Searching for Malware in BitTorrent*

Andrew D. Berns and Eunjin (EJ) Jung

April 24, 2008

Abstract

One of the most widely publicized aspects of computer security has been the presence and propagation of malware. Malware has adapted to many different changing technologies, including recently-popular P2P systems. While previous work has examined P2P networks and protocols like KaZaA and Gnutella for malware, little has been done so far that examines BitTorrent. This project explored BitTorrent for the presence of malware, and discovered a significant portion of malware in the downloaded file set. Statistics on torrents infected with malware were gathered and analyzed to find patterns that are helpful in creating basic filtering heuristics. While these heuristics may work in simple cases, several easy ways they can be defeated were found.

1 Introduction

Recently, peer-to-peer networks have emerged as a popular paradigm for Internet applications. In fact, a study in 2005 estimated that P2P traffic accounted for around 70% of all traffic on the Internet [2]. P2P technology is finding new applications as it grows, including voice-over-IP systems and streaming video delivery. While P2P has found several different uses, perhaps the most widely-known use is for file sharing.

One concern with P2P file sharing is that it can be used to distribute malware (malicious software, such as worms, viruses, and rootkits). On the one hand, users have access to huge amounts of data, while on the other hand, this data can easily be tainted with viruses, worms, and other forms of malware. An important consideration, then, is if the concern about malware in P2P networks is warranted, and if it is, are there ways to protect a computer and minimize the risk of malware infection from P2P networks.

This project was an introductory glance into these two concerns with the BitTorrent protocol. To address these concerns, I collected data over a nine day period, and examined it for malware. Also, I looked at the characteristics of the discovered malware, searching for patterns that could be used for creating a simple filtering system. Finally, the project explored a few simple ways the BitTorrent protocol could be exploited to defeat these rules. With the large amount of BitTorrent traffic today, controlling the spread of malware with the BitTorrent protocol is an important issue.

2 Background and Previous Work

2.1 BitTorrent Overview

BitTorrent is a P2P protocol for sharing data developed by Bram Cohen in 2002 [12]. BitTorrent offers download speeds that are usually faster than what can be achieved through many other P2P

*University of Iowa Computer Science Technical Report UICS-08-05

protocols such as Gnutella. The reason BitTorrent can offer fast download speeds is because it allows users that are downloading a file to also share the pieces they have downloaded with other users. Furthermore, it employs a tit-for-tat strategy, making sure those that are downloading are also sharing their already-received pieces. This results in many more users sharing a file, and helps control the impact of free riders.

To download a file with BitTorrent, a user first locates a metainfo file (this file has extension `.torrent`, and is commonly called a torrent). This file is often located by using a BitTorrent index, which keeps links to various metainfo files. The metainfo file contains the address of the server that is storing the information on the peers currently sharing the file. This server is called a tracker. A user connects to the tracker found in the metainfo file and requests a list of peers to connect to. The tracker responds back with a peer list and some statistics such as the number of total users sharing the file. Using the reply from the tracker, the client connects to the peers and downloads pieces of the files from them, as well as uploading completed pieces to other peers. This group of connected peers is called a swarm. When a file is being downloaded, this peer is considered a leecher. Once the file is completely downloaded and shared, the peer is considered a seed.

Estimates place BitTorrent as being the second-most popular P2P system, close behind eDonkey [2]. There are dozens of different BitTorrent clients and servers available, both free and for a price (as an interesting note, some BitTorrent clients actually install malware with the client, although the examination of these is beyond the scope of this project). Index sites boast over a million active torrents, and thousands of new torrents are added each day. It seems like a natural progression that this popularity would be accompanied by more malware both targeted by and distributed with BitTorrent.

2.2 Malware Background and Previous Work

While the popularity of peer-to-peer networks like BitTorrent is a relatively new occurrence, the presence of malware on personal computers is not. Malware has a long history, as does methods of distributing malware. Distribution has gone from requiring users to share an infected file between computers by disk, to now being able to infect a computer simply by a user viewing an image or visiting a webpage. This ability to replicate has been a huge advantage for malware propagation. Malware that automatically propagates to other computers has caused major problems, and new software errors for compromising systems are appearing at a fast pace.

As P2P applications have become more popular, they are being used more and more for malware distribution. P2P file sharing offers malware a natural system to spread – malware can simply disguise itself as a file that users want, and then users will download, run, and possibly even share, the downloaded file on their own. The malware creator does not need to bother with having the malware execute or propagate automatically, as this is handled by P2P file sharing. Research has shown that not only is malware present in some P2P file sharing networks, but that it is also relatively common [10]. Even though this type of malware in P2P networks does not create copies and distribute itself on its own, it is still a prevalent malware threat [9].

There are also some examples of malware that attempts to self-replicate in P2P systems, mainly by automatically creating copies inside the P2P application's shared folder. For instance, the Achar virus in KaZaA creates copies of itself inside a user's shared folder, taking on several different names in the hopes that it will be downloaded by other users [16]. Using an even more general method, the Mandragore worm in the Gnutella network works by intercepting all queries that reach an infected computer, and replying with the virus renamed to match the query [17].

Interestingly enough, I have only been able to find one example of malware that has exploited BitTorrent for automatic propagation. The Impard-A virus is usually spread by opening an image

file sent by an infected instant messaging client [20]. However, when it infects a computer, it also searches for the file bittorrent.exe, and, if found, will seed itself automatically using the bittorrent.exe client. It seems that this approach is quite simplistic, however, as it will only work if the mainline BitTorrent client is installed (named bittorrent.exe - which is not the most common client [11]), and does not appear to hijack other torrents with different names. This could be because the worm's main exploit focuses on instant messaging clients, and the BitTorrent infection could have been an afterthought. It is an interesting question why clients using the BitTorrent protocol have not yet been used for transmitting malware automatically.

The previous work on malware in P2P systems does not focus on the BitTorrent protocol, working instead with networks like KaZaA and Limewire [10,21]. This previous work, however, is quite extensive, both in the total data captured, and in the depth of their particular focused analysis. This project operates on a much smaller scale than previous P2P malware papers, in terms of both total data captured and depth of focus, but it examines the BitTorrent protocol for malware, which has not yet been extensively considered.

3 Implementation and Data Collection

Obviously searching for malware requires a bit of caution to prevent infection of the project's computer. For my project, I did all file downloading and scanning on a VMWare Server [18] virtual machine running the Ubuntu Linux distribution [15] as a guest operating system. My rationale for choosing Linux was that Linux has avoided being the target for most malware, and therefore using it added an extra layer of protection on top of the virtual machine. The Ubuntu distribution was chosen simply because it was on hand at the time of installation. The host system for VMWare Server was an Athlon XP 2500+ PC with 1 GB of RAM running Windows Server 2003 [19]. All downloading was done on a residential cable modem connection with a speed of 1.5 Mbps.

For my project, I had three different stages for capturing data concerning an illegitimate torrent: finding torrents, downloading and scanning the files, and finally capturing periodic data from the tracker. The original intent was to automate each stage so that as many torrents as possible could be examined. However, due to the extra time spent automating the periodic capture of torrent statistics, as well as the the need to download torrent files from indexes that made simple HTTP crawlers ineffective, I was unable to automate the selection of torrents. Once the data collection was underway, it was suggested to use RSS feeds for finding torrents, which would greatly simplify the problem of acquiring the torrent files. However, this consideration came too late to be implemented in this project.

Because torrents could not be selected automatically, I followed a simple procedure for manual torrent selection. I used a rotation of four different BitTorrent index sites - BushTorrent [4], isoHunt [7], Mininova [11], and BTJunkie [3]. These four sites were chosen after reading positive reviews of them in an article comparing BitTorrent indexes [13]. Every morning and evening, I selected a torrent index site, and viewed all torrents filed under the "Application" category (if there were multiple subcategories, I simply selected the largest of them). From this listing, I would select the torrents for downloads that were at most ten megabytes and that had been added in the past day. These metainfo files were saved in the `torrents_new` directory on the virtual machine.

The next step after finding the torrent metainfo file was to download the actual file(s) and check for the presence of any malware. Every five minutes, all metainfo files in the `torrents_new` directory were automatically passed to the Enhanced CTorrent program for downloading [6]. The Enhanced CTorrent client was chosen because it offers simple command line support and an easy way to execute scripts upon download completion. I created a script to run after completion that

used ClamAV [5], a GPL anti-virus software package, to scan the downloaded file for malware. ClamAV was chosen both because it was open source, and also because it was easy to install in Ubuntu. If malware was found in the downloaded file, the script transferred the torrent into the `torrents_done` directory, and deleted the downloaded file. If no malware was found, both the downloaded file and the torrent were deleted.

The final step for the project's data collection was to periodically retrieve and save information about malware from the file's tracker. To do this, I modified torrentsniff, a Perl utility originally meant to gather some limited information about a torrent, such as the number of seeds and leeches, and the actual file size [14]. To fit my purposes, I added a section of code that gets all tracker URLs from the metainfo file, and sends each one requests for peers, using randomly created peer IDs. The program stops requesting peers from a tracker if either the tracker has returned all the peers it has, or if it has already made 3 requests. After every request, a stop message is also sent to prevent the tracker from saving the information on the fake peer ID. This utility was run automatically every 30 minutes, using every metainfo file in the `torrents_done` directory as input, and the output was saved in a file on the virtual machine.

4 Results

Overall, I downloaded a total of 379 files for this project over the course of 9 days, from April 11, 2009 to April 20, 2009. Collection of statistics from the trackers continued for another 3 days. Index sites were searched twice a day, once in the morning and once in the evening. As stated previously, four different index sites were used in rotation, and downloads were for files of size at most ten megabytes.

When analyzing the sample, 75 files in 70 downloads were found by ClamAV to be infected, meaning 18.5% of all downloads contained malware. Most malware appeared more than once. Table 1 below shows the count of malware amongst all infected files.

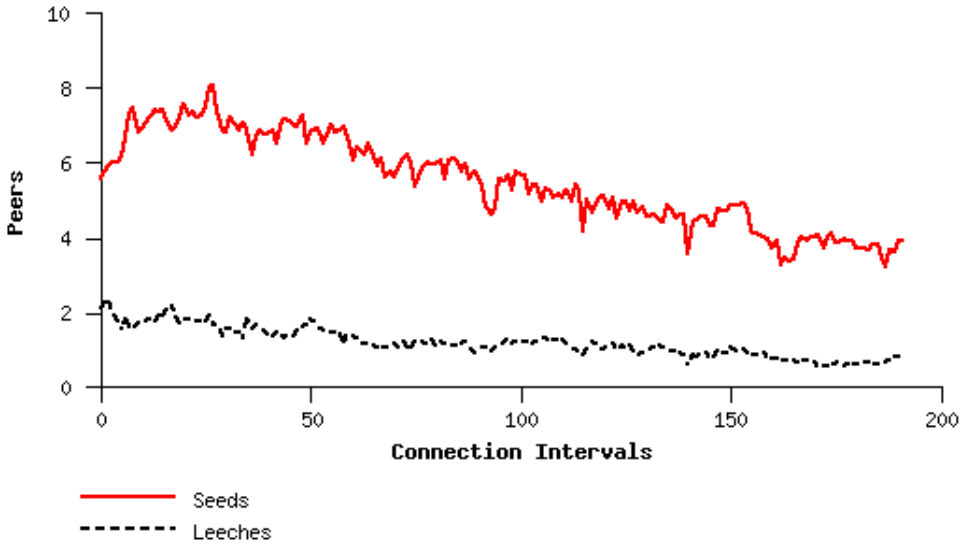
Table 1: Malware Occurrences in Sample

Malware Name	Number of Infected Files	Percent of Malware (Rounded)
Trojan.Small-5335	22	29.3%
Trojan.Zlob-3743	8	10.7%
Trojan.Dropper-3074	5	6.7%
Trojan.Agent-19483	5	6.7%
W32.Parite.B	4	5.3%
Trojan.Vundo-2185	3	4%
Trojan.Agent-11765	3	4%
Trojan.Spy-4973	3	4%
Trojan.Zlob-2789	2	2.7%
Trojan.Downloader-25772	2	2.7%
Trojan.Vundo-2505	2	2.7%
Others (only 1 occurrence found)	16	21.3%
Total	75	100%

There were a few common types of programs that contained malware. Fifteen of the infected downloads claimed to be a key generation or activation utility. Six were files claiming to be popular P2P file-sharing applications. Five files claimed to be utilities meant for copying and burning CDs and DVDs. The rest of the files were a mix between small utilities and plugins.

An interesting measurement is the average life cycle of a torrent that contains malware. This was calculated every half hour as the average number of seeds and leeches from all trackers tracking a particular torrent. The first four days (192 data collection intervals) of a torrent's life were plotted to look for patterns. Figure 1 shows a graph of the average number of seeds and leeches for malware torrents. Along with the infected torrents, fourteen torrents without malware ("clean" torrents) were analyzed. Figure 2 represents the average seeds and leeches for the fourteen clean torrents analyzed.

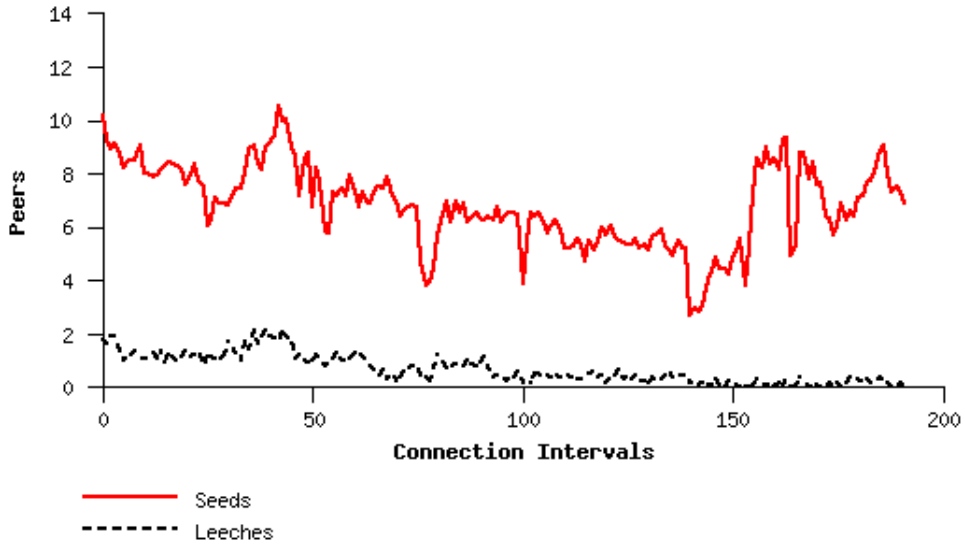
Figure 1: First Four Days of an Infected Torrent



The maximum number of seeds and leeches for analyzed torrents was captured as well. For the clean files, the maximum seed count of any torrent was 55, while the maximum leech count was 17. Torrents for malware had a much greater range, although they also had a much larger sample size. The maximum seed count for infected torrents was 109, while the maximum leech count was 36.

Finally, using the list of peers returned from the tracker, the distributions of time a peer was connected to a tracker (the sum of both leech and seed times) was calculated. Connection times were measured in thirty minute units to align with the data collection interval. The average connection time for infected torrents was five hours and twenty-five minutes, while the average connection time for clean torrents was nine hours and twenty-five minutes. However, over 90% of infected torrents had a connection time of less than 25 collection intervals (12.5 hours), and 90% of clean torrents had a connection time of less than 45 collection intervals (22.5 hours). The maximum connection time for infected torrents was 302 hours and thirty minutes, while the maximum connection time for clean torrents was 353 hours and thirty minutes. Figure 3 shows a graph of the distribution of connection times for the malware files, while figure 4 shows the distribution for clean files, both limited to the first 50 connection periods (this captures the vast majority of connection times without making the figure's scale unreadable).

Figure 2: First Four Days of a Clean Torrent



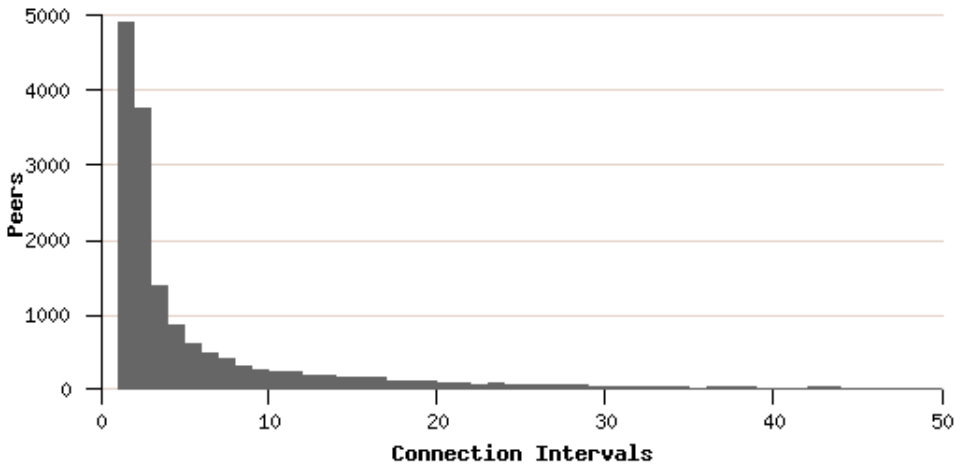
5 Discussion

Obviously, malware is present in many files available through BitTorrent trackers. Almost one-fifth of all downloads in the sample contained malware. However, while this is a significant portion, it is still significantly less than the proportion of malware found in some previous work examining KaZaA and Limewire [10,21]. While these previous studies had a much larger scale than this project, this result offers the suggestion that in fact BitTorrent is a "safer" protocol than previous P2P systems.

While this study did not run any tests to determine reasons why BitTorrent may contain less malware, several possible reasons can be derived from the details of the protocol. First, unlike many other current P2P protocols, searching does not ask individual computers inside the network for a file, but rather locates metainfo files, usually from large index sites. Because of this centralized search mechanism, an infected computer cannot respond as if it had the file being searched for in every query, but rather must generate multiple metainfo files and attempt to publish them in a popular place. Another reason may be that, because torrent files are indexed, torrents containing malware are quickly removed from the main index sites, preventing widespread dissemination. Unlike many other P2P networks, it is very easy to read comments and gather feedback from the index site about the files before downloading them. Index sites may remove malware before even being advertised, or quickly taken down before too many people download it.

Another interesting finding was the difference between torrents containing malware and other uninfected torrents. A previous work examined the life of a very popular torrent file, and found that the torrent experienced an initial period of fast-growing popularity, and also had more leeches than seeds [8]. This differs significantly with the results found for the malware files. For the malware examined, there was no large initial surge (no "flash-crowd"), nor were there more leeches than seeds. Also, the malware in our sample had significantly fewer seeds and leeches than the popular file examined in the previous study. This may suggest that malware torrents are much less popular than most legitimate files, although this project's examination of a few new clean torrent files suggests that low popularity is not unique only to torrents with malware. It does suggest,

Figure 3: Distribution of Total Connection Time for Infected Torrents



however, that further in-depth investigation with a larger sample size would be valuable.

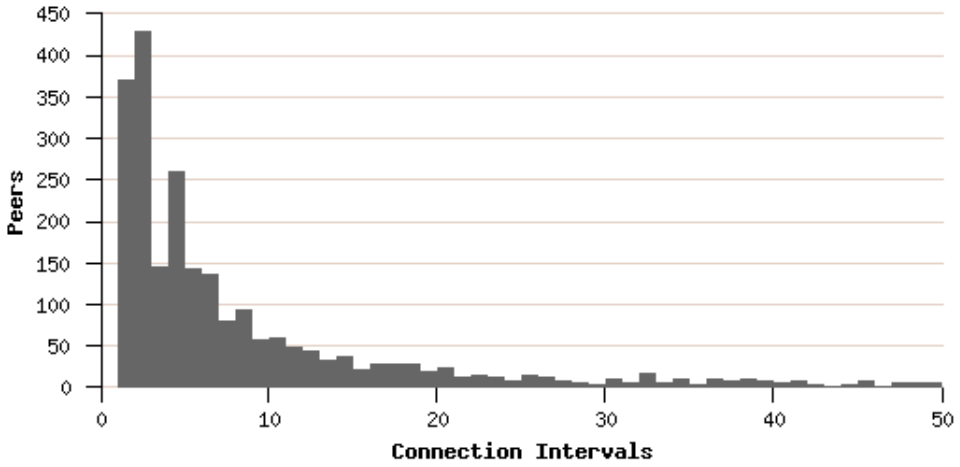
This knowledge of the differences between malware and clean files can be used to create simple heuristics to filter out malicious software. The most obvious filtering rule would be to not download torrents that have a low number of seeds (for instance, less than 110). Another method, which utilizes the results of the previous work on a popular torrent, would be to filter out all torrent files that had no flash-crowd (if history is kept), or new torrents with more seeds than leeches. As mentioned previously, however, there are clean files that would fit these patterns. If rules are based on the average information gathered here, one can expect both false positives on unpopular clean files, as well as false negatives on malware that greatly exceeds the average. A simple way to control these would be to use a segregated machine to download those torrents that are filtered from this pattern, and only after they are found to be clean allow them to be accessed by the rest of the computers on the network.

As with almost all computer security issues, user awareness also plays a large part in controlling malware in BitTorrent. Because the average connection time for a user downloading a file containing malware was lower than the connection time for a user downloading a clean file, there is a very good chance that many users realized the file was not what they intended and abandoned the file. Also, simple common sense principles when selecting a torrent to download are helpful. For instance, many of the torrents were new key generators with a low seed count. An informed user may wish to search for a different key generation utility that has been around longer and has more popularity. Also, several malware files claimed to be slightly more recent versions of files that were already available through other torrents. A savvy user may wish to settle for the popular version 3.7.14, instead of risking infection by downloading an unpopular version 3.7.15. A little bit of user precaution would appear to go a long way in preventing the downloading of malware in BitTorrent.

5.1 Exploitation of BitTorrent Statistics

After thinking of the basic rule of filtering out all torrents with low seed and leech counts, I wondered if these statistics could easily be faked, which could be a quick way around simple filtering rules based on file popularity. I tested this using two different methods, both with the main BitTorrent tracker program, bttrack [1]. In the first method, I assumed that the malicious user controlled

Figure 4: Distribution of Total Connection Time for Clean Torrents



the tracker for their malicious file. If this is the case, altering the peer count is quite simple. By changing one line of code in the tracker program, the torrent file can appear as popular as the malicious user wishes. Even if the user does not wish to modify the source code, bttrack uses a file to store its list of peers - all the malicious users needs to do is modify this file to show more peers than there actually are, and bttrack will handle the rest. Simple filtering rules based on popularity, then, cannot prevent against this easy modification.

The second method assumed that the tracker was not under the control of the malicious user, yet the malicious user still wishes to make the torrent appear much more popular. To inflate the seed count in this case, an attacker can send multiple announce messages to the tracker, creating the appearance of many different peers downloading the file. Using a short and simple program, I was able to easily raise the peer count on a test tracker I set up to over 300 in less than a minute, limited only by the speed HTTP messages could be sent to the tracker.

These two methods are very simple ways to defeat basic filtering rules. While there seems to be some easy ways to detect when these methods are occurring, the way the BitTorrent protocol is designed may very well prevent simple rules based on popularity from ever becoming very accurate against any mildly capable attacker, no matter how advanced the rules and detection. Because most trackers just store and display the information they receive from the users, any filtering method based on the current data most trackers contain is susceptible to attack. Perhaps a better approach would be to instrument trackers with more advanced malware prevention tools, and never allow bad files to be shared in the first place. BitTorrrent's use of a central tracker seems to allow it the possibility of easily blocking most malware.

As mentioned earlier, there are few instances of malware that uses a BitTorrent client to propagate automatically without user knowledge. There are many possible reasons for this, and this project did not examine each one. However, I will offer my current best hypothesis. Sharing a BitTorrent file requires running a BitTorrent client at all times without the user knowing (unlike other P2P protocols, where a virus can simply copy itself into a shared folder). In order to do this, a malicious program must obtain access to the user's computer, most likely by the user executing an infected file. Once the program has access to the computer, however, there are many different methods of sending malware - e-mail attachments, malicious websites, and instant messaging programs are a few common methods [9]. In comparison to all the other simple ways of spreading,

using the BitTorrent protocol seems like a much more intensive and expensive operation (although this was not tested), and an attacker may have an easier time using one of the previous methods. If this were true, it could be a main reason why automatic exploitation of BitTorrent clients is not common.

6 Limitations and Future Work

While completing this project, I thought of several new questions. As mentioned previously, a size limit of ten megabytes was chosen for all downloads. While this was chosen because I wanted to be able to download many torrent files without slowing the rest of the network to a crawl, many popular files on BitTorrent are larger than ten megabytes. For instance, a quick search in BTJunkie of the 50 highest-seeded .exe torrents show only 8 files that are at most ten megabytes. Therefore, the small file limit is not an ideal sample definition, even though practical considerations seem to require some sort of limit if many files are to be examined. Also, files were selected for downloading by checking for new torrent files inside the "Software" or "Application" categories of the torrent indexes. A better way, which also would generate a larger, more representative sample size, would be to automatically download every torrent as it appears. It would be interesting to see what would happen if the constraints on file size and category were removed.

In regards to sample size, it would also be helpful to gather more data on clean torrent files that are of the same age and size as the malware samples. When I started the project, I was concerned about using too many clean files, as I thought my resources were too limited to handle all malware and a significant portion of non-malware as well. Also, I had the paper that analyzed a popular torrent, and felt this was an adequate baseline. After completing the project, I found that the project's resources would have been adequate to analyze many more uninfected torrent files. Perhaps with more clean torrents analyzed, more clear and interesting patterns would emerge. If the project were extended in the future, this would be one area to consider collecting more data on.

A third area that seems worth investigating further is how effective torrent indexes are at removing torrent files that link to malware. Every index site that was used for the project had a way of rating different torrents, as well as reporting bad links or bad torrents (such as those infected with malware). As a precursory look at this question, I went back and examined four torrents from Mininova, and found that three of them were no longer available (for many of the files, I had no way of recovering what index site I retrieved them from due to my own unfortunate omission). An interesting question, then, is how long most files take to be removed (if they ever are) or how many downloads a file receives before somebody reports it as being bad.

Another issue involving torrent indexes is if there are certain index sites that link to more or less malware than the average. It is possible that certain torrent index sites link to more malware than others. If this is the case, it would be interesting to explore why it is true. For instance, some torrent index sites may respond to user comments faster, allow only trusted members to add torrents, or perhaps malicious users target only a small portion of trackers to list their malware. The differences between malware occurrences amongst torrent index sites may be a very interesting area to look at.

7 Conclusion

Malware is in fact quite common in BitTorrent, accounting for approximately one-fifth of all downloaded files in this project. While malware is present, this project's sample suggests it is much less

of a threat than in other P2P networks. This project was meant to serve more as an introduction to malware in BitTorrent, probing areas that may be worth further investigation. A closer examination of different aspects of malware with the BitTorrent protocol can help to make BitTorrent an even safer P2P protocol than it currently is.

8 References

[1] BitTorrent. At https://launchpad.net/ubuntu/+source/bittorrent/

[2] "BitTorrent: the 'one third of all internet traffic' myth". Accessed April 1, 2008 at http://torrentfreak.com/bittorrent-the-one-third-of-all-internet-traffic-myth/

[3] BTJunkie. At http://btjunkie.org/

[4] BushTorrent.com. At http://www.bushtorrent.com/

[5] ClamAV. At http://www.clamav.net/

[6] Enhanced CTorrent. At http://www.rahul.net/dholmes/ctorrent/

[7] isoHunt. At http://isohunt.com/

[8] M. Izal, G. Urvoy-Keller, E.W. Biersack, P.A. Felber, A. Al Hamra, and L. Garces-Erice. Dissecting BitTorrent: five months in a torrent's lifetime. In *Passive and Active Network Measurement*, 2004

[9] D. Gryaznov. Malware in Popular Networks. In *Virus Bulletin Conference October 2005*, 2005

[10] A. Kalafut, A. Acharya, and M. Gupta. A study of malware in peer-to-peer networks. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 2006

[11] R. Menta. "Top P2P applications: 1.6 million PCs rank them". *MP3Newswire.net* April 16, 2008. Accessed April 19, 2008 at http://www.mp3newswire.net/stories/8002/p2p.html

[11] Mininova. At http://www.mininova.org/

[12] S. Schiesel. "File sharing's new face." *The New York Times*, February 12, 2004. Accessed March 31, 2008 at http://www.nytimes.com/2004/02/12/technology/circuits/12shar.html?ex=1391922000&en=da75cefbee224928&ei=5007&partner=USERLAND

[13] "Ten Most Used BitTorrent Sites Compared." Accessed April 5, 2008 at http://comparebt.blogspot.com/

[14] Torrentsniff. At http://www.highprogrammer.com/alan/perl/torrentsniff.html

[15] Ubuntu. At http://www.ubuntu.com/

[16] Viruslist.com P2P-Worm.Win32.Achar.a. Accessed April 3, 2008 at http://www.viruslist.com/en/viruses/encyclopedia?virusid=23893

[17] Viruslist.com P2P-Worm.Win32.Mandragore. Accessed April 3, 2008 at http://www.viruslist.com/en/viruslist.html?id=4161

[18] VMWare Server. At http://www.vmware.com/products/server/

[19] Windows Server 2003. At http://www.microsoft.com/windowsserver2003/default.mspx

[20] Windows Worm uses BitTorrent to Propagate. Accessed April 2, 2008 at http://torrentfreak.com/windows-worm-uses-bittorrent-to-propagate/

[21] K. Zetter. "Kazaa delivers more than tunes." *Wired News*, January 9, 2004. Accessed April 8, 2008 at http://www.wired.com/techbiz/media/news/2004/01/61852