Protocol Encryption and Message Stream Encryption for WebTorrent

Brian Gu, Kelvin Lu

MIT, 6.858 (Computer Systems Security)
{bgu, kelvinlu}@mit.edu

May 12, 2018

Abstract

Message Stream Encryption (MSE), also known as Protocol Encryption (PE), is an extension to the Bittorrent Protocol that encrypts the Bittorrent stream and header information. The goal is to make Bittorrent traffic harder to detect by third parties (e.g. ISP's) while maintaining high download speeds. In this paper, we assess the security of PE/MSE and why it was important to implement for WebTorrent, a Javascript-based Bittorrent client for web browsers and Node.js servers.

I. Introduction

Readers can find Related Pull Request/Code here and here.

The goal of Message Stream Encryption (MSE) and Protocol Encryption (PE) is to obfuscate the header for third-parties to detect BitTorrent traffic by encrypting header information. This provides anonymity and confidentiality for peers, which is important in countries where ISP's are allowed to ban peers suspected of illegal file sharing. Furthermore, BitTorrent is a favorite target of ISPs for throttling mechanisms[1], and PE makes BitTorrent traffic harder to detect and throttle. Protocol Encryption also provides defense against man-in-the-middle attacks, preventing third-parties from spying on plaintext data being sent over the protocol.

Section II and III give context as to what problem we are solving and what the threat model looks like. Section IV details the PE/MSE handshake protocol, and Sections V and VI discuss the protocol and our discoveries while researching and implementing this protocol for WebTorrent.

[1]TorrentFreak, *Comcast Throttles BitTorrent Traffic, Seeding Impossible*

II. Background

i. Bittorrent

The BitTorrent Protocol is one of the most popular protocols for peer-to-peer file sharing. The peer-to-peer file sharing network model differs significantly from the typical client-server model in that all participants (nodes) on the peer-to-peer network are equally privileged: all nodes both consume and supply resources. This contrasts with the client-server model, where a central authority is needed to coordinate between clients. As of 2015, AT&T estimates that BitTorrent makes up 20% of all broadband traffic[2].

To download a file from a BitTorrent network, a peer user must obtain the corresponding `.torrent` file or magnet URI to discover peers who currently possess the file. The user then reaches out to each seeding peer and initiates the BitTorrent protocol to download portions of the file from each seeder. Once the user has downloaded the file, they are then also seeders of the file and other peers can download the file from them.

[2]AT&T, *AT&T patents system to âĂŸfast-laneâĂŹ BitTorrent traffic*

ii. WebTorrent

WebTorrent is an open-source torrent client written in Javascript, which works both in-browser and as a Node.js application. WebTorrent implements the BitTorrent protocol as well as a number of features that are interesting in the context of usability and security.

When run in browser, the WebTorrent client uses a modified version of the BitTorrent protocol and WebRTC, which encrypts data streams. The WebRTC protocol is important as the in-browser client does not support TCP and instead relies on WebRTC for transport. Our focus is on the Node.js server implementation of WebTorrent that uses TCP, as the browser client is incompatible with most BitTorrent clients and is already provided encryption (both a result of using WebRTC).

The current server implementation of WebTorrent does not provide PE/MSE, which is an issue for servers communicating with peers with restrictive ISP's. The lack of encryption is also an issue for users with particularly strict firewalls.

III. Threat Model

The threat model which Protocol Encryption addresses is one in which attackers may read the data streams between peers, in order to possibly:

- Passively eavesdrop, with the goal of protocol or content identification.
- Actively execute MITM attacks.

.

An example of an attacker which may read the data streams (i.e. TCP transports) between peers is an Internet Service Provider, which may do so with the goal of identifying and throttling BitTorrent traffic. Many ISPs are easily able to identify unencrypted BitTorrent traffic simply by reading the plaintext handshake sent between clients; the unencrypted handshake always begins with `\19Bittorrent Protocol`.

Our goal is to make BitTorrent traffic look indistinguishable from other Internet traffic so that ISP's and other eavesdroppers cannot detect that we are using BitTorrent. We also want to ensure that the PE/MSE handshake itself is secure from detection by eavesdroppers and attackers that use MITM attacks.

It is important to note that PE/MSE is **not** meant to provide security against an untrustworthy peer. The original BitTorrent protocol ensures that seeders cannot send a different file than what was originally asked for without downloaders noticing, and PE/MSE is meant to add an additional layer of security above BitTorrent without compromising any of its features.

IV. PE/MSE Protocol

i. Cryptographic Handshake Protocol

The BitTorrent protocol begins with a Diffie-Helman key exchange and encrypts using a RC4 keystream cipher. Suppose that Client A is a BitTorrent client that is seeking to download a file, and suppose that Client B is a BitTorrent client that is seeding the same file. When Client A initiates a connection with Client B, the handshake proceeds in five blocking steps. In the first two steps, the two clients exchange Diffie-Helman Public Keys in order to negotiate a shared secret. In the next two steps, the two clients verify the shared secret and shared torrent info hash, and negotiate a shared encryption method (usually RC4 cipher). These steps are described in more detail below.[3]

1. Client A sends their Diffie-Helman Public Key using $Y_a = G^{X_a} \pmod{P}$. G and P are known constants. Client A then sends a random-length padding `PadA` between 0 and 512 bytes.

2. Client B sends their Diffie-Helman Public Key Y_b, along with a random-length padding `PadB`. By this point, Client A and Client B both can calculate a shared secret $S = Y_a^{X_b} \pmod{P} = Y_b^{X_a} \pmod{P}$.

[3]Specification taken from Vuze Wiki

```
A->B: D-H Ya + PadA
-------------------------------------------------------------------------
B->A: D-H Yb + PadB
-------------------------------------------------------------------------
A->B: HASH('req1', S) + HASH('req2', SKEY) xor HASH('req3', S) + ENCRYPT(VC,
      crypto_provide, len(PadC), PadC, len(IA)) + ENCRYPT(IA)
-------------------------------------------------------------------------
B->A: ENCRYPT(VC, crypto_select, len(padD), padD) + ENCRYPT2(Payload Stream)
-------------------------------------------------------------------------
A->B: ENCRYPT2(Payload Stream)
```

Figure 1: *The PE/MSE handshake. Pluses are concatenations and quotations indicate string literals.*

3. Client A sends over three to four items in concatenation:
   - A SHA-1 hash of the string 'req1' concatenated with the Secret S. This allows Client B to resynchronize their buffer after the random-length padding
   - A SHA-1 hash of 'req2' with `SKEY` xor'd with a SHA-1 hash of 'req3' with Secret S. Client B is able to identify which infohash Client A is requesting with this, and initialize its RC4 keystream.
   - An encrypted concatenation of: a protocol verification constant (`VC`), a list of encryption methods Client B may select from (`crypto_provide`, methods currently supported are plaintext and RC4 encryption), and a pad `PadC` and its length (between 0 and 512 bytes).
   - (Optional) Client A may choose to begin and encrypt the regular bittorrent protocol (`IA`) To ensure synchronization, Client A always sends the length of IA (0 if not sent)
4. Client B sends an encrypted concatenation of: the same verification constant `VC`, an encryption method `crypto_select` selected from the list of methods `crypto_provide` provided by Client A, and a pad `padD` and its length (between 0 and 512 bytes). This ends the handshake.
5. From here on out, Client A and Client B communicate with the encryption method chosen by Client B (`ENCRYPT2`). This is usually continuing the RC4 cipher, though some clients may opt to revert to plaintext communication.

The stream key `SKEY` is the infohash of the torrent being requested. Client A will know this because it is initiating the connection in order to download a torrent with that infohash. In the current implementation of WebTorrent, a client may seed multiple files, so the seeding Client B will be able to determine which file Client A is requesting based on the hash of the stream key. See Figure 1 for a summary of the handshake.

ii. Handshake Security

Here, we explain the security features provided by the components of the cryptographic handshake.

In the first two steps, each client sends a sequence of between 96 and 508 bytes that are indistinguishable from a random sequence. The purpose of concatenating `PadA` and `PadB` to the ends of the respective clients' DH public keys is to obfuscate the fact that both clients are engaging in the MSE protocol; passive eavesdroppers would be able to more easily identify the protocol if a fixed length header was used instead. These two steps enable the clients to establish a shared Diffie-Helman secret S.

The information sent by Client A in the third step allows Client B to identify the appropriate torrent (via the obfuscated hash of the stream key), in the case where Client B is seeding multiple torrents. The encrypted verification constant allows Client B to verify that Client A agrees on the shared secret and stream key, and `crypto_provide` provides Client B with a list of encryption methods to use. Currently, all known implementations provide both plaintext and RC4 encryption as options; in the future, perhaps more will be provided. Finally, PadC may be used for future extensions to the MSE protocol.

The information sent by Client B in the fourth step includes the same verification constant, a selected encryption method, and a pad PadD. The verification constant allows Client A to verify that the two clients agree on shared secret and stream key; the selected encryption method completes the encryption method negotiation process; and PadD may be used in the future for extensions.

It is also of note that, because RC4 encryption uses a keystream, communication of the encrypted verification constant helps to protect against MITM replay attacks.

iii. Fallback

Because many clients will not support PE/MSE, clients must be prepared to "fallback" to the original unencrypted protocol. Determining whether or not a peer implements protocol encryption can be done with little overhead:

Seeders who receive requests can check the first several characters of the first message. The unencrypted protocol always begins with the ASCII encoded string `\19BitTorrent Protocol`.

Initiators who send requests can try to send the beginning of the encryption handshake, and get rejected. Afterwards, they will know to send only unencrypted messages.

V. System Evaluation

We implemented PE for WebTorrent according to the above specification. Through our testing, we found that WebTorrent clients with our implementation are able to communicate with each other, and also with other major torrent clients implementing PE (such as μTorrent).

As stated before, the goal of PE/MSE is to provide fast encryption and make BitTorrent traffic more difficult to detect[4]. However, in the years since the PE/MSE protocol was originally designed, further work has demonstrated key weaknesses and vulnerabilities. We evaluate how well PE/MSE is able to meet its goals given the current state of knowledge.

i. Protocol Detectability

The main goal of PE/MSE encryption is to make BitTorrent traffic undetectable to eavesdroppers such as ISP's. ISP's generally detect BitTorrent traffic either by looking for the handshake protocol and looking for traffic shape (i.e. looking at the lengths of packets sent).

In terms of traffic shape detection, the PE/MSE handshake hides itself very well. Each message is sent with a random-length padding whose length is either not sent or hidden in an encryption. It is generally difficult to identify the PE/MSE handshake alone. However, ISP's are still able to identify BitTorrent traffic using the shape of the rest of the BitTorrent protocol, which often involves large transfers of data in one direction.

Though the encryption methods used (RC4 cipher, SHA-1) have been proven to be insecure, in practice the protocol is still fairly effective against passive eavesdroppers. In theory, with enough computational power, an active attacker certainly could exploit the vulnerabilities in RC4 (Fluhrer attack) and SHA-1 (the SHAppening) for protocol/content identification or MITM attacks. However, seeding sessions are generally not long enough for such attacks to occur, and thus Protocol Encryp-

[4]SourceForge, *Identifying the Message Stream Encryption (MSE) protocol*

tion still provides a reasonable level of security against passive eavesdroppers that are not actively attempting to decode messages. Additionally, the protocol takes basic precautions to mitigate the risks posed by these insecurities: for example, the RC4 cipher discards the first 1024 bytes of the keystream to avoid the Fluhrer attack.

Overall, PE/MSE provides a very weak level of protection from detectability. It is generally used against small ISP's and other passive listeners who do not have the expensive resources to monitor traffic in detail.

ii. Performance

The PE/MSE protocol handshake only needs to be initiated once per session between seeder, as many times as the regular handshake is used. The methods used in the handshake (SHA-1, D-H key exchange) are all relatively lightweight and add little overhead to the protocol. RC4 itself is also a fairly lightweight cipher algorithm, and so even full protocol encryption is not a large overhead for BitTorrent. In practice, we found that the cryptographic handshake added no significant overhead to the time it took to download a file.

VI. Conclusion

Overall, the Protocol Encryption and Message Stream Encryption protocols are fairly weak encryption methods. However,they follow a paradigm of opportunistic encryption and as a result they are fast and provide a first line of defense against ISP detection and throttling. This will be immensely helpful for WebTorrent users who live in areas with non-standard ISP's and in areas with high risk of MITM attacks/eavesdroppers. We hope that our contribution to the WebTorrent client will be helpful for BitTorrent servers and its torrents.

As of 5/11/2018, our implementation is available online in two Github pull requests (here and here) to the open source WebTorrent repository. Because this is a fairly significant change to protocol implementation, the pull request is still currently under review by WebTorrent contributors.

We would like to acknowledge Feross Aboukhadijeh, the creator of WebTorrent, for suggesting this project to us, for providing us with resources and guidance on BitTorrent and WebTorrent, and for reviewing our implementation.

References

[1] *At&t patents system to 'fast-lane' bittorrent traffic*, Aug 2015.

[2] Azureus, *Message stream encryption*, Jan 2006.

[3] Ernesto, *Encrypting bittorrent to take out traffic shapers*, Dec 2015.

[4] SourceForge, *Identifying the message stream encryption (mse) protocol*, Jul 2009.