Some Security Concerns Regarding PPP–EAP–TLS[*]

Jonathan Herzog
jherzog@mitre.org
The MITRE Corporation
MP 00B0000019

August 1, 2000

1 Introduction

The *Point–to–Point Protocol* (PPP) [9] is an IETF standard for multiplexing the datagrams of multiple network protocols over a point–to–point link. Like several other standards, PPP has undergone constant revision and evolution since its inception, including the addition of cryptographic security mechanisms. Although mechanisms for authentication and confidentiality were present in the original drafts of PPP, these mechanisms have been significantly revised and expanded in later documents. In particular, RFC 2716 [1] proposes a way in which the *Transport Layer Security* (TLS) protocol [3] can be used to secure a PPP connection.

In this note we examine this RFC and note some unresolved issues and concerns. We will start by briefly describing some of the beginning phases of a PPP connection, particularly those related to security. We will then turn to TLS, touching on those aspects of the protocol relevant to its use in PPP. We will then detail some concerns regarding the suggested use of TLS, and conclude with some suggestions.

2 PPP

The Point–to–Point Protocol is a data link layer (Layer 2) protocol, lying between the physical layer (e.g., Ethernet, ATM) and the network layer (e.g., IP). It is responsible for configuring a single connection, framing packets, fragmentation and reassembly, and multiplexing the datagrams of higher protocols. The connection is assumed to be between a single pair of peers, such as a modem

[*]©2000 The MITRE Corporation. All Rights Reserved.

connection or dedicated link. PPP is not responsible for reliability, addressing, or routing.

During initialization the connection passes through several phases, only some of which will be detailed here. At the beginning of the protocol the physical connection is assumed to be in the *Dead* state. By some external mechanism, the PPP implementation is then informed that the connection is *Alive* and ready to be configured.

The first phase of the PPP life-cycle, the *Link Establishment Phase*, is regulated by a sub–protocol known as the *Link Control Protocol* (LCP). Various aspects of the connection are negotiated during this phase, such as encapsulation format options, packet sizes, and whether the connection is to be authenticated before used.

If so, and once the connection has been established by LCP, the connection can progress into an optional *Authentication Phase*. During this phase, the *Extensible Authentication Protocol* (EAP) [2] is used to negotiate the manner of authentication. As its name indicates, EAP can choose between any number of authentication mechanisms, several of which are defined separately.

Because PPP is a peer–to–peer protocol, it is assumed that authentication will be requested by each party that desires it. The party that makes the request (called the "authenticator") will actually send one or more messages to the peer, each specifying an individual authentication method. The peer then chooses one of these authentication methods and completes its side of the protocol. Note that the peer authenticates itself to the *authenticator*; it is the authenticator that will be sure of the identity of the peer. If the peer wishes for reciprocal authentication, it is responsible for making its own authentication requests and assuming the role of authenticator in those runs of EAP. It is explicitly stated that this could result in two authentication methods being used, each providing authentication in a different direction [2, "Security Considerations"].

Once the Authentication Phase is finished, either party can then request an optional *Encryption Phase* in which the *Encryption Control Protocol* (ECP) [7] is used to negotiate an encryption algorithm. At this point in time (August 1, 2000), only two standards for encryption algorithms have reached RFC status: DES [10] and triple–DES [6]. Both assume that a shared secret key has already been exchanged by some other method.

After the Encryption phase, there are several other optional phases through which the PPP connection can pass; none are related to security. When these are completed, the connection will enter the *Network Phase*, where the *Network Control Protocol* (NCP) is used to establish virtual connections for higher–level protocols.

3 Transport Layer Security

Like PPP, TLS also uses two sub-protocols to provide security:

- The *Handshake protocol* is a sequence of messages whereby the two parties can:

  - authenticate one another, if desired, and
  - negotiate two items:
    * a shared secret, and
    * a mutually acceptable cryptosuite (set of cryptographic algorithms).

- The *Record protocol* is a set of message formats in which the messages of the Handshake protocol and of higher–level protocols are sent. The Record protocol can encrypt these messages using the cryptosuite negotiated in the Handshake protocol, and with keys derived from the shared secret and random numbers.

The TLS protocol is quite flexible, and can handle a large number of options. We will not reproduce the inner workings of the protocol here, however; TLS has been thoroughly examined using a variety of methods ([8, 5, 11]). We do, however, note several things that will prove to be of relevance in our discussion:

- First, the Handshake and Record sub–protocols are interdependent. A TLS connection always begins with the establishment of a Record protocol connection, albeit one without any cryptographic security. On top of this connection a run of the Handshake protocol then commences, wherein the cryptosuite and shared secret are negotiated. Just before the Handshake protocol completes, the underlying Record protocol begins protection of its contents (using the negotiated values.) Once the Record protocol has become a secure tunnel, the Handshake protocol completes by sending its last messages. The secure tunnel is then available for the use of higher–level protocols.

  Hence, not only does the Record protocol require initialization by the Handshake protocol, but the Handshake protocol requires the secure tunnel provided by the Record protocol.

- The set of cryptosuites available to the Record protocol is fixed and explicitly enumerated in the TLS specification ([3]). Each suite specifies

  - an authentication/key exchange method (e.g., authenticated Diffie–Hellman),
  - a public key signature algorithm,
  - a bulk encryption algorithm, and
  - a keyed MAC algorithm.

  The first two algorithms are used only in the Handshake protocol, for authentication. The second two—the bulk encryption and the keyed MAC—are for use in the Record protocol and protect the communications of higher–level protocols. (The keys for these two algorithms are derived from the shared secret.)

One cryptosuite is *Null*, where no algorithms will be used for any of the above purposes. This is the cryptosuite used at the beginning of a TLS connection, before another suite is negotiated by the Handshake protocol. All non–*Null* suites specify algorithms for at least key exchange, public–key signatures, and MAC algorithms, and many also specify an algorithm for bulk encryption.

This implies two interesting facts. First, all non–*Null* cryptosuites require the use of a MAC algorithm. Second, all non–*Null* cryptosuites require the use of a public key signature. Both of these facts will become relevant in our discussion of PPP–EAP–TLS.

- TLS is derived from SSL [4], which is designed to protect web connections. Because of this, TLS is designed for client–server communications: the initiator of the protocol is designated the "client," the responder is designated the "server," and the protocol places different requirements upon them. In particular, the client and the server have different authentication burdens: the server is required to authenticate itself whenever the client does, but not vice–versa. The server can of course *request* that the client authenticate itself, but the client is allowed to refuse.[1]

4 TLS in PPP

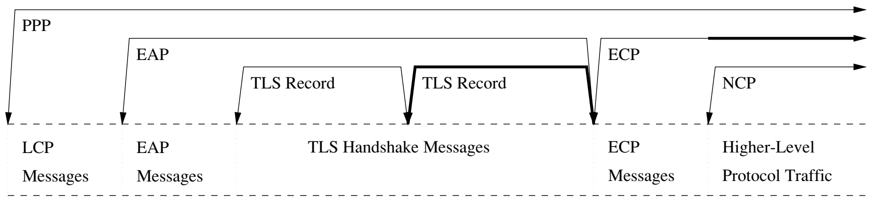
In RFC 2716 [1], it is recommended that TLS be added to the authentication mechanisms available for use in PPP, and suggests a way in which TLS can be embedded within EAP. In fact, this embedding is very simple: TLS already specifies the formats that its messages take, so these messages are simply sent as the data of PPP packets.

In PPP–EAP–TLS, TLS is executed only until the end of the first run of the Handshake protocol. At that point, the authentication has finished, the shared secret has been exchanged, and the underlying Record protocol has been initialized and is providing a secure tunnel. In PPP, however, the authentication phase and the encryption phase are distinct; all that PPP requires from TLS is authentication and a shared secret. Hence, the secure tunnel provided by TLS is abandoned, and the shared secret stored for use by ECP. (Figure 1 shows the relationship between the TLS protocol and PPP connection establishment.)

Although PPP does not guarantee reliability, this does not seem to compromise the security of TLS.[2] What could be troubling, however, is the fact that PPP is a peer-to-peer connection while TLS is based on a client-server model. Except for PPP–EAP–TLS, all EAP protocols provide authentication in one direction only. Two simultaneous authentication protocols, therefore, would not conflict. TLS, however, can provide mutual authentication, and it is

[1]The server can then refuse to continue, but to do so is actually considered an error rather than a resolution of the protocol.

[2]In fact, since TLS is designed to be secure against active attacks (i.e., those that disrupt or forge messages), vulnerability to packet loss would indicate a serious design flaw.



Bottom "pipe" contains actual messages sent during establishment of PPP connection (from left to right) where EAP–TLS is used in Authentication phase and an unspecified protocol is used by ECP during the Encryption Phase. Tabs on top represent the nesting of protocols over the same period of time. Bold lines represent use of cryptographic algorithms.

Figure 1: Protocols and Messages in PPP Establishment, using EAP–TLS

unclear what should happen when it is requested by both sides. The desirable behavior, of course, would be for both requests to collapse into a single run of the protocol, using mutual authentication. However, the specification seems to allow the possibility of two independent runs of TLS. If this happens, then at their completion the PPP connection will have two separate shared secrets and two (possibly distinct) cryptosuites. PPP–EAP–TLS specifies how one shared secret is used to generate the keys necessary for ECP, but not how to pick which of the two shared secrets or cryptosuites are to be used.

Another related oddity is that the role of the authenticator in PPP–EAP–TLS does not map directly to the role of authenticator in other authentication protocols. Elsewhere, the authenticator is the one *receiving* the authentication; the peer proves its identity to the authenticator. But in PPP–EAP–TLS, the authenticator takes on the role of server, and as mentioned above, it is possible for authentication in TLS to be server–to–client *only*. And while in PPP–EAP–TLS (as opposed to TLS) the client is required to authenticate itself when the server requests such, the server is not required to request. So, the authenticator can now request an authentication mechanism by which it is authenticated to the peer but *not* vice versa.

Also, there appears to be a large disconnect between PPP–EAP–TLS and ECP. During PPP–EAP–TLS, the client and server negotiate a set of cryptographic algorithms, and according to the specification, any subsequent run of ECP is required negotiate those same algorithms. This raises several potential sources of concern, however.

- First, all cryptosuites defined in TLS require that a keyed MAC be added whenever encryption is used. However, none of the encryption algorithms defined for ECP have any type of integrity check. It is therefore impossible for ECP to negotiate the exact cryptosuite which was agreed upon in TLS.This is easily fixed by defining ECP protocols to match the TLS cryptosuites, but no such protocols are defined at this time.

- Even if ECP performs the most natural action and uses the encryption algorithm from the TLS cryptosuite, problems remain. ECP is designed to be extensible; theoretically, ECP can negotiate any algorithm mutually acceptable to both parties. However, TLS is not similarly extensible; the list of cryptosuites available to TLS are enumerated in the specifications, and to use any cryptosuites not on that list would mean either revising or deviating from the standard. In other words, compliant implementations of ECP will no longer be able to use private or proprietary algorithms when EAP–TLS is used for authentication.

- Lastly, this requirement—that ECP use the algorithm negotiated in TLS—potentially undermines the basic design of PPP. In the original design of PPP, authentication and encryption are negotiated and enacted in separate phases. If TLS is used as proposed, however, the negotiation of encryption has been transplanted from ECP to the earlier phase of EAP. While this may not necessarily introduce any weaknesses, it does significantly change the design of PPP.

5 Summary and Suggestions

The use of TLS as an authentication mechanism is an excellent idea. However, some minor issues should be addressed before it should be used. In particular, it would be advisable for the authenticator to play the role of client (rather than server), to bring EAP–TLS into concordance with the way those roles are used in other EAP protocols.

Also, the specification should resolve the ambiguity that results when both sides request TLS. It would be simpler to require that both requests be merged into a single run of the protocol, but if the specification wishes to allow simultaneous runs it should describe how to choose between the resulting secrets and cryptosuites.

Lastly, EAP–TLS raises three concerns about ECP:

- First, EAP–TLS relocates cryptosuite negotiation from ECP to EAP. While this may be a change for the better, it is still a significant alteration to the design of PPP and may deserve further consideration.

- Second, it highlights the fact that no defined ECP protocol includes a keyed MAC for integrity. While this is not a problem caused by EAP–TLS, it is still serious enough to warrant note.

- Third, it prohibits ECP from using any cryptosuite that cannot be negotiated in TLS, including all private algorithms.

The solution to the last two concerns above would seem simple and two-fold. First, ECP cryptosuites that include integrity protection should be defined. Ideally, these would include all the cryptosuites defined in the TLS specification.

Second, the list of cryptosuites available to TLS should be expanded to include all ECP cryptosuites. The TLS specifications explicitly allows subsequent s-tandards to expand the list of cryptosuites; the EAP–TLS specification should map ECP cryptosuites to a set of TLS cryptosuites which may only be available during EAP–TLS.

References

[1] B. Aboba and D. Simon. PPP EAP TLS authentication protocol. RFC 2716, October 1999.

[2] L. Blunk. PPP extensible authentication protocol. RFC 2284, March 1998.

[3] T. Dierks and C. Allen. The TLS protocol. RFC 2246, January 1999.

[4] Alan Frier, Philip Karlton, and Paul Kocher. The SSL 3.0 protocol. Internet Draft, November 1996.

[5] Jonathan Herzog, Laura Feinstein, and Joshua Guttman. A strand space analysis of TLS 1.0. MITRE Document MTR 0B0000011, 2000.

[6] H. Jummert. The PPP triple–DES encryption protocol. RFC 2420, September 1998.

[7] G. Meyer. The PPP encryption control protocol. RFC 1968, June 1996.

[8] Lawrence C. Paulson. Inductive analysis of the Internet protocol TLS. *ACM Transactions on Computer and System Security*, 1999.

[9] W. Simpson. The point–to–point protocol. RFC 1661, July 1994.

[10] K Sklower and G. Meyer. The PPP DES encryption protocol, version 2. RFC 2419, September 1998.

[11] D. Wagner and B. Schneier. Analysis of the SSL 3.0 protocol. In *Proceedings, Second USENIX Workshop on Electronic Commerce*, pages 29–40, 1996. Available at http://www.counterpane.com/ssl.html.