Simple Opportunistic Encryption

Andrea Bittau Michael Hamburg Mark Handley
David Mazières Dan Boneh

January 15, 2014

Abstract

Network traffic encryption is becoming a requirement, not an option. Enabling encryption will be a communal effort so a solution that gives partial benefits until fully deployed is needed. A solution that requires little changes to existing infrastructure will also help as it can be quickly deployed to give immediate short-term benefits. We argue that tcpcrypt, a TCP option for opportunistic encryption is the path of least-resistance for a solution against large-scale traffic encryption. Tcpcrypt requires no changes to applications, is compatible with existing networks (works with NATs), and just works by default. It is high performance, so it can be deployed on servers without much concern. tcpcrypt attempts to maximize security for any given setting. By default, it will protect against passive eavesdropping, and also allows detecting large scale interception. With authentication, tcpcrypt can provide full security against active attackers and so it is a complete solution both for the short-term and long-term.

1 Introduction

Cases of large-scale network interception are being uncovered [4] making the public concerned about data privacy. Unfortunately the Internet is clear-text by default (TCP) and only special traffic is encrypted (*e.g.,* with TLS [7]). This is due for reversal, and all traffic will be encrypted by default in future.

The first question in finding a solution is: *what is the minimum change we need today to enable network encryption?* Tackling the problem at the network layer, we could enable IPSec [6] globally, but that comes at the cost of incompatibilities with NATs. Alternatively, we could tackle the problem at the application layer and use TLS in all our applications, though we'd be force to rewrite them. Clearly the choice of layer is important, and a promising new approach might be at the transport layer, which is compatible with NATs and requires no application changes.

The second question is: *what is the best security that can be offered in any given setting?* When incrementally deploying a security solution, if the remote end does not yet support it, no security at all may be possible. When browsing a public web site that does not require authentication, protection against passive eavesdropping is possible with an anonymous key exchange. When checking e-mail, mutual authentication is possible using a weak-password and PAKE [3]. The level of security, and more precisely authentication choice, depends on the particular application. This suggests that authentication should be left to the application, while privacy and encryption can be implemented at a lower layer as they are generic, application agnostic, concepts.

Based on the above observations, the choice of which layer is responsible for which aspect of security is critical. In the next sections we discuss in more depth the benefits of each layer. We then discuss tcpcrypt [2], a TCP option for opportunistic encryption that can be used to stop large-scale monitoring.

2 Privacy and Integrity at the transport layer

Privacy and integrity are generic concepts that applications want and do not necessarily care *how* they are done. This allows for their implementation at lower levels. The benefits of performing them at the transport layer are:

- No changes to applications. This would be difficult at application layer.
- Compatible with NATs. This would be difficult at IP layer.
- Can piggyback on the existing 3-way TCP handshake for negotiation. This allows for low latency connections because packets in addition to the the 3-way handshake may not be needed. Being able to signal encryption in the SYN gives an opportunity to gracefully fall back to vanilla TCP if encryption is not supported. All of these aspects cannot be implemented in the application layer in a backward compatible way.
- Natural granularity for authentication and security statements. *E.g.,* "this *connection* run by this user performed this action". As opposed to, if operating at the network layer, "this *packet* sent by this user".

3 Authentication at the application layer

Traditionally, network protocols dealt with all security aspects in a single layer. For example, TLS and IPSec both provide privacy, integrity and authentication, all in a single layer. It may be possible to split these aspects and implement them in the layer where they fit most naturally. For example, we already argued that the transport layer is a convenient place to implement privacy and integrity. Authentication, instead, is inherently an application-specific concept. The application may do no authentication, or it can use any mechanism like a password, cookie, certificate or something else. In fact, today, double authentication often occurs: *e.g.,* at the transport layer with TLS using certificates, and at the application layer using passwords. It is possible, instead, for the application to mutually authenticate a connection using passwords only (*e.g.,* using PAKE), not requiring any additional, lower-level, authentication steps (*e.g.,* using certificates). We therefore argue that authentication should be implemented at the application layer, whereas privacy and integrity should be implemented at the transport layer because they are generic, non-application specific, concepts.

Because encryption and authentication occur at different layers, a mechanism to link the two is needed. In tcpcrypt, we introduce the concept of a *Session ID*. The Session ID is a hash that captures the key negotiation. Each connection yields a Session ID, and if equal on both ends, no man-in-the-middle attack occurred. This mechanism alone makes it possible to detect large-scale interception after the fact, even with no authentication. One simply logs Session IDs and compares them after the fact.

It is simple for applications to integrate transport security with their existing authentication mechanisms. For example, when comparing passwords, the application can include the Session ID in the password hash to make sure that both the connection is secure, and that the user is who he claims to be.

This design maximizes security for any given setting. With no authentication, applications are by default secure against passive eavesdropping, and they can also detect man-in-the-middle attacks after the fact. With any authentication (*e.g.,* passwords-only, certificates), one is protected against active attacks too.

4 tcpcrypt

tcpcrypt is a TCP option for opportunistic encryption. It has the following main features:

- Anonymous encryption by default.
- Provides hook for application-level authentication (Session ID).
- Provides mechanism to probe whether application is tcpcrypt-aware. This is useful for authentication.
- High performance: high connection rate on servers by pushing expensive cryptographic operations on client; low latency connection set up by piggybacking on TCP's existing handshake. This makes it possible to turn it on by default without too much thought: *i.e.,* it will not slow down the Internet.

5 Deployment strategy

One of the main design goals of tcpcrypt was to make it incrementally deployable. The following time line is how we envisioned tcpcrypt's deployment:

1. **Operating systems to support tcpcrypt.** During this era, most of the traffic will either be plain-text TCP (the other end does not yet support tcpcrypt), or unauthenticated tcpcrypt. This will let the public detect large-scale eavesdropping. It will protect against passive attackers. This shows the advantage of operating at the transport layer: it provides a backward compatible mechanism to signal whether the option is supported or not, and one can gracefully fall back to vanilla TCP if needed. This luxury is not present at other layers.

2. **Applications to support tcpcrypt.** During this era, most traffic will be unauthenticated tcpcrypt, or authenticated (if both ends of the application support tcpcrypt). This will protect against active attackers. Again, this shows the advantage of operating at the transport layer: one can signal (in the SYN) whether the application supports tcpcrypt, something not easily done at other layers.

3. **Integration with other security initiatives.** Once tcpcrypt is deployed, it can be integrated with other security initiatives like DNSSEC [1] and DANE [5] to provide some authentication by default. For example, APIs that connect to hostnames can provide server authentication by default if certificates are held in DNSSEC. This can easily be done using tcpcrypt's Session ID, by having the server sign it. This shows that tcpcrypt has been designed from the start as a complete security solution, and not just a short-term, opportunistic encryption protocol.

6 Lessons

tcpcrypt is not an attempt to reinvent the wheel. All of which is achieved in tcpcrypt can probably be achieved in other layers with some more complex modifications. The transport layer merely looks like the simplest place to apply encryption. In particular, we found that being able to signal tcpcrypt awareness and application awareness in the SYN is critical for backward compatibility. The rest of the tcpcrypt negotiation could in fact occur at the application layer, but why give up extra round trips, when tcpcrypt can complete within the existing 3-way handshake when session caching? Everything seems to point to the transport layer when it comes to convenience.

A new concept introduced by tcpcrypt is the session ID, needed to tie in with application layer authentication. This can be implemented by other layers, but we found that the transport (or connection) most closely resembles a user session, often the granularity needed for authentication, and so the transport seems like a natural place for exposing authentication hooks.

7 Conclusion

Internet traffic will be encrypted by default. There is an urgency to meet this requirement, so a solution that is simple to incrementally deploy is needed. This solution should also be complete enough so that it can be used as the sole security system once used by applications. We believe that the path to least resistance is providing privacy and integrity by default at the transport layer. This requires no changes to applications and is compatible with existing networks (NATs). We built tcpcrypt to satisfy this. Applications can use tcpcrypt's Session ID in their existing authentication logic to protect against man-in-the-middle attacks. tcpcrypt is complete enough that it will meet short-term opportunistic encryption needs, as well as longer-term full security requirements needed by applications.

References

[1] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033 (Proposed Standard), Mar. 2005. Updated by RFC 6014.

[2] A. Bittau, M. Hamburg, M. Handley, D. Mazières, and D. Boneh. The case for ubiquitous transport-level encryption. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 26–26, Berkeley, CA, USA, 2010. USENIX Association.

[3] D. Boneh and V. Shoup. A graduate course in applied cryptography. Version 0.1, from `http://cryptobook.net`, 2008.

[4] CNN. Latest NSA leaks point finger at high-tech eavesdropping hub in UK. `http://www.cnn.com/2013/12/20/world/europe/nsa-leaks-uk/`.

[5] P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698 (proposed standard), IETF, 2012. `http://tools.ietf.org/html/rfc6698`.

[6] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, 1998.

[7] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley Professional, 2000.