

TLS Compression Fingerprinting and a Privacy-aware API for TLS

Karthikeyan Bhargavan, INRIA, karthikeyan.bhargavan@inria.fr
Cédric Fournet, Microsoft Research, fournet@microsoft.com
Markulf Kohlweiss, Microsoft Research, markulf@microsoft.com
Alfredo Pironti, INRIA, alfredo.pironti@inria.fr
Pierre-Yves Strub, INRIA, pierre-yves@strub.nu

May 11, 2012

1 Abstract

TLS is by far the most widely used secure communications protocols on the Internet. It has a long history of flaws and attacks, ranging from its cryptographic design to traffic analysis. Interestingly, the standard excludes traffic analysis resistance from the design goals of TLS, while at the same time specifying some mechanisms for hiding the length of encrypted messages. Recently, these features have received increased attention, e.g., in the work of Paterson et al. [2]

We look at the traffic-analysis protection mechanism of a recent verified reference implementation of TLS. The implementation provides an extended API for programmers that allows for the fine grained control of length-hiding features. The programmer specifies length ranges through the API, and the implementation guarantees that nothing besides these ranges is revealed.

During the implementation of the reference implementation we discovered a simple but effective traffic analysis attack that exploits TLS-level compression. TLS first fragments data and then compresses each fragment individually, so each compressed fragment sent on the network has a different size. In our attack, when TLS-level compression is in place we can precisely identify single files being transferred over TLS, by looking at their compression patterns.

2 TLS Compression Fingerprints

It is well known that traffic analysis allows to identify individual web pages [3], and that compression leaks some information about the plaintext [1]. However, the TLS practice of compressing each fragment before encryption enables a powerful compression-attack: without length-hiding mitigation (e.g. when using TLS 1.0, or any stream cipher with any version) one can learn information about individual files a web server is sending to a user. In our experiments just looking at the length of few (typically less than 10) compress-then-encrypt fragments is sufficient to fingerprint the file. First, we show how to precisely recover the type of the file being sent (e.g. .bmp, .jpg or .mp3). Then, once the file type is known, we show that the lengths of any sequence of few consecutive fragments uniquely fingerprints a specific file in a given set.

Consider two different file formats, e.g. .mp3 and .jpg, where the first has a header, and the second a trailer. Now, it is likely that the first (last) TLS fragments will contain such header (trailer). The header (trailer) will compress to a peculiar size, that depends on the file type (in practice, our experiments show that the particular contents of the header do not affect compression size significantly), giving a fingerprint of the file type being transferred.

Once the file type is known, a specific file from a given set can be usually recognized by observing few consecutive fragments. In our dataset, composed of 10 files of each type, all of the same original size and so

requiring the same number of fragments (145), only 3 consecutive fragments need to be observed in order to distinguish a specific .bmp file. This is not surprising, because compression is expected to operate very well on the uncompressed .bmp format, giving expressive fingerprints. More surprisingly, only 7 consecutive fragments need to be observed to fingerprint an .mp3 file. Apparently, the TLS compression algorithm is able to exploit the little redundancy left by the .mp3 compression algorithm, making the compression attack work effectively. In appendix A, figure 1 plots the wire size of each compressed-then-encrypted fragment for the 10 mp3 files in the dataset. The .jpg files give the worst results, requiring 144 fragments (almost the full file transfer) to be observed before getting a fingerprint; note however that a fingerprint can be obtained for every file, and no file was indistinguishable from any other.

We believe the impact of this attack is quite relevant, as web browsers such as Google Chrome and web servers backed up by OpenSSL will negotiate to use compression by default.

3 A Privacy-aware API for TLS

The TLS standard specifies the messages exchanged over the network, but not its application interface. Since this is critical for using TLS securely, we design our own API, with an emphasis on precision—our API is similar to those provided by popular implementations, but gives more control to the application, so that we can express more precise security and privacy properties:

In our implementation, we chose to support only the trivial, NULL compression algorithm. Our TLS implementation offers a padding-based length-hiding API to applications. We index plaintexts with a *range $m..n$* where $0 \leq m \leq n$. The idea is that an observer of the encrypted connection will only be able to estimate that the plaintext length falls between the given range (or a larger one, but crucially not a smaller one), the real length of the plaintext remaining secret.

For example, consider a website that provides a variable-length personalized home page (between 100 and 500 bytes long) after the user logs in. It could be possible to identify specific users by looking at the length of the returned home page. With our API, the website may give home pages the indexed abstract type `(;(100,500))appdata`, hence, requesting for the actual length to be hidden. The range (100,500) is treated as public, and suffices to determine fragmentation and padding. Using a ciphersuite of the form `...WITH_AES_128_CBC_SHA`, for instance, any value of this type can be uniformly split, MACed, encoded, and encrypted into two fragments of 36 blocks each. In TLS 1.1, the maximal net padding size for any given fragment is limited to 255 bytes, so hiding the text length in the example above may require sending several padded fragments.

Our implementation follows a simple fragmentation and padding algorithm: given a plaintext size range $m..n$, we compute the minimal number of fragments mf needed to include up to $n - m$ bytes of additional padding. Then, if $mf = 1$ and n fits in a single fragment, we can simply send out a single fragment. Otherwise, we compute a sub-range $m'..n'$ that fits in a single fragment, and split the application data into a fragment f within the sub-range, and the remainder. We send f and we iterate on the remainder. Our splitting function crucially reserves at least one byte of application data for the last fragment, so that the receiving application will not show any observable behavior until the last fragment is received.

The actual sizes of the plaintext and the padding added to obtain an encoded fragment remain provably secret and do not influence the size of the fragment on the wire.

On the receiving end of a connection, the same length-hiding specification applies: as wire packets are processed, the application is notified of the arrival of “some” bytes, with a public range size that depends only on the ciphersuite and the size of those wire packets. Continuing with our example, the receiver would get two data chunks, each with a size range of 0..250.

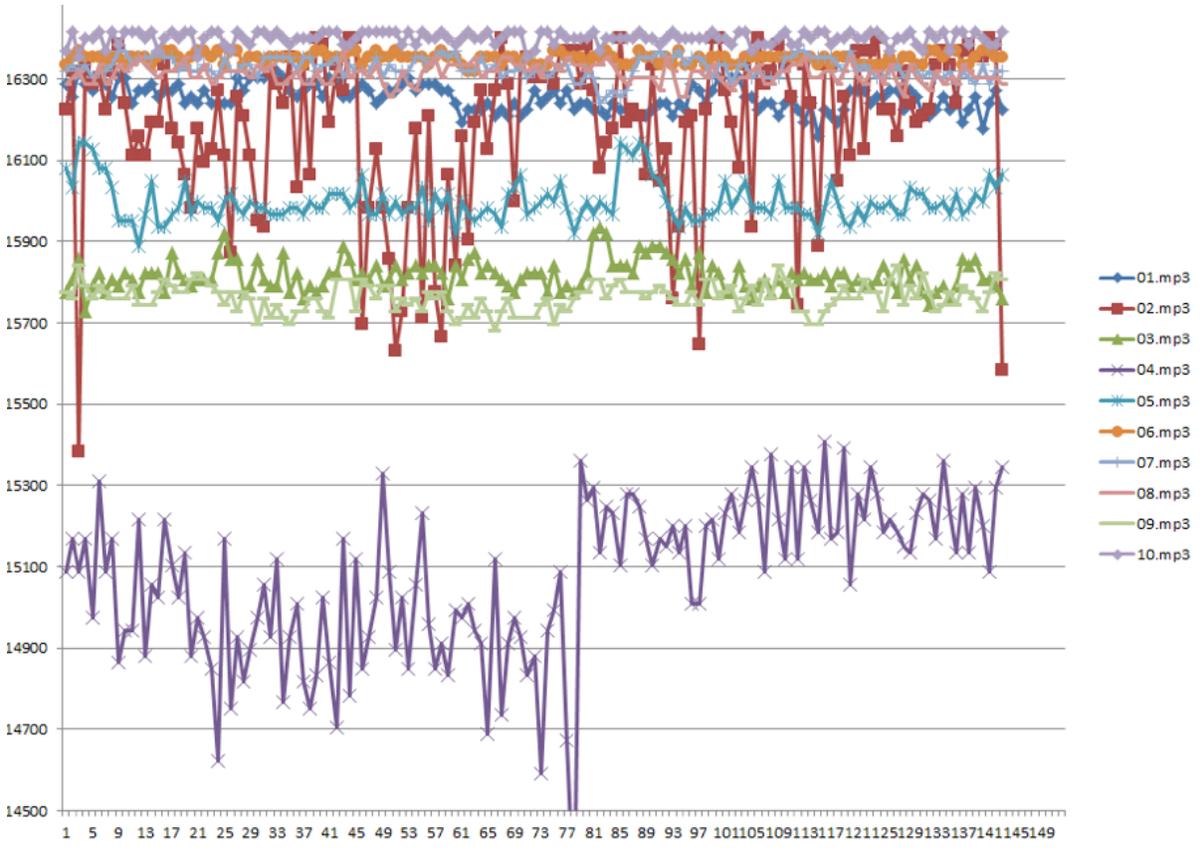


Figure 1: Wire size of compressed-then-encrypted TLS fragments for 10 different mp3 files. X-axis is the fragment number for each file; Y-axis is the size, in bytes, of the compressed-then-encrypted fragment as sent on the network. The very first fragment of each file (fingerprinting the file type) is not plotted because out of range in this graph.

A Exploiting the Compression Attack on mp3 Files

References

- [1] J. Kelsey. Compression and information leakage of plaintext. In *Fast Software Encryption*, pages 95–102. IACR, 2002.
- [2] K. G. Paterson, T. Ristenpart, and T. Shrimpton. Tag size does matter: Attacks and proofs for the TLS record protocol. In *ASIACRYPT 2011*, pages 372–389, 2011.
- [3] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu. Statistical identification of encrypted web browsing traffic. In *IEEE Symposium on Security and Privacy*, 2002.