A BitTorrent-Driven Distributed Denial-of-Service Attack

Jerome Harrington, Corey Kuwanoe, Cliff C. Zou
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816
psi@y0ru.net, eschalon@gmail.com, czou@cs.ucf.edu

***Abstract*—BitTorrent is a popular peer-to-peer file-sharing protocol that utilizes a central server, known as a "tracker", to coordinate connections between peers in a "swarm", a term used to describe a BitTorrent ad-hoc file sharing network. The tracker of a swarm is specified by the original file distributor and trusted unconditionally by peers in the swarm. This central point of control provides an opportunity for a file distributor to deploy a modified tracker to provide peers in a swarm with malicious coordination data, directing peer connection traffic toward an arbitrary target machine on an arbitrary service port. Although such an attack does not generate huge amount of attack traffic, it would set up many connections with the victim server successfully and hold these connections until time out, which could cause serious denial-of-service by exhausting a server's connection resource. In this paper, we present such an attack that is entirely tracker-based, requiring no modifications to BitTorrent client software and could be deployed by an attacker right now. The results from both simulation and real-world experiments show the applicability of this attack. Due to the skyrocketing popularity of BitTorrent and numerous large-scale swarms existed in the Internet, BitTorrent swarms provide an intriguing platform for launching distributed denial-of-service attacks based on connection exhaustion.**

Keywords-BitTorrent, Distributed denial-of-service, Peer-to-peer networks

I. Introduction

Over the past several years, peer-to-peer (P2P) networks have enjoyed a tremendous rise in popularity, primarily as a means of transferring large files over the Internet. In 1999, "Napster" became the first peer-to-peer file-sharing network to attract mainstream attention and was widely used to share music via the Internet. The Napster network makes use of a centralized server to maintain a list of currently connected clients and the files that each client makes available at a given point in time [3].

The next peer-to-peer file-sharing protocol to garner a great deal of attention was "Gnutella" [4]. Unlike Napster, Gnutella provides a true peer-to-peer network that does not need centralized servers for client tracking. Instead, a Gnutella client requires only the address of a single remote peer to bootstrap its connection to the Gnutella network, obtaining the identities of further peers by querying the peer or peers used during the bootstrap phrase [4].

Following Gnutella, the next wave of peer-to-peer file sharing came in the form of the "FastTrack" protocol, which implemented a supernode-based architecture [5]. A supernode is a high-powered, well connected client in the P2P network that can assume the functionality of a directory server for a number of lower-powered or lesser-connected clients, relieving overheard from those machines and allowing for greater scalability. The most popular of the FastTrack-based networks, Kazaa, achieved a great deal of popularity around 2003 [5].

"BitTorrent" is a different peer-to-peer protocol for sharing large files over the Internet created by Bram Cohen [6]. Because of its file-centered design and its fairness mechanism that rewards users for up sharing [9], BitTorrent is very efficient in transferring large files among peers, and hence, has gained its popularity in the last several years [6].

However, BitTorrent has a serious vulnerability that has not been discovered before. BitTorrent utilizes a central server, known as a "tracker", to coordinate connections between peers in a "swarm", a term used to describe a BitTorrent ad-hoc file sharing network for a file (or a set of files) provided by a file distributor. The tracker of a swarm is specified by the swarm's original file distributor. All peers in the swarm trust the tracker without implementing any authentication or verification procedures. This central point of control provides an opportunity for a file distributor to deploy a modified tracker to provide peers in a swarm with malicious coordination data, directing peer connection traffic toward an arbitrary target machine on an arbitrary service port.

In this paper, we present a potential attack that exploits the above BitTorrent vulnerability. By deploying such an attack, a malicious attacker could use a popular file (such as a pirated movie) as a bait to launch an application-level, connection exhausting distributed denial-of-service (DDoS) attack using the members of a BitTorrent swarm. This BitTorrent-driven attack does not require any modifications to the client-side BitTorrent software, and hence, it could be immediately implemented in the current BitTorrent world by an attacker. Instead of sending out attack traffic from an attacker's

This research was supported by NSF grant CNS-0627318 and Intel Research Fund.

compromised computers, the actual DDoS attack traffic is initiated by the large number of innocent peers within a swarm, which makes the attack efficient, easy to be implemented, and hard to defend. Furthermore, the BitTorrent attack causes real, complete TCP connections to be made to an arbitrary service port specified by an attacker, allowing the attack to be adapted to a range of target services such as HTTP and SMTP.

The rest of this paper is organized as follows. The related work is introduced in Section II. We introduce BitTorrent and describe its architecture in Section III. In Section IV, we present details of the vulnerability inherent in the BitTorrent architecture, and provide the theory behind our DDoS attack. Then, in Section V, we evaluate our BitTorrent-driven attack, both by simulations and a small-scale real world experiment. We look at some possible defense tactics against the presented attack in Section VI. Finally we summarize this paper in Section VII and discuss some opportunities for future work.

II. Related Work

Many people have studied how to secure a peer-to-peer network so that the network can run normally when it is under attack, either from outside machines or from malicious members in the P2P network. Wallach [17] presented a survey of P2P network security including routing protocol, fairness and trust issues. Castro et al. [16] presented attacks to prevent correct message delivery in structured peer-to-peer overlays and also defenses to these attacks. Douceur et al. [15] studied Byzantine fault isolation in a P2P distributed file system. [18] and [19] presented a set of defenses against various denial-of-service attacks in P2P systems. Our research is about how attackers could use a P2P network to conduct large-scale DDoS attacks to other targets, not the P2P network itself.

Attackers could also take advantage of the P2P network infrastructure to facilitate their attacks to members of a P2P network, such as using a P2P network to propagate an Internet worm. Yu et al. [21] presented a P2P-based worm attack and provided its propagation model. Zhou et al. [20] presented a self-defense infrastructure inside a P2P network to contain P2P-based worms. A P2P-based worm tries to infect members of a P2P network, which is different from the BitTorrent-driven DDoS attack presented in this paper.

BitTorrent is a special P2P network protocol [9]. Current research on BitTorrent is mainly on modeling and analyzing its robustness, fairness and performance [22, 23, 24, 25]. Liogkas et al. [14] presented three selfish-peer exploits and studied BitTorrent's robustness under these exploits. No research has been done on exploiting BitTorrent to attack an arbitrary target.

The BitTorrent-driven DDoS attack we present in this paper utilizes the communication center (i.e., the tracker) of a BitTorrent swarm to direct connections from members of the network to a target. Conceptually speaking, this attacking idea is similar to the idea deployed by the botnet monitoring system presented in [26]: by hijacking the domain name of the communication center of a botnet (the command & control server), the botnet monitor is able to redirect connections from members of the botnet to itself [26].

"DNS reflection attack" [27] is an amplification flooding attack: an attacker sends spoofed DNS queries to many DNS servers, letting those DNS servers generating a larger volume of DNS response traffic to a spoofed victim. Compared to this DNS reflection attack, the proposed BitTorrent attack exhausts connection resource of a victim instead of bandwidth of a victim.

III. Background on BitTorrent

The BitTorrent protocol was developed by Bram Cohen and originally released in 2001 [2]. BitTorrent differs from earlier peer-to-peer protocols because there is not a single BitTorrent network. Instead, smaller ad-hoc networks, known as swarms, are formed for each file (or set of files) that is being transferred. The members of each of these swarms regularly announce their presence to a centralized server, or "tracker", which maintains a list of all currently connected peers, and distributes that list among the peers as they announce to the tracker.

To transfer a file among a group of users via BitTorrent, the original distributor of the file must first have a server available that is running BitTorrent tracker software. Then, he or she must generate a "torrent" file. This file contains the URL of the tracker to which peers in the swarm should announce. Additionally, the total payload to be transferred is divided into a group of smaller chunks that are all hashed separately, and these checksums are also added to the torrent file. The torrent file is then registered with the tracker. At this point, the distributor of the file must provide an initial "seed" for the torrent. That is, he or she must launch a BitTorrent client, load the previously created torrent file, and direct the client toward the file that he or she already possesses.

At this point, the torrent file can now be offered to potential downloaders via any normal distribution method. Most often this is done by publishing the torrent file on a popular website.

When a downloader retrieves a torrent file and opens it within their BitTorrent client software, the client reads the tracker URL contained in the torrent file, and announces itself to the tracker. The tracker then records the IP address of the new peer and a timestamp indicating the time at which the peer last checked in to its local database. The tracker returns a response to the client containing a list of addresses belonging to the other clients in the swarm, as well as an indication of whether or not each peer in the swarm is a seeder that possesses the file in its entirety or not.

Upon receipt of this list of peers from the tracker, the BitTorrent client then begins making connections to the listed machines, requesting individual chunks of the payload. When a chunk is received in its entirety, it is hashed, and the resulting checksum is compared to the checksum given for the chunk in the torrent file. If the two checksums match, then the client considers the chunk to be completed, and will no longer request that chunk of data from peers. This process continues, with the client connecting many peers at a time, requesting chunks and downloading them in parallel. Because of this parallel download model, each client transfers data with many active members in its swarm at the same time. Since transfer speeds actually increase as the size of the swarm grows, this makes BitTorrent ideal for transfer of files that are in high demand and provides a more economical means of file distribution than a traditional client-server model as the cost of bandwidth can be shared among all participants in a swarm.

While the payload is being downloaded, a client will announce regularly to the tracker, confirming its continued participation in the swarm and receiving updated peer lists each time it announces. Once a given client has successfully downloaded all chunks of the torrent payload and verified that their checksums are correct, that client will again announce to the tracker, declaring itself as a seeder.

Fig. 1 illustrates the architecture and communication paths of a typical BitTorrent swarm. As seen here, there exists a control path between each peer in the swarm and the centralized tracker server whenever the peer announces itself to the tracker. Any given peer may additionally have data connections with any other peers. Each data connection between peers is independent of all other data connections in the swarm. Therefore, a single peer is free to download chunks from any combination of peers within the swarm.

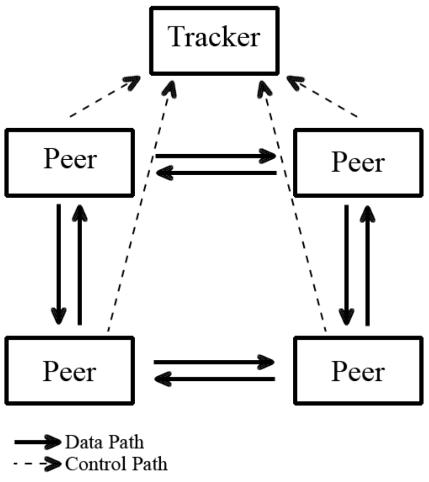


Figure 1: Architecture of a normal BitTorrent swarm

Although it was originally released in 2001, the BitTorrent protocol began to become truly popular among file sharers after the decline of the FastTrack-based networks, primarily starting in 2004 [10]. BitTorrent is currently being used for many legitimate and legal purposes, including distribution of Linux ISO images and software updates for Blizzard's multi-player online role-playing game World of Warcraft. Recently, Warner Brothers has made a deal with BitTorrent to distribute and sell over 200 Warner Brothers movies and TV programs through BitTorrent [11]. BitTorrent has arguably received more attention from legitimate interest groups than any previous peer-to-peer file sharing protocols or networks.

Meanwhile, BitTorrent has become massively popular among the piracy community, being used to transfer music, movies and television shows en masse. Such popular torrent directory websites, such as The Pirate Bay [7] and mininova [8], allow anyone to upload a torrent file, usually anonymously, pointing to an arbitrary tracker specified in the torrent file. Thus, it is possible for a malicious file distributor to create a torrent using a tracker over which he or she has full control, and offer it to a huge pool of potential downloaders in an effective and straightforward manner.

These torrent directories have proven to be quite popular, often hosting numerous torrents with one thousand or more downloaders or, in BitTorrent terms, leechers at any time.

IV. A Vulnerability in BitTorrent Architecture

A. *Modifying the Tracker*

As we have previously seen, peers in a BitTorrent swarm rely on the response provided by the tracker at announce time to determine the identities of the other clients within the swarm and, consequently, what peers they should connect to. Therefore, if an attacker has control of a tracker, he or she can alter the response that the tracker provides to the peers in the swarm.

In our experiments, we made use of an open-source PHP tracker called "BlogTorrent" [1]. Our modifications were limited to a single function called BTAnnounce() in the tracker program. This function is called each time a client announces to the tracker. The function's pseudo-code is listed below. It demonstrates the primary logic involved in accepting a peer announcement and returning normal output to a BitTorrent client:

```
1.  remote_addr = IP address of announcing peer
2.  port = BitTorrent port of the announcing peer

3.  // split the IP of the announcing peer
4.  // into an array of its octets
5.  peer_ip_array = split remote_addr on '.'

6.  // pack the contents of peer_ip_array into
7.  // a binary string of unsigned char
8.  peer_ip = pack("C*", peer_ip_array[0],
    peer_ip_array[1], peer_ip_array[2], peer_ip_array[3])

9.  // pack the peer's port number into
10. // a binary string of unsigned short
11. peer_port = pack("n*", port)

12. // generate 0-127 based on the current minute
13. time = round_to_int((time() % 7680) / 60)

14. // if the peer is a seeder, set the high bit of time
15. if peer is a seeder
16.     time = time + 128
17. endif

18. // pack time into a binary string of unsigned char
19. time_out = pack("C", time)

20. // concatenate time, peer_ip, and peer_port to
21. // produce the peer entry in the expected format
22. peer_data = time + peer_ip + peer_port
```

```
23. // update or add the peer to the local database
24. if peer is in database
25.     update entry to peer_data
26. else
27.     add peer to database with peer_data
28. endif

29. // initialize the string to be returned to the client
30. output = ''

31. // concatenate the database entries for all peers
32. // to generate the output to be returned
33. foreach peer
34.     output = output + peer
35. endforeach

36. // this is the string in the expected
37. // format that will be returned to the client
38. return 'd8:intervali1800e5:peers' + length(output) + ':' +
    output + 'e'
```

This function BTAnnounce() serves two main purposes. First, it serves to maintain a local database of currently connected peers within the swarm. Second, it is responsible for creating a peer list to be returned to the calling client, containing the addresses of the currently connected peers stored in the database.

We have modified this function to return our own list of configurable addresses along with the legitimate data returned normally. Given the pseudo-code above, our additions would take place between lines 30 and 31.

Here, we repeat the same process as shown in lines 3 to 22 to construct a data entry for each of our configured target address and port combinations. Our illegitimate entries are then concatenated and assigned to the output variable before line 31. Having pre-populated the output with connection information for our target or targets, we then proceed to add the legitimate peers to the output string, yielding a binary string to be returned to the BitTorrent client that contains the addresses of the legitimate peers within the swarm as well as the addresses (and service ports) of the arbitrary machine or machines that we have configured to attack.

With the ability to inject arbitrary IP addresses into the peer list, peers in the swarm will now attempt to connect to those targets in an attempt to download chunks of the advertised file. In the next section, we will see how this vulnerability in the BitTorrent architecture may be exploited to stage a DDoS attack.

Fig. 2 shows the network architecture of a modified BitTorrent swarm. The modified swarm is able to function normally as a file-transferring swarm since all peers can transfer files to each other normally. In addition, peers within the swarm will make additional connections to our supplied target.

A modified BitTorrent swarm can be configured to attack multiple targets as well. Note that the addresses of these multiple targets may belong to the same server or different machines. If a target has multiple IP addresses assigned to it (e.g., a web farm), this technique can be used to increase the number of connections being made to a target. Alternatively, this could be used to target several distinct machines at the same time.

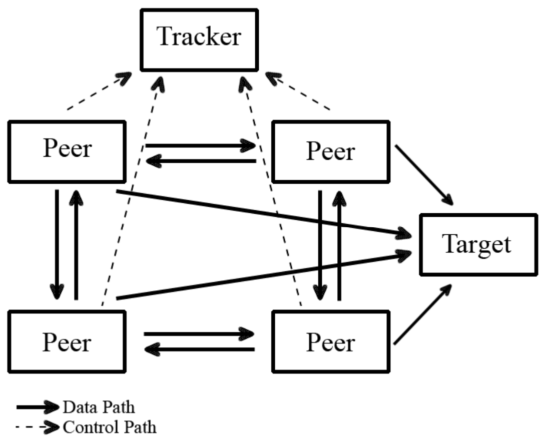


Figure 2: Architecture of our modified swarm (BitTorrent file exchange function is still working, but each peer would generate an additional connection to a victim target on a specified service port)

B. A BitTorrent-driven DDoS Attack

Given the current usage of BitTorrent and the capability of providing clients with arbitrary addresses in the peer list, it is not difficult to envision a realistic attack scenario. An attacker would first set up a modified tracker, most likely on a server that he or she has previously compromised. Next, the attacker would need to obtain a file or a set of files that are likely to generate high demand — a pirated copy of a blockbuster movie still in theatrical release or a popular new computer game, for example. With these prerequisites in place, the attacker is now free to generate a torrent file for his or her payload and register the torrent with the modified tracker.

While the attacker could use any means to distribute the newly created torrent, the most straightforward approach would be to upload the torrent to a highly trafficked torrent directory, such as The Pirate Bay [7]. Upon upload to such a site, the torrent is made available for any user to download freely and join the swarm. At that point, the peers in the swarm that are downloading the file will begin connecting to the supplied target or targets, while still retrieving the torrent payload data normally.

When the number of peers in the swarm with a connection open to an attacked target exceeds the maximum number of connections the target application is configured to accept, the victim service will no longer accept new TCP connections, rendering it unreachable and causing a successful denial-of-service attack.

From the descriptions above, we can see that this BitTorrent-driven DDoS attack has the following properties:

- It requires no modification whatsoever to the BitTorrent client software.
- The attack is hidden from clients since the attack traffic volume from each client is very small and all clients can still upload and download files normally.
- The attack can target multiple victims on arbitrary service ports specified by the attacker.
- The attack does not expose an attacker's real compromised machine (i.e., the tracker) to a victim.
- An attacker can arbitrarily decide the start and end time of a DdoS attack by controlling the tracker.

V. Evaluation of BitTorrent-Driven DDoS Attacks

A. *Attack Simulation*

First, we evaluate our theoretical attack by using a controlled simulation. We make use of a virtual machine running Slackware Linux 10.2 and an Apache 1.3.x web server to act as our target machine. Our modified BlogTorrent tracker software is run on Mac OS X with Apache 1.3.3. The Mac machine also supplies the initial seed client and runs a multitude of BitTorrent client instances (using the command-line client btdownloadheadless.py [12]), simulating the peers within the swarm.

To monitor the number of connections made to the target at any given point in time, a script is written to constantly parse the Apache server-status page, reporting the number of concurrent requests being served at each parsing time.

Additionally, a client-spawning script is used to control the generation of the client processes in the BitTorrent swarm. This script is written to wake up at the top of every minute, randomly spawning one to n client processes each minute (where n is an adjustable parameter), and logging the number of clients spawned along with the total number of peers in the swarm. The script would then go back to sleep until the top of the next minute.

Fig. 3 shows the results from a medium-scale simulation. The maximum number of client processes to be spawned is set at 260. The client-spawning script is configured to spawn up to 20 clients at the top of each minute. The attack is configured to be launched with no requirement on the minimum swarm size, and hence, the attack traffic, in terms of the number of attack connections, begins ramping up immediately, coming close to the current number of spawned peers. The magnitude of the attack closely follows the growing size of the swarm until the swarm reaches its maximum size 260. Once the size of the swarm stops growing, we observe that the attack magnitude then becomes cyclic for the remainder of the swarm's lifetime. In addition, the attack magnitude is slightly smaller than the maximum possible attack magnitude 260 (each peer can set up one connection to the target at a time). The reason for the cyclic dynamics is because the connections from some peers have timed out and the new connections from these peers have not yet set up.

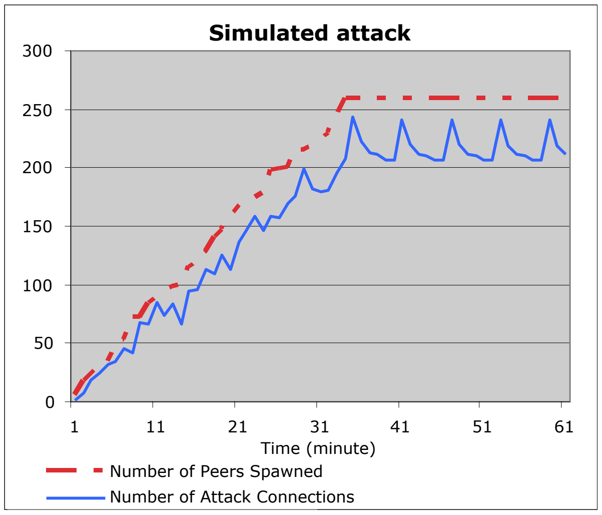


Figure 3: Swarm size and attack magnitude during a simulation study with 260 peers

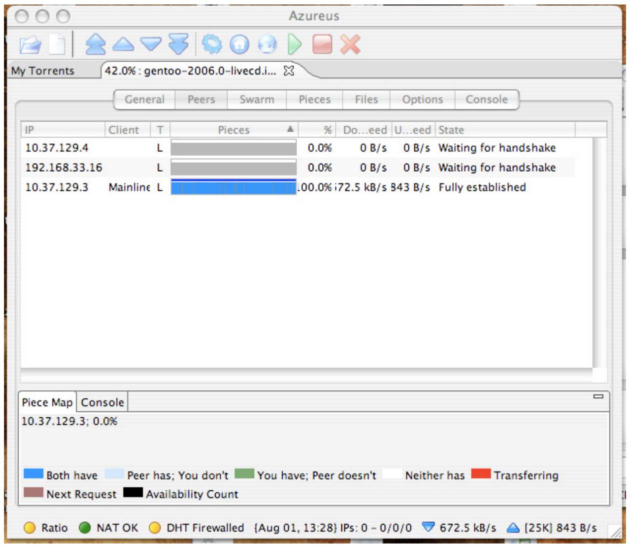
It would be better if we can verify the above prediction on a large-scale simulation as well, such as simulating a large swarm that has more than a thousand peers. Unfortunately, our current simulation testbed requires running all client processes on one single machine. A simulation with more than 500 peers would either cause out-of-memory error or crash our testbed machine. A large-scale simulation requires us to generate a brand-new simulation environment and run it on a cluster of machines. We plan to conduct this research as one of our future work.

B. *Peer Behavior*

Several interesting observations can be made with regard to the behavior of the attacking clients within a swarm. First, an attacking client will hold its connection to a target server open until the server times the connection out. For a standard Apache installation, this is usually a period of 5 minutes.

Second, during our simulation, we have also observed that BitTorrent clients re-attempt the connection every time they announce to the tracker. The clients have never blacklisted the target machine, despite the fact that it has never returned a valid response. This is due to the fact that the BitTorrent client software has not considered what to do when receiving an invalid response.

Another issue to be considered is what the actual data exchange looks like between a BitTorrent client and an attacked target. Upon connection to the target server, a BitTorrent client sends a request to the server containing the string "BitTorrent protocol". This does provide a possible means of fingerprinting the attack connections.



IP	Client	T	Pieces	%	Do...eed	U...eed	State
10.37.129.4		L		0.0%	0 B/s	0 B/s	Waiting for handshake
192.168.33.16		L		0.0%	0 B/s	0 B/s	Waiting for handshake
10.37.129.3	Mainline	L		100.0%	672.5 kB/s	843 B/s	Fully established

Piece Map | Console

10.37.129.3; 0.0%

🟦 Both have ▭ Peer has; You don't 🟩 You have; Peer doesn't ▭ Neither has 🟥 Transferring
🟫 Next Request ■ Availability Count

● Ratio ● NAT OK ● DHT Firewalled {Aug 01, 13:28} IPs: 0 - 0/0/0 672.5 kB/s [25K] 843 B/s

Figure 4: A screen shot of a peer client downloading a file normally while sending attacks to multiple targets at the same time

Fig. 4 is a screen capture of a popular BitTorrent client, Azureus [13]. In the screenshot, the client is currently being used as an attacker in our modified swarm. The "Azureus Peer Details" screen shows this client attacking two separate target addresses, 10.37.129.4 and 192.168.33.16. Simultaneously, the client is connected to a seeding peer, 10.37.129.3 and is transferring the torrent payload successfully. Note that Azureus displays a generic "Waiting for handshake" error. This error could be caused by a number of things, such as network latency or packet loss. Therefore, even if an end user checks this Azureus Peer Details screen, the user still has no clear clue that any malicious activity is taking place.

C. *Real World Experiment*

In addition to our attack simulation and experiments around the behavior of peers in a swarm, we conducted a small-scale "real world" test of our method. For this, we set up a modified tracker, generated a torrent file and registered it with our tracker. The torrent was then given out to a small group of friends, who were informed as to the nature of the experiment they were to be taking part in.

In this real-world experiment, our tracker was configured to provide the peers with the IP address of a web server under our control as the attack target machine. Furthermore, our tracker was configured with a value of one for the minimum number of peers required to begin the attack. We were thus able to see attack traffic being spawned immediately upon the swarm becoming active.

Fig. 5 illustrates the number of attack connections generated by our BitTorrent swarm during a 24-hour window of our swarm's lifetime. Each data point is the average number of attack connections served by the target web server within one hour. This average was calculated by tracking the number of open connections at an interval of one second. This running count was then totaled and divided by 3600 during later analysis to compute the average number of connections over the course of each 1-hour period.

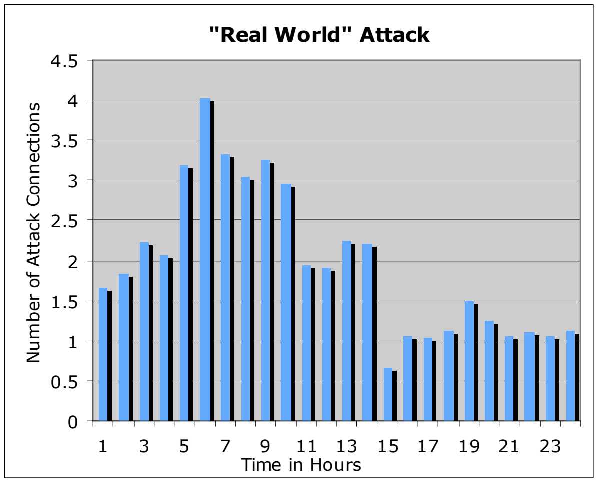


Figure 5: Attack magnitude during a "real world" experiment

We experienced a steady stream of new downloaders during the first six hours of this window. This correlates to the increasing number of attack connections being spawned during that time period. The attack peaked during hour 6, and after that time, declined in terms of the number of attack connections. This is due to the fact that our swarm experienced the heaviest influx of traffic early on. As the timeline progresses, the peers who joined early on are completing their downloads and becoming "seeding peers". As a result of this state change, those peers are no longer connecting to other peers in the swarm since they stopped seeking chunks of the offered file for download.

From the above analysis, we see that it is important that a swarm remains sufficiently populated with downloading peers at any given point in time to make this attack effective. This has been taken into account in our original attack proposal and is confirmed here. As mentioned in our attack proposal, large, public torrent directories have user pools large enough to make such attacks effective and sustainable for a sufficient period of time.

Another way in maintaining the population of downloading peers is to prevent them from finishing download and becoming "seeding peers". We have proposed two approaches, which are discussed in detail in Section VII.

VI. Possible Methods of Defense

There are several possible methods that could be used to defend against the proposed BitTorrent-driven DDoS attack. We introduce them in this section.

A. *Blacklist-based Traffic Filtering by Target*

The first possible method of defense against our attack is for the target to detect and block attack traffic. As explained in Section V.B, an attack TCP connection coming from a BitTorrent peer does not follow the target service specifications and contains the string "BitTorrent protocol". This makes it easy for a target machine to detect BitTorrent-driven attack connections and block further attack connections by using a blacklist.

This traffic filtering defense, however, has its limitations. Given the dynamic nature of BitTorrent swarms, the blacklist-based filtering defense requires the blacklist to be updated frequently in order to block attacks from new BitTorrent peers. In addition, the blacklist could be very large if a large BitTorrent swarm is used in the attack.

B. Torrent Validation

Another alternative for defending against the outlined attack is for BitTorrent directory websites to validate submitted torrent files before making them available to the public. Presumably, this validation would be done by: (1). joining the swarm specified in a torrent file; (2). analyzing the peer lists returned by the tracker over a period of time; and (3). verifying that the hosts returned in the peer lists are in fact legitimate peers.

However, this is not a trivial process, and it is prone to error. In addition, an attacker could set a minimum swarm size that must be satisfied before the tracker begins to behave maliciously. Thus, early in the torrent's lifespan, when verification is most likely taking place, the tracker would behave normally and not raise any alarms during the verification process.

C. Strict Client Protocol Checking

Another defense would be for the authors of BitTorrent clients to add an increased level of intelligence to their client software. This would still require that each peer connects to the target once. If a peer receives an invalid response (i.e., not following BitTorrent protocol) from a connected target, the client software on the peer can blacklist that target IP address and not attempt any further connections to the target.

However, such a defense could be effective only if a large portion of BitTorrent client software have been upgraded. This could be hard to realize, because currently there are more than 50 BitTorrent client software existed and used by users [30]. In addition, many users never bother to upgrade their BitTorrent client software as long as their client software works fine in file downloading.

D. Disabling Malicious Trackers

The victim of an attack could attempt to locate the malicious tracker and have it disabled or removed from the network through the collaboration with the network or ISP hosting the tracker machine. The tracker is the single point of failure for the survivability of the DDoS attack introduced in this paper.

As with all DDoS attacks, this is easier said than done. First, the victim needs to identify the tracker based on the limited data transmitted by a BitTorrent swarm (the tracker does not attack a victim directly). Second, shutting down a remote tracker would need collaboration from another ISP, which is a time and resource consuming task.

E. Behavioral Anomaly Detection

We have introduced several deterministic techniques above for detecting and filtering the BitTorrent attack. Another defense method is to use behavioral or probabilistic-based anomaly detection. Researchers have proposed many anomaly detection based DDoS defense systems, such as [28, 29]. Ranjan et al. [28] presented a "DDoS-resilient scheduler" to prioritize incoming connections based on a suspicion score assigned to each connection. Kim et al. [29] presented a dynamic DDoS defense system "based on a per-packet score which estimates the legitimacy of a packet given the attribute values it carries."

Anomaly detection based DDoS defense would also be suitable for detection and defense of the proposed BitTorrent based DDoS attack. It is a rich research area to explore, but we will not discuss further since it is out of the focus of this paper.

VII. Conclusion

In this paper, we presented a vulnerability inherent in the BitTorrent architecture which can be leveraged to cause innocent peers within a swarm to make connections to one or more configured target machines on arbitrary service ports. Further, we showed how a malicious attacker might utilize this vulnerability to launch a distributed denial-of-service attack. Although such an attack does not generate huge amount of attack traffic, it could cause serious denial-of-service by exhausting a victim server's connection resource.

An attack launched using our method has a number of positive properties from the perspective of attackers. First, the attack does not require any modification to the BitTorrent client software used by peers within a swarm — it only requires a modified tracker to be run. Second, the attack provides a high level of configurability, with the ability to configure an arbitrary number of hosts and arbitrary service ports to be attacked. Third, because the attack does not affect the normal transfer of file-sharing data within a swarm, the attack is stealthy to existing BitTorrent clients, reducing the likelihood that a peer would notice that it was being used to generate attack traffic. Fourth, the attack also hides the identity of the real malicious host, the tracker, from the target machine or machines. Attack victims will only receive connections from the peers within a swarm. Finally, an attacker can arbitrarily start or stop the attack and the peers will respond accordingly upon their next announcement to the tracker.

A. Future Work

There are several areas in which additional research is useful. The primary opportunities for improvement to our work at this point are in artificially extending a peer's life as a downloader and further analysis of the behavior of large-scale swarms that are employing our method.

As seen previously, it is imperative to maintain a high saturation of downloaders within a swarm to increase the effectiveness of an attack. This is because seeding peers will not initiate peer connections to retrieve data chunks, as they already possess the complete payload. Thus, if attackers can extend the length of time that a peer remains a downloader, rather than a seeder, they can increase its usefulness as a member of the attack.

There are several possible approaches that might be taken to extend the lifetime of a downloader. First, the initial seed's upload bandwidth could be throttled much lower than its maximum throughput. By limiting the amount of data that the initial seed is able to push out to peers requesting chunks of the file, it will take each peer a longer time to download the torrent payload, resulting in an extended download lifetime for peers that are downloading from the initial seed.

A second approach, which is possibly more effective, to increasing the lifetime of a downloading peer is to attempt to provide inaccurate data from the initial seed. For example, the initial seed may be running a modified client that modifies a chunk of data before sending it to the requesting peer. This behavior could be configurable and controllable by the attacker. For instance, a modified seed might properly deliver all chunks of a file except for one. If left like this indefinitely, all downloading peers would eventually receive the entire payload accurately, with the exception of the last chunk. From the client perspective, this has the effect of causing the download to be stuck at, for example 99% for an extended duration. After some time has passed, the attacker could disable this behavior on the seed, and allow the waiting clients to complete the download successfully. By using such a technique sparingly, it is quite likely that the attacker could increase the effectiveness of his or her attack while maintaining relative stealth.

The second area in which our work could be built on is to study the behavior of larger swarms carrying out our attack. Due to limited resources, both the simulation and our "real world" experiment were executed with relatively small swarms. It would be useful to stage simulations involving hundreds or thousands of peers, as this would more realistically model the environment in which a live attack would be executed. We are currently working on joining the PlanetLab project [31] and use this large-scale distributed network to conduct a realistic denial-of-service attack experiment.

Finally, it may also be possible to extend the techniques presented here to other peer-to-peer networks. This technique is potentially applicable to any peer-to-peer network that has a centralized control server. It may be possible, for example, to create a modified Gnutella supernode capable of reporting to clients that a given host has a specific file when, in fact, it is the target of an attack.

References

[1] http://www.blogtorrent.com

[2] C. Thompson, "The BitTorrent Effect". http://www.wired.com/wired/archive/13.01/bittorrent.html

[3] http://en.wikipedia.org/wiki/Napster

[4] http://en.wikipedia.org/wiki/Gnutella

[5] http://en.wikipedia.org/wiki/Fasttrack

[6] http://en.wikipedia.org/wiki/Bittorrent

[7] http://thepiratebay.org/

[8] http://www.mininova.org/

[9] B. Cohen, "Incentives Build Robustness in BitTorrent". http://www.bittorrent.org/bittorrentecon.pdf

[10] R. Naraine, "BitTorrent, 'Gi-Fi,' and Other Trends in 2004". http://www.internetnews.com/ent-news/article.php/3294271

[11] B. Helm, "BitTorrent Goes Hollywood", BusinessWeek, May 2006. http://www.businessweek.com/technology/content/may2006/tc20060508_693082.htm

[12] The MST3K BitTorrent Guide. http://mst3k.booyaka.com/bittorrent_guide.shtml

[13] Azureus : Java BitTorrent Client. http://azureus.sourceforge.net/

[14] N. Liogkas, R. Nelson, E. Kohler, L. Zhang, "Exploiting BitTorrent For Fun (But Not Profit)", 5th International Workshop on Peer-to-Peer Systems (IPTPS), 2006.

[15] J. R. Douceur, J. Howell, "Byzantine Fault Isolation in the Farsite Distributed File System", 5th International Workshop on Peer-to-Peer Systems (IPTPS), 2006.

[16] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Security for structured peer-to-peer overlay networks." In OSDI, 2002.

[17] D. Wallach, "A Survey of Peer-to-Peer Security Issues", International Symposium on Software Security, 2002.

[18] P. Maniatis, T. J. Giuli, M. Roussopoulos, D. S. Rosenthal, M. Baker, "Impeding attrition attacks in P2P systems", Proc. 11th workshop on ACM SIGOPS European workshop: beyond the PC, 2004.

[19] T. J. Giuli, Maniatis, M. Roussopoulos, M. Baker, D. S. Rosenthal, M. Roussopoulos, "Attrition Defenses for a Peer-to-Peer Digital Preservation System", Proc. USENIX Technical Conference, 2005.

[20] L. Zhou, L. Zhang, F. McSherry, N. Immorlica, M. Costa, S. Chien, "A First Look at Peer-to-Peer Worms: Threats and Defenses", 4th International Workshop on Peer-To-Peer Systems (IPTPS), 2005.

[21] Wei Yu, Coryer Boyer, Sriram Chellappan and Dong Xuan, Peer-to-Peer System-based Active Worm Attacks: Modeling and Analysis, in Proc. of IEEE International Conference on Communications (ICC), 2005.

[22] A. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and Improving BitTorrent Performance", In Proc. Infocom 2006.

[23] D. Qiu and S. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Perr-to-Peer networks", In Proc. SIGCOMM 2004.

[24] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, Analysis, and Modeling of BitTorrent-like Systems", In Proc. ACM/SIGCOMM Internet Measurement Conference (IMC), 2005.

[25] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In 4th International Workshop on Peer-to-Peer Systems (IPTPS), Feb 2005.

[26] D. Dagon, C. C. Zou, and W. Lee, "Modeling Botnet Propagation Using Time Zones," Proc. 13th Annual Network and Distributed System Security Symposium (NDSS), 2006.

[27] G. Evron and R. Vaughn, "DNS Amplification Attacks," http://www.securiteam.com/securityreviews/5GP0L00I0W.html

[28] S. Ranjan, R. Swaminathan, M. Uysal, and E. Knightly, "DDoS-Resilient Scheduling to Counter Application Layer Attacks under Imperfect Detection," Proc. INFOCOM, Barcelona, Spain, 2006.

[29] Y. Kim, W.C. Lau, M.C. Chuah, and H.J. Chao, "Packetscore: Statistics-based Overload Control Against Distributed Denial-of-Service Attacks," Proc. INFOCOM, HongKong, 2004.

[30] Comparison of BitTorrent software. http://en.wikipedia.org/wiki/Comparison_of_BitTorrent_software

[31] PlanetLab: An open platform for developing, deploying, and accessing planetary-scale services. https://www.planet-lab.org/