a normal QUERYHIT, with one exception: the 'source' field of the QUERYHIT is set to be the IP of the victim machine. Thus, whenever any machine queries for anything that reaches the malicious peer, an HTTP connection with the victim machine will be started. Due to the query-flooding nature of the 0.4 protocol, this means that the growth in the number of connections that will be started will be exponential in nature, and will quickly become enough to take the victim machine offline. In the 2002 paper, the experimental DDoS attack caused the victim machine to receive more than 80 requests per second, at which point the victim machine froze and became unresponsive [1].

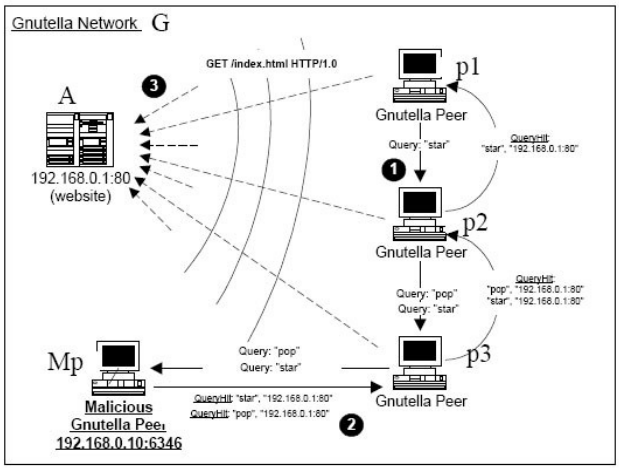

Fig. 2 – A Gnutella 0.4 DDoS attack [1]

V. Virus Injection

The virus injection attack outlined in the paper was theoretically very simple, but exploited a vulnerability present in the 0.4 protocol. This vulnerability was that whenever a servent sent a 'push' message to another machine, it would automatically accept whatever file was being sent without any sort of verification. This meant that if on receiving a push, the malicious servent simply sent a different file, such as a virus or a Trojan, to the victim who sent the push, it would be accepted unquestionably and the machine would be compromised.

VI. Gnutella 0.6 Changes

Gnutella 0.6 introduced many changes to the fundamental workings of the network. The first notable change is the addition of an Ultra-peer [5]; moving the network away from a strictly peer to peer topography and closer to a traditional client server model. Clients (which are now called leaf nodes), connect to one or more Ultra-peers, and make requests through them. Upon receiving a request, an Ultra-peer will send requests to the other leaf nodes it's connected to on behalf of that leaf node. The Ultra-peer collects responses, and forwards them back to the original leaf node. This completely removes the query flooding nature of the network.

A handshaking mechanism was also added to the protocol [5]. The details are not relevant for our treatment but what is important is that any communication between leaf nodes and Ultra-peers, or leaf nodes and other leaf nodes must be preceded by a handshake. Because messages are now sent in direct TCP/IP connections, comparing the IP of the sender in the transmission layer and application layer is possible, and required by the network specification [5]. This removes the ability to spoof a message, or appear to be another location on the network. Handshaking eliminates the DDOS techniques which worked on the 0.4 version of the network.

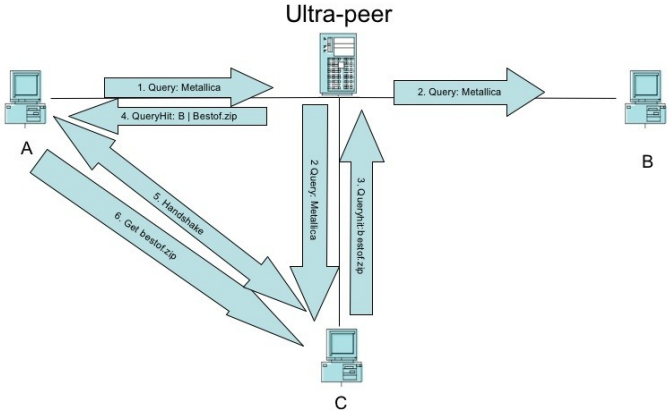

Fig. 3 - Querying a Gnutella 0.6 network

Refer to figure 3 for the following example. Suppose that a leaf node p wishes to search the network. P would send a

QUERY to the Ultra-peers it is connected to (1). Each Ultra-peer would then QUERY leaf nodes (2), and collect the QUERYHIT messages (3) forwarding them to 'p' as they came in (4). 'p' would then handshake with the leaf nodes who supplied QUERYHIT messages (5), and then download the file (6).

At first glance, it would appear possible to replicate the QUERYHIT attack of the 0.4 network if a malicious user acted as an Ultra-peer. This is not so, because of the handshaking undertaken. On a 0.4 network, servents, upon receiving a QUERYHIT, may start a HTTP connection on the listed source whereas in the 0.6 protocol, the leaf node will send a handshake. This handshake will be disregarded by a target which is not concerned with the Gnutella network. The only thing the malicious Ultra-peer will succeed in is initiating a handshake with leaf nodes on the Gnutella network with his/her attack target. The amount of these handshake messages the Ultra-peer can send, is limited by it's own bandwidth, and entails a greater overhead than that which will be placed on the attack target; rendering this attack worthless.

VII. Malicious File Injection Objectives

The Gnutella peer-to-peer network protocol is subject to many security vulnerabilities. In this part of the project, our objective is to implement a dynamic malicious file injection attack. Due to the weak design of the Gnutella protocol, significantly large number of exploits may be discovered in the massive interconnected network. Each Gnutella servent consists of a search or message distribution engine and a mini HTTP 1.1 web server. The Gnutella search engine uses several descriptors for sending and receiving messages: Ping, Pong, Query, QueryHit and Push. Ping messages are sent by a servent to explore active hosts on the network. Pong messages are sent by active hosts in response to Ping requests each time the host wishes to communicate with the servent. Each search item is called a Query. When an active host contains the file that is requested by the client, a QueryHit message is sent. If a host is protected by a firewall, the host Gnutella client uses the Push command to push the file back to the peer.

Recent publications in the area of Gnutella protocol security provide the design flaws of how a malicious file injection attack works, but no implementation of a dynamic file injection attack could be found. Older design models of such an attack consisted of creating numerous malicious files on several hosts. When a client requests any of these files from such hosts, the malicious files with the corresponding name is sent.

VIII. Implementation Details

Our implementation consists of a dynamic malicious file injection attack. Several Gnutella protocol compliant clients are freely available on the Internet. Due to the high complexity and limited documentation of JTella, LimeWire, and Gnucleus, Phex 3.0 proved to be the easiest client understand and thus was used for this prototype. Phex 3.0 is a P2P file sharing client that is developed using the Java language. The code base for Phex 3.0 is relatively small in comparison to the more popular clients such as LimeWire. Furthermore, the Phex 3.0 Gnutella client is relatively easy to edit and compile using the Eclipse IDE.

Two different methods of creating malicious file content were tried. When a user sends a query through the Gnutella network, there are two ways the file could be downloaded. A file can be downloaded either from a single host or it could download different segments of the file from multiple hosts. When a QueryHit message is returned by a host, a SHA1 hash value of the file being requested is also returned. Once the file has completely downloaded onto the client's machine, the Gnutella client computes the SHA1 hash value of the file to check if it equals to the SHA1 has value sent from the host. If they are equal, then the client considers the file to be valid, else it considers it to be corrupt. When a file is split into different segments where each segment is downloaded from different hosts, the client computes a SHA1 hash value of each segment and another SHA1 value of the entire file. As each segment is downloaded, it checks if that segment is valid by computing the SHA1 hash value and comparing it to the expected value.

Our first approach for implementing the attack involved sending dynamic malicious file segments to peers. The Gnutella client obtains a list of hosts that have the identical file. Because multiple hosts contain the same SHA1 hash value for each file, the client checks for the most popular SHA1 hash values for each segment from the different hosts. When a query requesting a particular file segment was received by our malicious host, a new malicious file segment was created dynamically with the same name as the query request and sent to the servent. The receiving servent obtained the segment and computed its SHA1 hash. Since this hash value was different from the expected value, the client identified the segment as malicious and blocked the download. This implementation of file verification makes it almost impossible to send false segments without the receiver discarding the data.

The second approach involved sending a complete file from a single host. When a query is received by a modified host (the one implemented by us), a malicious file is dynamically generated by copying another file and renaming it to the query text. Then the vector list of the modified host is updated to contain the new file. The modified host developed, dynamically creates a new file for each query received and then renamed the new file to the query text and responds with a QueryHit message. The client now has the option to download the newly created malicious file.

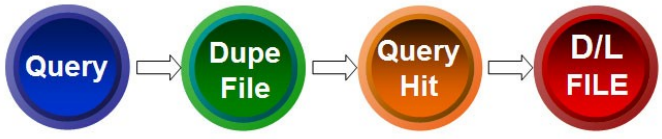

Fig. 4 – Malicious File Injection Process

IX. IP Logging and Reverse DNS Lookup

Another feature implemented in our modified Phex client was IP logging and reverse DNS lookup of all clients which sent incoming search queries. During testing, approximately 1,200 unique IP addresses were recorded within a period of 1 hour. This logging coupled with the malicious file distribution could aid a malicious user in identifying victim computers of his/her Trojan distribution along with the network in which they belong to. Test cases done at the UBC lab revealed the DNS lookup returned not only the ece.ubc.ca domain but also the computer name which the query originated.

X. Testing

The exploit just described was executed on both a private and a public network. The private network required one computer system to run the modified host, while the other computer system executed the unmodified client. On the other hand, in the public network, only one computer system running the modified host is needed and a Gnutella webcache is required. Once the modified host is connected to a webcache, it proceeds by downloading a list of other Gnutella servents from the public domain to which it will attempt to connect to. Once connections have been made to one or more peers, the modified host then eavesdrops on incoming queries, begins to create dynamic malicious files and starts to respond with QueryHit messages. When the modified host was executed on a live public Gnutella network, we recorded approximately 10, 000 queries within a seven hour execution time frame. Of these queries, 879 were .mp3 files, 689 were .exe, and 489 were .zip. The totoal number of downloaded files from the modified host were 78 downloads. This shows that less than 1% of the malicious file content was downloaded on a live network.

Further analysis of the test case revealed many of the queries were general search terms. Thus, since no query filtering had been applied to the modified client, these terms were automatically created as files. Modification of the client to filter for and duplicate only particular file types such as ".mp3" should provide a much higher download rate since the duplicated file would seem more authentic. Further query analyzing algorithms could be implemented to search for specific keywords which would determine not only the type of file the user is looking for but also possible file sizes. Doing so, the client could utilize more than one base malicious file and select the appropriate file based on the query string. Such an example would be a base file of Song.mp3 (4mb size) and Install.exe (20mb size). A query that filters to a mp3 file type can use the base Song.mp3 while a query that filters to an application file can use the Install.exe base file.

XI. Recommendations

The current Gnutella protocol infrastructure and protocol is unable to handle and distinguish malicious files from authentic ones. One possible implementation would be a file authentication rating level based on voting by shared users. Files that receive over a set threshold of negative ratings can have their hash values blacklisted. These blacklists can be stored on the Gnutella webcaches and updated and retrieved regularly. Threshold values for blacklisting will have to be adjusted as lower threshold values run the risk of having authentic files blacklisted by malicious voters and malicious files authenticated by the same malicious voters. A second possible solution is to take the same voting scheme and apply it at the servant level. This way, malicious users can be singled out according to their IP address before they can pollute the Gnutella network with too many malicious files.

XII. Conclusion

The Gnutella network is a very popular distributed file sharing network that is increasing in popularity. With the current number of users, security features of the protocol become paramount. Although the newer 0.6 version of the Gnutella protocol has made significant improvements over the older and vulnerable 0.4 implementation, much work is still needed to address the security weaknesses that are still present. The work presented in this research demonstrates the alterations to a popular Gnutella client to turn it into a mass distribution mechanism for malicious files.

References

[1] D. Zeinalipour-Yazti, "Exploiting the Security Weaknesses of the Gnutella Protocol", March 2002, Available: http://www.cs.ucr.edu/~csyiazti/courses/cs260-2/project/gnutella.pdf

[2] K. Shen, "Computer Networks", 2006, Available: http://www.cs.rochester.edu/~kshen/csc257-fall2006/assignments/gnutella.jpg

[3] The Gnutella Developer Forum, "The Annotated Gnutella Protocol Specification v0.4", Rev. 1.6, Available: http://rfc-gnutella.sourceforge.net/developer/stable/index.html

[4] P. Kirk, "Gnutella 0.6 – Defining a Standard", July 2003, Available: http://rfc-gnutella.sourceforge.net/developer/index.html

[5] The Gnutella Developer Forum, "RFC – Gnutella 0.6", Available: http://rfc-gnutella.sourceforge.net/developer/testing/index.html