Laboratory Report
Rohit Raj

The EAP-TLSv1.3 connection behavior

1 Introduction

EAP-TLS is widely considered as one of strongest EAP methods for wireless enterprise authentication. It uses Transport Layer Security protocol and unlike other implementations, it requires x509 client-side certificates. The latest version of Transport Layer Security (TLS) 1.3 was released in August 2018. This release has brought some exciting new features along with improved latency and security during connections. These new features affects the way current EAP-TLS authentications are handled. Our goal of this experiment was to analyze the behavior of different EAP-TLS 1.3 implementations(namely *HostApd* and *FreeRadius*).

1.1 Experimental Setup

The setup comprised of the following three EAP components:

- *Supplicant* : The client attempting EAP-TLS connections was a Ubuntu 18.03 machine. For making the EAP-TLS connections, *wpa-supplicant* was used.
- *Authenticator* : *D-LINK DWL-6600AP* router acted as an authenticator and this access point was configured for EAP-TLS communication along with the address of Authentication server on one of its SSID.
- *Authentication Server* : Our choice of Authentication server were popular flavors of RADIUS server such as *FreeRadius* and *HostApd*. The server itself resided on Cloud and received the RADIUS packets from our Authenticator on port 1802.

As per the standards of TLS1.3, Elliptic Curve certificates were generated using *prime256v1* NIST curve. A single root CA signed both the server and client certificates. Finally, the rootCA certificate was installed in both the supplicant and Authentication Server. The default key share algorithm used by wpa_supplicant in the *client_hello* is x25519. As the default Pairwise-Master Key Security Association (PMKSA) duration is 43200 seconds, to enable faster wireless renegotiation, we reduced the *dot11RSNAConfigPMKLifetime* property to 10 seconds.

2 HostApd TLS 1.3 behavior

1. The default key share algorithm of HostApd is x25519. As a result, as per the specifications mentioned in the **RFC 8446** of TLS1.3, the key negotiation is successful in 1-RTT. It means that the server replies with *Server_Hello* along with some encrypted [Application Data] used by the supplicant for integrity and authentication checks.

2. *wpa_supplicant* offers a total of 31 supported cipher suites as part of Client Hello request. The HostApd selects "TLS_AES_256_GCM_SHA384 (0x1302)" as its preferred choice.

3. The *"Version"* parameter in all the hello requests is set as "TLS 1.2 (0x0303)" as mentioned in Section 4.1.2 & Apendix D of RFC 8446 for backward compatibility.

4. To re-establish the identities without the use of certificates after the PMKSA duration has expired, hostApd and wpa_supplicant negotiated a *Pre-Shared Key*.

5. For subsequent negotiations, the "client_hello" requests had a *pre_shared_key* extension that contained the 32 byte PSK Identity. The total time between EAP-request and EAP-success in this scenario was found to be averaging 0.1414 sec where it was 0.1903 seconds without a pre-shared key.

6. After the successful renegotiation and EAP success state, there were a total of 4 EAPOL messages for PMK and 2 EAPOL messages for GMK

7. Talk about PMK and GMK. Also about the hello retry request'sha as per original rfc guideline

3 Burning Questions!

1. Is there no flexibility from the Authentication Server's end in the cipher spec?

2. Will the PSK be common for all the devices?

3. Will the key renegotiation be a part of TLS1.3 architecture, or are we supposed to do something else?

4. Starting point of implementation. *wpa_supplicant* or *hostap*? At what point will we deviate from standard EAP-TLS (purely in implementational sense)?