BitTorrent Protocol Abuses

Written by Michael Brooks and David Aslanian

Introduction

The BitTorrent protocol is now 7 years old. The protocol has become wildly successful in a very short period of time, but with this success comes growing pains. The protocol originally envisioned by Bram Cohen is fairly secure. Vulnerabilities have been introduced by people trying to do things with the protocol that it was never meant to do. BitTorrent was never meant to be an exclusive protocol; however private BitTorrent networks are increasing in popularity.

Most vulnerabilities can be identified by a Common Vulnerability and Exposure (CVE) number. Currently there are 36,976 vulnerabilities classified by this system[i]. Most CVE numbers fall into specific categories and given a Common Weakness Enumeration (CWE) number. For instance, all SQL Injection Vulnerabilities are classified by CWE-89.

In the current BitTorrent Specification there are three CWE violations.

Immortal Sessions

All private BitTorrent communities are vulnerable to CWE-384[ii] (Session fixation) which ranks in at Number 7 of the OWASP top 10[iii]. All BitTorrent Clients authenticate with private trackers using a session ID sent as a GET parameter. This value cannot be changed because there is no system in the protocol to notify the BitTorrent client of a new Session ID.

One could exploit this vulnerability by sniffing the network for announce requests or .torrent files. Another tried and true method is XSS in the web application. Due to this session fixation vulnerability, SQL Injection in the tracker leads to immediate access without need to break a password hash. Often immortal session Ids are a password hash, which is the case with many of the private BitTorrent networks. Password hashes should always be kept secret. The hashing of passwords is meant to be a last line of defense after the database has been compromised, but many programmers don't understand this principle.

Client Side Trust

Another vulnerability that exists in every private network is CWE- 602 [iv] “Client-Side Enforcement of Server-Side Security” otherwise known as Client Side Trust. With every announce request the client is expected to report on how much they have uploaded, downloaded and how much is remaining to download. Private BitTorrent networks use these announce requests to enforce a share ratio. For instance some Private BitTorrent networks will impose a

50% ratio, in which you must upload ½ as much as you download. The tools used to spoof this information are well known. These simple tools send HTTP GET requests informing the tracker that the client has uploaded data, when in fact little or no data has been uploaded by the BitTorrent client. RatioMaster is an example of a spoof client. However some BitTorrent communicates have been able to detect this method. One approach to detecting this type of cheating by looking at a large jump in the amount of data transferred. Another approach would be to identify clients that are uploading when no "leechers" are present.

A more difficult to detect method is if the BitTorrent client lies to the tracker saying that it hasn't transferred any data. In addition the client never reports that the transfer as completed. From the announce server's perspective the BitTorrent client is unable to contact other people in the swarm, which could happen under legitimate conditions. The BitTorrent client used to accomplish this is called StealthBomber. Modifications where to the original Python BitTorrent 5.2.2 client written by Bram Cohen to produce this client. The StealthBomber is open source and can be download here: http://code.google.com/p/bittorrenthacks/

Man in the Middle Attack Against Message Stream Encryption

The 3rd vulnerability to affect BitTorrent falls under CWE – 300[v] (Channel Accessible by Non-Endpoint) better known as a "Man in the Middle" attack. This is a vulnerability in the MSE protocol used for BitTorrent encryption. This is not the first security problem discovered that affects MSE. A company by the name of Ipoque is able to detect and throttle MSE traffic using Deep Packet Inspection.[vi] Further more, a research paper titled Attacks on Message Stream Encryption[vii] goes over many problems with MSE. The biggest problem with this paper is that many of the attacks discussed are highly theoretical and may not possible in the real world. To make matters worse not source code has been provided to prove or otherwise verify any of the attacks presented in the paper. The attack on MSE discussed in this paper was developed without knowledge of the paper "Attacks on Message Stream Encryption", and source code to the following attack is public. [viii]

Threat profiling is the most important part of building a cryptographically secure protocol. The engineer must be very clear on what he is trying to defend against and must use what is already available. Good engineering is standing on the shoulders of giants. MSE on the other hand is re-inventing the wheel. In fact some of the threats facing protocols like MSE where discussed in the fundamental paper "New Directions In Cryptography" written by Whitfield Diffie and Martin E. Hellman in 1976.[ix] If the engineers that built MSE had read "New Directions In Cryptography", they probably would have built their protocol differently.

What is MSE's threat model?

The following quote is from the MSE specification on bittorrent.org.[x]

"The objective is NOT to create a cryptographically secure protocol that can survive unlimited observation of passing packets and substantial computational resources on network timescales. The objective is to raise the bar sufficiently to deter attacks based on observing ip-port numbers in peer-to-tracker communications. "

In the reference for the protocol its self they are saying that its insecure. It doesn't go out and say it explicitly, but the whole reason why MSE was made was to fool ISPs' traffic throttling ability. Your ISP is a very powerful threat. You have to ask the ISP to use their wires and their routers to access their network.

What are some of the problems with MSE?

MSE uses RC4, which is proven to be weak by the Fluhrer, Mantin and Shamir attack. This attack is most notably used in cracking WEP. MSE is using RC4-drop 1024 in an effort to improve RC4, but this isn't a perfect fix. When AES is implemented properly, it is far more secure than RC4 and that's why AES is implemented in WPA2. Many of the other attacks developed to speed up WEP cracking are not possible with MSE.

The real problem with MSE is how the RC4 keys are generated. Currently, MSE is using a SHA1 hash to generate the RC4 key. The RC4 key is a 20 byte SHA1 hash. The key is base-256, which is a full byte. Cryptographically secure keys should always be a full byte, and this is similar to the string-to-key function outlined in RFC 2440[xi]. However, this is an unnecessary calculation, a protocol with equivalent security could be built without ever having to make this SHA1 calculation. Instead the key generation could rely on randomly generated numbers which are cheaper to calculate. During a handshake each client must make two SHA1 hash calculations which is a waste.

The SHA1 hash contains three elements. The first element is a static string "KeyA". This would make a prefixing attack against SHA1 much more difficult. The problem is that the attacker shouldn't know the hash in the first place so generating a collision is out of the question. This is probably a novice's mistake that just ends up wasting resources. The second element is a shared secret derived from a Diffie-Hellman (DH) key exchange. DH key exchanges are very important as they provide a shared secret. It all comes down to secrets and lies, this is the title of a great book by Bruce Schneier. How do you keep a secret? How do you tell if someone is lying? Modern cryptography has powerful tools to answer these two important questions.

DH key exchanges allow you to share a secret with someone. If the person lies about who they are, then your secret is worthless. This is where Man In The Middle attacks come into play. SSL/TLS also uses a DH key exchange; however, SSL/TLS is accompanied by a public-key and often a Public Key Infrastructure (PKI) to verify identity.

This leads us to the third and final element of the RC4 key which is the "infohash" of the torrent. The "infohash" is a SHA1 hash of the "info" sub-tree contained in the .torrent file. There are three ways that this "infohash" can be leaked to a passive observer. The infohash is apart of every announce and scrape request that the BitTorrent client sends to the BitTorrent tracker. This is ideal from the attacker perspective because a scrape request must be sent prior to any MSE connection. The "infohash" is also contained in the handshakes with other peers in the swarm. Finally the "infohash" could be obtained by sniffing the transfer of the .torrent file.

MSE does have a specification for announce requests should use a SHA1 hash of the infohash. There are a few problems with this. The biggest is that I can't find a single announce server that supports this feature and thus it is never used. Also the double-SHA1 could be easily cracked if the attacker had a database of all known infohashes. An ISP could build a very inclusive database of known infohashes and their corresponding double-sha1 counter parts (a rainbow table) by monitoring their entire network. Keeping such a widely used piece of information a secret is not practical. Further more, no static string should ever be used for this type of identity verification. There is a strong parallel between this use of the double-sha1 "infohash" and the use of a password's message digest as a session ID (which was discussed at the beginning of the paper).

What is the problem with a static string, and what makes asymmetric cryptography so important? A static string is like a password. In order to use a static string for authentication, it must first be distributed. Securely transmitting the static string is difficult, and a successful attack would mean that the secret could be reused by anyone indefinitely. In contrast, when a public key is used to verify the authenticity of a signature, your private key remains a secret. Hence, this means of authentication does not sacrifice your identity. That is to say, an attacker could not then use your signature to claim to be you. Specifically in terms of MSE the static string can be guessed. In the source code provided for MSE MITM, we make the assumption that we don't know the exact key used for an incoming connection. We are given cipher text, but we are missing the exact "infohash" portion of the RC4 key. Within the MSE protocol 8 null bytes are used to verify that the RC4 cipher-text stream has been decrypted properly, this is referred to a Verification Check (VC). This check can be used against the protocol. We can loop though a list of probable keys, when the 8 null bytes are decrypted present in the decrypted message we know we have the correct "infohash" and we can complete the handshake.

Ideal Conditions for MSE.

So, what if the .torrent file was sent over a proper HTTPS connection? What if all peer communication was also encrypted with MSE? What if the announce server was also over HTTPS or supported MSE? What if the .torrent file was on a private network? If all these conditions were met, then the attacker could not complete the RC4 key and therefore could not act as Man In The Middle.

There are a number of problems with this scenario. First of all, there is nothing forcing the client to use HTTPS. Nothing is forcing the Tracker to use MSE, and I can't find a single Tracker that supports it! The cryptographic chain is broken. In fact, I suspect that every single BitTorrent client is leaking the very secret that MSE relies on.

There is yet another problem; only two BitTorrent clients (Opera and uTorrent under Windows) will verify HTTPS properly. At the time of this writing uTorrent under Wine doesn't even support https. Many BitTorrent clients don't even bother to verify the TLS certificate and always assume it is valid. An advanced attack such as SSLStrip isn't required; dsniff is more than enough to carry out this attack. Azurues/Vuze, which was the first to implement MSE does not recognize any Certificate Authority as valid, which is just as bad. Unauthenticated HTTPS carries the exact same Man In The Middle problem as MSE. HTTPS is similar to MSE, they are both using a DH key exchange and HTTPS can also use RC4. Due to compatibility, no client by default forces MSE, so the key is being leaked in handshakes to other peers in the swarm. Most

BitTorrent communication is done using public trackers such as ThePirateBay.org and thus all of the infohashes are public knowledge.

MSE can defend against a specific attack, and that is NSA Signal Intelligence. To this date, the vast majority of the spying being conducted on US Citizens has been passive. In this case, MSE does quite well. I bet there is an NSA guy thinking to himself: "Yeah, I can break that tiny RC4 key with one command my big ass server farm."

A Real Solution

So what is the solution to all of these problems? MSE should never have been built! Let me be very clear, MSE was built by people who don't fully understand the cryptographic systems they are using. MSE is wasteful of resources and does not provide the type of protection that the authors desired. In fact, the solution to traffic shaping and keeping your data a secret existed long before BitTorrent. The answer is SSL, which is now called TLS. TLS is both efficient and effective, two things that MSE is not.

TLS or Transport Layer Security is the solution and should have been used from the very beginning. TLS is an encrypted tunnel for the OSI Layer 4 otherwise known as the Transport Layer. This means that it doesn't matter what the Application Layer looks like, the traffic will indistinguishable to the attacker. If BitTorrent used TLS then an ISP could not disrupt it without affecting HTTPS and other TLS dependent protocols. To make BitTorrent disruption even more difficult, clients can change their communication port to 443 or another port commonly used by TLS. However, in more extreme cases such as Iran, encrypted communication is indiscriminately blocked. [xii] An even more extreme solution is required to circumvent this terrible practice.

A very important feature of TLS is that it supports the ability to resume handshakes. This makes it so that only one TLS handshake is required per user in the swarm, even though the clients are continuously connecting and disconnecting from each other. Resuming TLS handshakes is an important part of FTP over TLS.[xiii] I suspect that the authors of MSE didn't know of this property of TLS, which could have been their motivation for reinventing the wheel.

Another important feature of TLS is that both the server and the client can verify each other's identity! This addresses the session fixation issue described earlier between the client and tracker. BitTorrent clients could also authenticate other peers in the swarm. This is an effective means of securing private BitTorrent networks. But who would be the Certificate Authority? Verisign doesn't support piracy and their trust is expensive! Furthermore, I don't trust Verisign, but we all are forced to. The best solution is that the BitTorrent tracker could also be a Certificate Authority. If a member of the community is kicked out, their certificate could be revoked immediately. OpenCA is free and simple to use, with minor modifications I believe it could be an effective Certificate Authority for BitTorrent networks.

---

[i] http://cve.mitre.org/cve/ (Total CVEs given in the upper right)
[ii] http://cwe.mitre.org/data/definitions/384.html (Session Fixation)
[iii] http://www.owasp.org/index.php/Top_10_2007-A7 (Broken Authenticantion and Session Management)
[iv] http://cwe.mitre.org/data/definitions/602.html (Client-Side Enforcement of Server-Side Security)

[v] http://cwe.mitre.org/data/definitions/300.html (Man In the Middle)
[vi] http://www.prweb.com/releases/2007/04/prweb521773.htm (Improved Identification of Encrypted P2P Protocols)
[vii] http://www.tcs.hut.fi/Publications/bbrumley/nordsec08_brumley_valkonen.pdf (Attacks On Message Stream Encryptoin)
[viii] http://code.google.com/p/bittorrenthacks/ Source Code for A MITM against MSE.
[ix] http://www.cs.rutgers.edu/~tdnguyen/classes/cs671/presentations/Arvind-NEWDIRS.pdf (New Directions in Cryptography)
[x] http://www.bittorrent.org/beps/bep_0008.html
[xi] http://www.ietf.org/rfc/rfc2440.txt (OpenPGP Message Format)
[xii] http://asert.arbornetworks.com/2009/06/a-deeper-look-at-the-iranian-firewall
[xiii] http://en.wikipedia.org/wiki/Secure_Sockets_Layer#Resumed_TLS_handshake