Comentario técnico: CTC-090
Componente: **WiFi Enterprise Authentication con ESP32 y Mongoose-OS**
Autor: Sergio R. Caprile, Senior R&D Engineer

Revisiones	Fecha	Comentarios
0	17/04/20	

En el CTC-089 comentamos cómo funciona la autenticación Enterprise en 802.11 WPA2. En este Comentario Técnico vamos a ver los detalles de configuración del ESP32, con Mongoose-OS, para poder conectarnos a una red WiFi con este tipo de seguridad. Para poder comprobar la correcta operación, hacemos uso de conceptos enunciados y archivos provistos en dicho Comentario Técnico; por lo que es altamente recomendable su lectura.

El ESP32 soporta, mediante el IDF, las siguientes opciones; con las características mencionadas:

- **EAP-TLS**
  - Requiere certificados de la CA y del usuario, así como la clave (key) del usuario
- **PEAP-MSCHAPv2**
  - Requiere certificado de la CA, nombre de usuario y password. Certificado y clave (key) del usuario son opcionales
- **EAP-TTLS (EAP-MD5 ó MSCHAPv2)**
  - Requiere certificado de la CA, nombre de usuario y password. Certificado y clave (key) del usuario son opcionales

Todas permiten autenticar al punto de acceso, y dependiendo de la implementación opcionalmente autentican también al usuario.
Tanto cuál de estas opciones se utiliza, como si se valida o no el certificado del usuario; es decir, si es realmente opcional o no, son determinadas por la configuración del servidor RADIUS. En otras palabras, son decisión del administrador de la red.

Para verificar el funcionamiento de estas opciones, hemos modificado en el servidor RADIUS el archivo *mods-available/eap*, particularmente la opción *default_eap_type*.

A continuación, configuramos el ESP32 con Mongoose-OS de acuerdo al método a probar. La configuración puede realizarse manualmente mediante RPC, en un archivo de configuración JSON, como indica la documentación[1]; o definirla en el archivo YAML que describe el proyecto. Para las pruebas elegimos esta última opción.

```
config_schema:
  - ["wifi.sta.enable", true]               # Enable Station mode
  - ["wifi.sta.ssid", "Sandbox"]            # WiFi network name
  - ["wifi.sta.user", "bob"]                # Username for auth in PEAP/TTLS
  - ["wifi.sta.pass", "hello"]              # Password
  - ["wifi.sta.anon_identity", "anonymous"] # Bogus identity for external EAP
  - ["wifi.sta.cert", "sandboxclient.crt"]  # Client certificate (*  optional)
  - ["wifi.sta.key", "sandboxclient.key"]   # Client key (EAP-TLS or optional)
  - ["wifi.sta.ca_cert", "ca.crt"]          # CA certificate
```

Así, las diferentes configuraciones resultan:

1 https://mongoose-os.com/docs/mongoose-os/api/net/wifi.md

- **EAP-TLS**

```
- ["wifi.sta.enable", true]              # Enable Station mode
- ["wifi.sta.ssid", "Sandbox"]           # WiFi network name
- ["wifi.sta.anon_identity", "anonymous"] # Bogus identity for external EAP
- ["wifi.sta.cert", "sandboxclient.crt"]  # Client certificate
- ["wifi.sta.key", "sandboxclient.key"]   # Client key
- ["wifi.sta.ca_cert", "ca.crt"]          # CA certificate
```

- **PEAP-MSCHAPv2; EAP-TTLS/MSCHAPV2; EAP-TTLS/MD5**

```
- ["wifi.sta.enable", true]              # Enable Station mode
- ["wifi.sta.ssid", "Sandbox"]           # WiFi network name
- ["wifi.sta.user", "bob"]               # Username for auth in PEAP/TTLS
- ["wifi.sta.pass", "hello"]             # Password
- ["wifi.sta.anon_identity", "anonymous"] # Bogus identity for external EAP
- ["wifi.sta.cert", "sandboxclient.crt"]  # Client certificate (*  optional)
- ["wifi.sta.key", "sandboxclient.key"]   # Client key (EAP-TLS or optional)
- ["wifi.sta.ca_cert", "ca.crt"]          # CA certificate
```

El parámetro *anon_identity* suele ser utilizado por el Access Point para otorgar acceso a la red, dado que es el primero en procesar EAP antes de conectarse con el servidor RADIUS; por lo que es necesario definirlo.

En caso de querer modificar sólo uno o algunos parámetros, podemos hacerlo directamente desde *mos tool*[2] usando la siguiente fórmula:

`mos config-set <nombre del parámetro>=valor [<nombre del parámetro>=valor]`

Es decir, por ejemplo: `mos config-set wifi.sta.ssid=miRed`
Lo mismo podemos realizarlo mediante RPC desde cualquier transporte soportado (aunque en este caso sólo hemos compilado el soporte elemental). En ese caso enviaremos un objeto JSON:

```
"wifi": {
  "sta": {
    "ssid": "miRed"
  }
}
```

La configuración actual podemos verla mediante el comando `mos config-get wifi.sta` , que nos devuelve un objeto JSON.
Finalmente, para modificar los archivos de configuración, observamos el directorio *fs* dentro del dispositivo. Dado que generamos la configuración en YAML, no lo veremos en nuestro directorio de la computadora en la que compilamos el programa. En *mos tool* pedimos un directorio via RPC mediante `mos call FS.List` y observaremos dos archivos: *conf0.json* y *conf9.json*; este último sólo estará presente una vez que hayamos hecho alguna modificación manual, por ejemplo mediante `config-set` como comentamos en el párrafo anterior. Deberemos editar *conf9.json*; para verlo en pantalla o copiarlo el comando es `mos get conf9.json` .
Para subirlo al dispositivo, el path es desde donde lo tengamos en la computadora; por ejemplo, si lo ubicamos donde debería estar, en el directorio *fs*, es `mos put fs/conf9.json` . Todo lo que se encuentra en ese directorio es empaquetado al compilar (`mos build`) y enviado al grabar el firmware (`mos flash`), por lo que guardar el archivo en *fs* hace que estos cambios no sólo sean permanentes en el dispositivo sino también en el proyecto.

2 https://mongoose-os.com/docs/mongoose-os/quickstart/setup.md