Extensible Authentication Protocol

The Extensible Authentication Protocol (EAP) is best considered as a framework for transporting authentication protocols, rather than as an authentication protocol itself. EAP can be used for authenticating dial-up and VPN connections, and also LAN ports in conjuction with IEEE 802.1X.

The basics

In EAP [1], the party demanding authentication is called the authenticator and the party being authenticated is called the supplicant. The EAP protocol defines four types of packet: *request*, *response*, *success* and *failure*. Request packets are issued by the authenticator, and solicit a response packet from the supplicant. Any number of request-response exchanges are permitted to complete the authentication. If the authentication is successful, a success packet is sent to the supplicant; if not, a failure packet is sent.

The basic EAP packet format is simple. A *type* field indicates the type of packet, such as a response or a request. An *Identifier* field is used to match requests and responses. Response and request packets have two further fields. The first, confusingly called *type*, indicates the type of data being transported (such as an authentication protocol), and the second, *type-data*, consists of that data. Note that EAP *method* is synonymous with type, and both are used frequently.

The EAP specification defines three 'basic' authentication EAP types *(MD5-Challenge, OTP and GTC*) and three non-authentication types (*Identity, Nak,* and *Notification*). The three 'basic' authentication types are not considered secure for typical use, particularly in wireless environments, and consequently other types, which will be discussed later, should be used. The Identity type is used by the authenticator to request the user name claimed by the supplicant, and is typically the first packet transmitted. The Nak type is used by the peer to indicate that a type proposed by the authenticator is unacceptable (for example, the authenticator has proposed an authentication protocol that is unsupported by the peer, or policy forbids its use). If the supplicant Naks a proposed type, the authenticator may choose to try another, thereby allowing supplicant and authenticator to negotiate a mutually acceptable authentication protocol. The Notification type, which is rarely used, returns a message that must be displayed to the user.

Finally, EAP permits *pass-through* authentication. Pass-through authentication allows the authenticator to forward all responses, using the RADIUS protocol, to a remote *EAP server*. This server assumes the role of the authenticator for the remainder of the EAP session. The main advantage of this is that it permits centralised management of authentication. Another advantage is that the authenticator does not need to support the type negotiated by the peer and the EAP server. An example EAP exchange is shown in figure 1 below. The peer refuses OTP authentication, but agrees to MD5-Challenge and is authenticated.

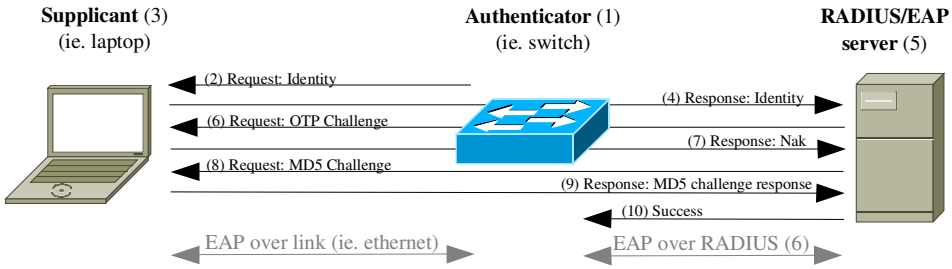

Figure 1: an example authentication

The following diagram shows an extract of the output from a supplicant (*wpa_supplicant* [2]*)* in the above scenario (note the use of 'method' rather than 'type'):

```
EAP: Received EAP-Request method=1 id=2                          (step 2)
EAP: EAP-Request Identity data – hexdump_ascii(len=38):
    00 6e 65 74 77 6f 72 6b 69 64 3d 65 64 75 72 6f   _networkid=eduro
    61 6d 2c 6e 61 73 69 64 3d 43 43 57 33 2c 70 6f   am,nasid=CCW3,po
    72 74 69 64 3d 30                                 rtid=0
EAP: using real identity – hexdump_ascii(len=9):                (step 4)
    74 65 73 74 2d 75 73 65 72                        test-user
EAP: Received EAP-Request method=21 id=3                         (step 6)
EAP: Building EAP-Nak (requested type 21 not allowed)            (step 7)
EAP: Received EAP-Request method=4 id=4                          (step 8)
EAP-MD5: generating Challenge Response                           (step 9)
EAP: Received EAP-Success
```

EAPs types TLS, TTLS and PEAP

As previously mentioned, the 'basic' authentication types should not be used. In particular, they do not provide the keying material required for IEEE 802.11 encryption, Consequently, a number of other more secure types have been developed. Of these, only three have been widely implemented: TLS [3], TTLS [4], and PEAP [5].

The *TLS* type is based on the Transport Layer Security (TLS) [6] protocol, which uses public key cryptography for authentication and production of keys that can be used to encrypt data. TLS is also the protocol used for securing HTTPS and so they work in a similar way. The main difference is that HTTPS is transported over TCP, whereas EAP TLS is transported over the EAP session between the supplicant and EAP server. Similarly to HTTPS, the supplicant authenticates the server's certificate using a locally stored root certificate. However, unlike most HTTPS transactions, EAP TLS uses a user certificate to authenticate the supplicant to the server.

Consequently, TLS can only be used by organisations with a Certificate Authority (CA) that issues user certificates; as such, although it offers excellent security, it is not widely deployed. Consequently, two further EAP types, *Protected EAP* (PEAP*)* and *Tunneled TLS* (TTLS) cleverly work around this problem. Both of these types also use TLS, but avoid the need for user certificates by using a second authentication protocol that is protected by the TLS exchange between the supplicant and the server. This is very similar to conventional HTTPS authentication, where the user's plain-text credentials are protected by TLS. The main difference between the types is that PEAP can only protect other EAP types, whereas TTLS can protect almost any authentication protocol. An overview of the protocol layering, which can be fairly confusing, is shown in Figure 2 below (the RADIUS, EAP and user database servers are shown as separate entities to improve clarity; in practice, they will probably be more integrated).

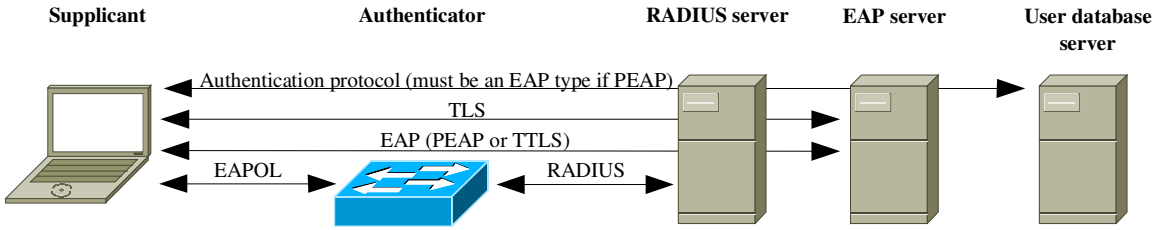

Figure 2: protocol layering in EAPs PEAP and TTLS.

Considerations when implementing EAP

Determine how your passwords are stored in your user database(s), and which operating systems you need to support. These factors will largely determine which type(s) you deploy. Windows' supplicant only supports the *MSCHAPv2* EAP type within PEAP, requiring that the EAP server is able to authenticate MSCHAPv2 credentials. If this is not the case, TTLS should be considered. Windows' supplicant also integrates poorly with Windows domain authentication. While this is not typically required for casual network access, it can make it unsuitable for authenticating conventional desktop terminals; “Longhorn” will rectify this shortcoming. If Windows' supplicant is deemed unsuitable, a third-party supplicant will be necessary. The UKERNA IEEE 802.1X factsheet contains a comparison of supplicants.

Ensure that your RADIUS server supports EAP, your chosen EAP type(s), and authentication against your user database(s). A variety of products are known to be used by JANET sites, including Microsoft's IAS, Open Solution's Radiator, Funk's Steelbelted RADIUS, Cisco's ACS and the open-source FreeRADIUS.

Consider how you will acquire a certificate for the RADIUS server. If you acquire certificates from a vendor, as opposed to a self-signed certificate, ensure that you configure the supplicants to perform name-based constraints on the certificate's server name attribute. This prevents a rogue EAP server from masquerading as the authorised EAP server by acquiring a certificate from the same vendor.

Bibliography

1: B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, Ed., RFC3748 - Extensible Authentication Protocol (EAP) , 2004

2: Hostap development team, WPA Supplicant, , http://hostap.epitest.fi/wpa_supplicant/

3: B. Aboba, D. Simon, RFC2716 - PPP EAP TLS Authentication Protocol, 1999

4: Paul Funk, Simon Blake-Wilson, EAP Tunneled TLS Authentication Protocol, 2002

5: Ashwin Palekar, Dan Simon, Glen Zorn, Joe Salowey, Hao Zhou, S. Josefsson, Protected EAP Protocol (PEAP) Version 2, 2003

6: T. Dierks, C. Allen, The TLS Protocol Version 1.0 , 1999