UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Autenticación avanzada con PANA en el subsistema de IP Multimedia (IMS)

Autor: Guillermo Rodea Palomares
Tutor: Andrés Marín López

7 de junio de 2009

Agradecimientos

Dedico este PFC a mi novia Lara por aguantar mis agobios continuos, a mis amigos, por hacer que los fines de semana sirvieran para olvidarme de él y a mi familia, por el apoyo y la ayuda prestada.

Agradecimientos especiales a mi tutor, Andrés Marín, por ayudarme en todo lo que pudo y a Davide Proserpio, del departamento de telemática, porque sin él, el HSS nunca hubiera estado terminado.

Resumen

Debido al uso masivo de IP en la actualidad, resulta importante poder disponer de métodos de autenticación y autorización, que permitan a los clientes registrarse en los servidores, de manera que estos puedan proporcionarles un acceso, en función de los servicios que tengan contratados.

De la misma manera, tanto los servidores como los clientes, esperan tener seguridad y confidencialidad en sus datos, y también que esta seguridad no suponga retardos ni pérdida de funcionalidades en los servicios.

El presente Proyecto de Fin de Carrera consiste en un análisis de los protocolos de seguridad que pueden estar involucrados en un entorno IMS[7], buscando su comprensión, y la integración de un cliente de seguridad en dicho entorno.

En este documento se tratarán, por un lado, la base teórica de protocolos de seguridad necesarios y, por el otro, la implementación de un cliente PANA[4] capaz de autenticarse con un servidor de autorización y autenticación situado en un entorno de IMS[7].

El hecho de elegir PANA como protocolo base se debe a su capacidad de ser independiente del nivel de enlace y de las tecnologías de los niveles inferiores.

La explicación, implementación y pruebas de distintos sistemas de autenticación, utilizando PANA como protocolo base, frente a un servidor de autenticación integrado en un core de IMS, es, en última instancia, el objetivo del presente PFC.

Abstract

Due to the massive use of the IP, nowadays, it is important to have authorization and authentication methods that allow the clients to register in servers, and allow the servers to provide access to these clients regarding the services that they have bougth.

In the same way, clients and servers want security and cofidentiallity to its data without lossing functionallity or access.

This PFC is about the integration of a PANA client in an IMS environtment.
This documents explains both, the teoric and the practice, about the integration of a PANA client in a IMS core.

We choose PANA because of its capability of being independent from the link layer and the lower levels technologies.

The explanation, development and test of different authentication systems, using PANA, against an authentication server in a IMS core, is the final goal of this PFC.

Índice general

Índice de figuras

Índice de cuadros

Parte I

Introducción al Proyecto

Capítulo 1

Introducción

1.1. Motivación del proyecto

En la actualidad, los protocolos capaces de proporcionar conexiones seguras entre varios extremos tienen una gran importancia.
El uso masivo de internet y el hecho de que cada vez puedan realizarse más acciones por esta vía, obliga a disponer de medios para garantizar la confidencialidad de los usuarios y sus datos, así como para garantizar la no suplantación de la identidad, y, en definitiva, garantizar conexiones seguras en todos los aspectos.

Por otro lado, desde el punto de vista de los proveedores de servicios, es importante tener la capacidad de dar acceso solamente a los usuarios que están registrados, y además poder dar distintos servicios, en función de quién sea el que se está conectando a la red.

Con el presente Proyecto de Fin de Carrera, se pretende programar y probar un cliente capaz de autenticarse en un entorno IMS[7]. Ésto es, un entorno de IP adaptado para servicios multimedia y capaz de interactuar con multitud de dispositivos, entre los que se incluyen los dispositivos móviles.
Para ello, es necesario conocer las distintas opciones disponibles y seleccionar cuales interesa usar. Por otro lado, es deseable probar varias de estas opciones para poder hacer una comparación.

A día de hoy, siendo un campo de rápida evolución y en pleno auge, existen pocas implementaciones de los protocolos de seguridad que aquí se estudiarán. Se trata de un tema que está en desarrollo, y muchos de los RFCs en los que se basa este trabajo son muy recientes, siendo algunos

incluso del año 2007 ó 2008.

Intentaremos crear un cliente capaz de autenticarse usando estos protocolos de manera flexible. Se elegirá PANA como protocolo básico, por los motivos que se explican más adelante. Por encima de PANA, es decir, en el nivel superior, se intentará conmutar entre varios protocolos distintos.

El entorno elegido se basará en una estructura de pasarela: Se utilizará un servidor intermedio para encaminar las peticiones del cliente al servidor de seguridad. Dicho servidor de seguridad final estará integrado en un entorno IMS[7].
Ésta estructura es bastante común en el entorno de IMS, y permite que el servidor final no esté obligado a entender todos los protocolos de todos los posibles clientes. Por el lado opuesto, también permite que los clientes no entiendan el protocolo que utilice el servidor de autenticación y, aún así, pueden autenticarse en él.

Para la implementación de los protocolos citados se usará la biblioteca de funciones OpenDiameter.

En resumen, los objetivos de este proyecto son tres: la familiarización con los protocolos de seguridad utilizados (PANA[4], EAP[2], Diameter[16] y EAP-TLS[1] principalmente), con sus funciones y con los distintos métodos de autenticación que pueden transportar.
La comprensión y manejo de las bibliotecas de funciones OpenDiameter.
Y, como última instancia, la creación de una cliente PANA funcional.

El hecho de elegir PANA como protocolo base para el cliente se debe a su independencia de tecnologías de nivel de enlace (Muchos otros protocolos de seguridad son dependientes de este factor) y a su capacidad para transportar otros protocolos de seguridad, tales como EAP[2].

También es destacable que, con el auge de las redes IP móviles, los usuarios se encuentran muy a menudo en la situación de que ni cliente ni servidor confían el uno en el otro a priori. La flexibilidad de PANA ayudará a solventar estas situaciones. Además, en estas situaciones, es también imprescindible tener un protocolo que sea independiente del nivel de enlace, ya que un protocolo limitado sólo a ciertos niveles no sería útil en gran cantidad de escenarios.

1.2. Objetivos

Los objetivos están elegidos para que se adapten a la motivación del proyecto, y para que, además, nos permitan realizar un análisis adecuado, tanto de los protocolos utilizados como de las bibliotecas de funciones.

Hay que tener en cuenta, que el tema tratado en el presente PFC es muy amplio, y que, por lo tanto, tendremos que centrarnos en ciertos aspectos del mismo, ya que sino sería imposible abordarlo de manera satisfactoria.

Los objetivos son los siguientes:

1. Analizar el funcionamiento de distintos protocolos usados para la autenticación de clientes en servidores:
   Se analizarán protocolos de distintos niveles y su integración mutua. Este análisis pondrá un especial interés en Diameter[16], ya que es el protocolo base usado en las autenticaciones en IMS[7].
   Los protocolos analizados serán: EAP[2], PANA[4], Diameter[16], Diameter-EAP[17], TLS[5] y EAP-TLS[1].

2. Realizar un análisis de las bibliotecas de OpenDiameter, que se usarán para la realización del trabajo: Se trata de bibliotecas de código abierto que, tal y como se explica en posteriores capítulos, proporcionan una implementación en el lenguaje C++ de los protocolos que se usarán, además de algunos programas ya construidos que, como veremos durante el desarrollo del proyecto, nos serán de utilidad.

3. Creación de un cliente PANA capaz de autenticarse con el servidor de IMS[7] (HSS): Usando el protocolo EAP[2] sobre PANA[2] y el sistema de autenticación MD5[20], se pretende crear un cliente que pueda ser autenticado en el entorno de IMS, a través de un servidor intermedio. Este es el cliente más básico que podemos crear con las especificaciones dadas, como puede comprobarse en la memoria, la autenticación se consigue en un número reducido de mensajes, y sin la intervención de ninguna entidad externa.

4. Utilización de EAP-TLS, en lugar de MD5: Conseguida la autenticación básica, se procederá a la integración en el cliente de un protocolo más complejo, que ofrece un nivel se seguridad mucho mayor, al basarse en criptografía asimétrica y certificados de seguridad.

5. Realización de pruebas sobre los distintos escenarios: Terminadas las

implementaciones, se probará el código creado y las contingencias específicas de cada escenario.

6. Análisis de los resultados de las pruebas y obtención de conclusiones: Se obtendrán los resultados de las pruebas y se comentarán los resultados obtenidos, con especial atención a los errores. Es importante en este punto ser críticos; tanto con el código programado, como con las librerías de OpenDiameter.

1.3. Contenido de la memoria

El presente documento se divide en cuatro partes:

1. Primera Parte: Introducción al proyecto: Esta sección incluye el Estado del arte, la motivación del proyecto y un resumen de objetivos. Es la presente sección del documento, por lo que no es necesario extenderse más en ella.

2. Segunda Parte: Desarrollo del proyecto en MD5: En esta sección se incluye el desarrollo del cliente EAP sobre PANA con MD5. Se incluye, además, el resumen del funcionamiento de las bibliotecas OpenDiameter y todos los datos referentes a la autenticación en ámbito local y remoto usando MD5, incluyendo la implementación del módulo Diameter-EAP en el servidor de IMS (HSS). Es la parte inicial del proyecto, en la que, como se ha explicado en los objetivos, se consigue una autenticación sencilla mediante el cliente programado.

3. Tercera Parte: Desarrollo del proyecto usando TLS. Autenticación en local usando EAP-TLS: Esta parte del proyecto consiste en sustituir, en el cliente inicial, MD5 por EAP-TLS. Sólo se implementó en local. Los motivos por los que la autenticación no se llevó a cabo en remoto se explicarán más adelante, pero, principalmente, se debe a que el servidor de IMS no poseía interfaz Diameter-EAP.

4. Cuarta Parte: Pruebas, conclusiones e historia del proyecto: Se incluyen las pruebas pasadas a cada una de las fases proyecto (MD5 en local y en remoto, TLS en local); las conclusiones obtenidas y un resumen del desarrollo del proyecto.

A estos apartados hay que añadir seis anexos:

1. Manual de instalación: Instalación de las bibliotecas OpenDiameter, tanto sin TLS como con TLS.

2. Manual de usuario: Manual para poder reproducir los escenarios comentados en el documento, así como para poder poner en funcionamiento el servidor de AAA, el NAS y el cliente PANA.

3. Implementación del módulo Diameter-EAP en el HSS: En este anexo se incluyen el código añadido en el servidor de IMS para implementar el módulo Diameter-EAP.

4. Ficheros de configuración: Explicación de los ficheros de configuración que usan los programas usados durante el desarrollo del proyecto.

5. Estructura de directorios: Explicación de la estructura de directorios de OpenDiameter y de los ficheros incluidos, en dicha estructura, durante la realización del presente proyecto.

6. Pruebas: Descripción y resultados de las pruebas expuestas en el capítulo de Pruebas del presente documento. Esta sección se añade a modo de complemento al trabajo realizado, y para que sea posible justificar algunas de las conclusiones expuestas.

Capítulo 2

Estado del arte

En este capítulo se realizará un análisis de las distintas tecnologías que tomarán parte en el proyecto,así como de los términos usados en el mismo. Dichas tecnologías son:

1. EAP
2. PANA
3. Diameter (también llamado Diameter Base Protocol)
4. Diameter-EAP
5. TLS
6. EAP-TLS

2.1. Terminología y palabras clave

- Autenticador (Authenticator) : El final de un link en el que se ha iniciado un proceso de autenticación,su función es verificar la identidad del otro extremo.
- Peer: El otro extremo del link, es decir, el que requiere la autenticación por parte del autenticador.
- Backend authentication server: Servidor que proporciona a un autenticador los servicios de autenticación.
- AAA: Authentication, Authorization and Accounting.

- Descartar silenciosamente: Acción de descartar un mensaje sin tomar ningún procesado adicional.
- Autenticación exitosa: Final de un proceso en el cual el autenticador decide dar acceso al Peer y el Peer decide usar el acceso proporcionado por el autenticador.
- Message Integrity Check (MIC): Función de tipo hash para la autenticación y protección de los datos.
- Master Session Key (MSK): Material criptográfico que es derivado por el cliente y el servidor y exportado mediante un método EAP. Tiene al menos 64 bytes de longitud.
- Pana Client (PaC): Lado del cliente del protocolo PANA, reside en el dispositivo que pretende acceder a la red y debe proporcionar sus credenciales para ser autenticado e identificado.
- Pana Authentication Agent (PAA): Entidad del protocolo PANA cuya finalidad es verificar las credenciales proporcionadas por el PaC para acceder a la red. Está colocado en el mismo nodo que un autenticador EAP o en un Backend Authentication Server.
- Pana Session: Sesión establecida entre un PAA y un PaC. Mantiene un identificador fijo durante su duración. Puede terminar por varios motivos: Un fallo de autorización, un mensaje fallido después del número máximo de retransmisiones, fin del tiempo de vida de la sesión o, en general, cualquier evento que cause la desconexión.
- Session LifeTime (tiempo de vida de la sesión): Duración asociada con una determinada sesión segura, puede ser extendido antes del fin de la sesión.
- Session Identifier (Identificador de la sesión): Identificador usado para identificar una única sesión de algunos protocolos de seguridad, tales como PANA, es único para cada sesión.
- PANA Security Association (PANA SA): Sesión de seguridad que se forma entre el PaC y el PAA compartiendo material criptográfico y un contexto asociado. Se usa para proteger el tráfico de señalización bidireccional entre el PaC y el PAA.
- Enforcement Point (EP): Nodo de la red de acceso donde se aplican políticas de control (por ejemplo, filtros) a los paquetes tanto entrantes como salientes.

- PRF: Función pseudoaleatoria.
- Nonce: Número aleatorio usado en criptografía.
- Negociación protegida del entorno de seguridad (Protected ciphersuite negotiation): Capacidad de negociación del entorno de seguridad y seguridad de dicha negociación.
- Autenticación mutua (Mutual authentication): Cada extremo del enlace autentica al otro extremo mediante un intercambio de mensajes.
- Protección de la integridad (Integrity protection): Capacidad para proteger el origen de los datos y para evitar las modificaciones o lecturas no autorizadas de los datos.
- Protección contra la repetición (Replay protection): Capacidad de evitar que un atacante pueda replicar los mensajes ya enviados.
- Confidencialidad (Confidentiality): Capacidad de evitar que un tercero pueda leer los mensajes intercambiados, está ligada a la encriptación.
- Derivación de claves (Key derivation): Capacidad para exportar material criptográfico derivado de unas aplicaciones a otras.
- Fortaleza de la clave (Key Strength): Dificultad teórica para romper una clave.
- Resistencia a ataques de diccionario (Dictionary attack resistance): Los ataques de diccionario se basan en que a la hora de elegir una clave se utilizan un conjunto reducido de las 2 elevado a (N-1) posibles palabras-código. Hay protección contra ataques de diccionario cuando un atacante no puede usar este hecho para romper la clave.
- Reconexión rápida (Fast Reconnect): Se refiere a, cuando una asociación de seguridad ha sido creada anteriormente entre los extremos, se puede crear otra más eficientemente en el futuro.
- Asociación criptográfica (Cryptographic binding): Demostración del Peer al servidor y viceversa de que una sola entidad ha actuado como Peer o como servidor en todos los mecanismos a través de un método que implique túnel. Puede ser aplicado en ambos sentidos.
- Independencia de sesiones (Session independence): Consiste en garantizar que la captura de mensajes pasados no pueda usarse para crear o mantener posteriores sesiones.

- Vinculación por medio de canal (Channel binding): Uso de un canal que protege la integridad de los datos.
- CA (Certification Authority): Entidad de certificación, entidad encargada de garantizar que los certificados de dos Peers son correctos.

2.2. Extensible Authentication Protocol (EAP)

2.2.1. Introducción

EAP[2] es un contexto de autenticación que soporta una serie de métodos de autenticación. Su principal función es la autenticación de Peers frente a un servidor. Es también una función de EAP el proporcionar un contexto seguro para las comunicaciones entre ellos (Peer y Servidor). Puede ser usado individualmente o combinado con otros protocolos de seguridad, que irán por debajo de EAP. Típicamente corre directamente sobre el nivel de enlace, por ejemplo PPP o 802.x sin requerir IP, aunque también puede ir sobre otros protocolos de seguridad, como TLS o PANA.
El motivo de la elección de EAP, y por el cual se incluye en esta memoria, es que se integra muy bien con PANA[4], ya que PANA está diseñado para llevar EAP. Además EAP permite cambiar de método de autenticación cambiando simplemente un campo de la trama y se integra con Diameter[16], existiendo un RFC específico para llevar EAP sobre Diameter[17]. Ésto es importante, teniendo en cuenta que el core de IMS usa Diameter para sus comunicaciones.

EAP soporta retransmisión y eliminación de duplicados, pero no soporta reordenación de paquetes en destino ni fragmentación (por sí mismo, ciertos métodos de EAP si pueden soportar alguna o todas estas características). EAP está diseñado para funcionar tanto en redes cableadas como redes radio,independientemente de si se trata de links dedicados (paquetes) o de circuitos conmutados.

La principal ventaja de EAP es su flexibilidad. Típicamente, EAP se utiliza para elegir un determinado mecanismo de autenticación después de que el autenticador (authenticator) requiera más información para elegir el método de autenticación que será usado. Otra característica importante de EAP es que se trata de un protocolo de 'un solo paso'(lock step), es decir, sólo un mensaje puede estar transmitiéndose al mismo tiempo. Lo que lo hace poco indicado para transportar datos.

2.2.2. Procedimiento general

Veremos ahora como se produce en general un autenticación mediante EAP:

- El autenticador envía una Petición (Request) para autenticar al Peer. Esta petición tiene un campo 'Tipo'para indicar lo que se está pidiendo (Identidad, Reto MD-5...). Típicamente el campo tipo corresponderá a identidad (Identity), es decir, se requerirá la identidad del otro extremo.
- El Peer envía un paquete de tipo Respuesta (Response) que contiene un campo 'Tipo'con el mismo valor que el campo tipo que le llegó en la Petición.
- El proceso se repite durante un número de iteraciones hasta que el autenticador toma una decisión y finaliza el proceso, enviando un Éxito (Sucess, código 3) o Fracaso (Failure, código 4). El servidor no debe enviar un Éxito si hay paquetes retransmitiéndose o si alguna de las respuestas del Peer ha sido errónea.

2.2.3. Secuencias de métodos

EAP puede soportar secuencias de métodos. Por ejemplo una petición de tipo identidad seguida de un reto MD-5. El autenticador no debe enviar una Petición de un tipo distinto hasta recibir la Respuesta a la petición del tipo que envió anteriormente. En esencia debe respetarse el principio de solo un mensaje viajando por la red al mismo tiempo.

2.2.4. Torre de Protocolos en EAP

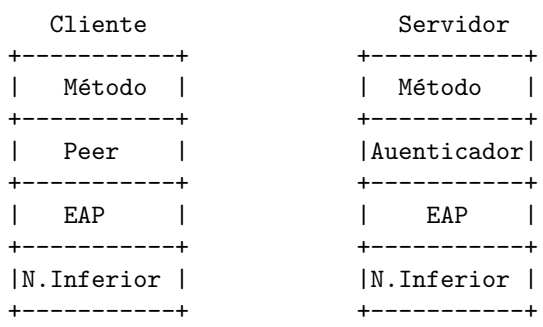


Figura 2.1: Torre de protocolos en EAP

1. Nivel inferior (Lower Layer): Se encarga de transmitir las tramas entre los dos extremos de la comunicación, es cualquier protocolo que esté situado por debajo de EAP, y como se ha dicho antes, puede ser directamente un nivel de enlace, IP, otro protocolo de seguridad como PANA, etc.

2. Nivel EAP: Recibe y transmite paquetes EAP a través del nivel inferior. Implementa la detección de duplicados y la retransmisión.

3. Niveles Peer y Autenticador: Basándose en el campo -Tipo-, el nivel EAP demultiplexa los paquetes hacia los niveles Peer y autenticador de la implementación. Es posible que una misma máquina presente ambos niveles, aunque típicamente se implementará uno de los dos. El campo tipo funciona de manera similar a los puertos en UDP y TCP.

4. Nivel de métodos EAP: Implementa los métodos y algoritmos de autenticación propiamente dichos, como por ejemplo, MD5. En caso de soportarse fragmentación, dado que los niveles inferiores no lo soportan, deberá implementarse en este nivel.

2.2.5. El nivel inferior en EAP

Requisitos del nivel inferior

EAP puede funcionar sobre muchos protocolos de nivel inferior (desde el punto de vista de EAP), no obstante, se requiere que esos protocolos cumplan los siguientes requisitos:

1. No retransmisión: El nivel EAP realiza las retransmisiones necesarias cuando no recibe respuestas a sus peticiones, por lo tanto no necesita que el nivel de enlace realice retransmisiones de paquetes no confirmados. Puede darse la situación de que ambos niveles realicen la retransmisión (aunque no es recomendable). Por otro lado, los paquetes de Éxito (Success) y Fracaso (Failure) no son retransmitidos, por lo que un medio con una alta tasa de error y un nivel de enlace que no realice retransmisiones pueden derivar en fallos de autenticación.

2. Seguridad: EAP no requiere ninguna seguridad en el nivel de enlace, sin embargo, en métodos que soporten derivación de clave, pueden proveerse características de seguridad al nivel de enlace sin que esto supongo un problema para EAP.

3. Tasa de error: Debido a la política de no retransmisiones de Success y Failure, una alta tasa de error puede derivar en graves problemas de eficiencia del protocolo.

4. MTU mínima: Se requiere una MTU de 1020 octetos o superior.

5. Conservación del orden: EAP no requiere un orden creciente en el identificador de los paquetes, sin embargo, paquetes desordenados pueden desencadenar errores de autenticación, por lo tanto, se recomienda el uso de protocolos de nivel inferior que garanticen el orden (UDP o IP -crudos- no son recomendados).

Uso de EAP sobre PPP

Formato de los mensajes:

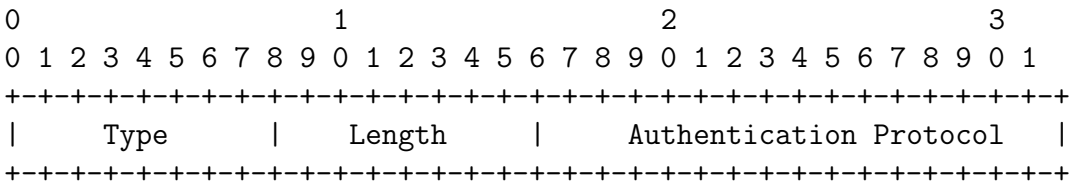


Figura 2.2: Formato de mensaje EAP sobre PPP

Uso de EAP sobre 802

El RFC no especifica formato de mensajes.

2.2.6. Los mensajes en EAP: Formato y tipos

Formato de los mensajes EAP

Un mensaje en EAP, de forma genérica, tiene el siguiente formato:
Junto con la correspondiente descripción de los campos.

- Code: 1 octeto. Los códigos soportados son cuatro: 1 (Request), 2 (Reponse), 3 (Success) y 4 (Failure), cualquier otro Code debe ser descartado silenciosamente.

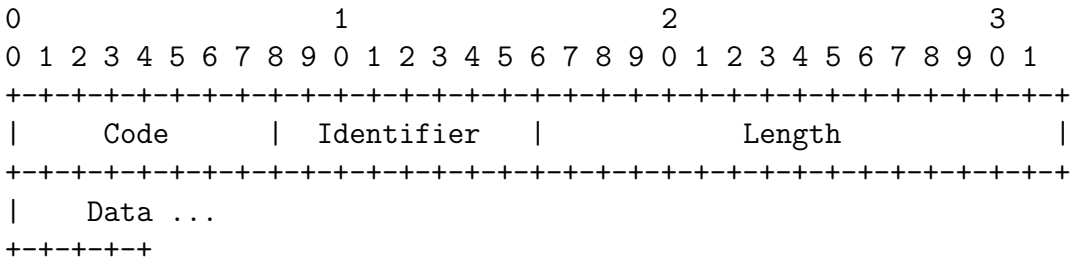


Figura 2.3: Formato de mensaje EAP genérico

- Identifier: 1 octeto, une las respuestas (Reponse) con las peticiones (Request). La respuesta a una petición debe llevar el mismo identificador que llevaba esta última.
- Length: 2 octetos, incluye Code, Identifier, Length y Data están incluidos. Los octetos fuera de rango son ignorados, y si un paquete tiene un campo Length mayor que la longitud real recibida será descartado silenciosamente.
- Data: 0 o más octetos conteniendo los datos transportados por el protocolo.

Tipos de mensajes EAP

Existen cuatro tipos de mensajes que pueden enviarse en EAP:

1. Request: El mensaje de tipo Request (petición) es enviado por el Autenticador y recibido por el Peer. Debe llevar un campo Type asociado, que indicará el tipo de petición de la que se trata.
   Cada paquete deberá llevar un campo Identifier distinto, que no podrá ser reutilizado hasta que el paquete haya sido confirmado. En caso de recibir un paquete con un Identifier al que ya se ha contestado se asumirá que se trata de una retransmisión y retransmitirá la respuesta anterior sin hacer ningún procesado adicional.
2. Response: El mensaje de tipo Response (respuesta) se envía, como su nombre indica, como respuesta a un mensaje de tipo Resquest recibido. Debe llevar el mismo campo Identifier que llevaba el Request.
   El formato de estos dos mensajes es el mismo:
3. Success: El mensaje de tipo Success se envía del Autenticador al Peer cuando el proceso de autenticación ha tenido éxito. Su campo -Type-

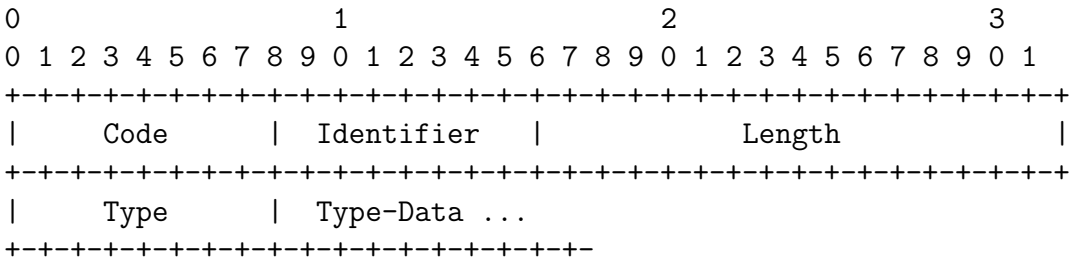


Figura 2.4: Formato de mensaje Request/Response

debe contener un 3 y su campo -Data- debe estar vacío.
Un Peer sólo debe aceptar mensajes de este tipo cuando espera recibirlos, y no antes ni después. Este mecanismo evita ataques de falsos autenticadores. Los mensajes de tipo Success recibidos una vez se ha procesado el primero serán descartados silenciosamente.

4. Failure: El mensaje de tipo Failure se envía del Autenticador al Peer cuando el proceso de autenticación ha fallado. Su campo -Type- debe contener un 4 y su campo -Data- debe estar vacío.

El formato de estos dos mensajes es el siguiente:

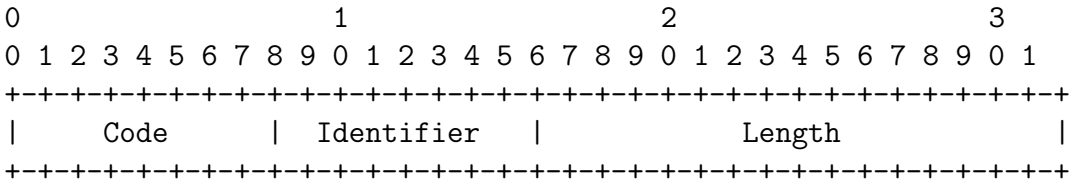


Figura 2.5: Formato de mensaje Request/Response

Tipos de Request y Response

Existen varios tipos de Requests y Responses que pueden ser intercambiados entre un Servidor y un cliente EAP:

1. Identity (Type 1): Es la petición inicial más común, sirve para pedir la identidad del Peer. Tras recibir esta petición el Peer enviará una respuesta de tipo 1. El campo -Type data- puede contener caracteres UTF-8 para ser mostrados al usuario (o puede estar vacío).

2. Notification (Notificación) (Type 2): Es un tipo opcional de mensaje que envía un mensaje desde el autenticador al Peer, el cual será mostrado al usuario (Notification Request). Debe ser contestado con un Notification Reponse. La longitud máxima de los datos de esta notificación es por defecto 1020 octetos.

3. Nak (Type 3). Hay dos tipos: -Legacy Nak-, sólo válido en respuestas y que indica que el tipo de autenticación requerida es inaceptable; y el -Extended Nak- también enviado en respuestas a peticiones de tipo 254 para pedir alternativas de autenticación.

4. Desafío o reto MD-5 (Type 4): La petición contiene un desafío MD-5[20][1] para el Peer. Puede contestarse con una respuesta del mismo tipo o con un Nak que indica que se quieren alternativas de autenticación.

5. OTP (One Time Password) (Type 5):Indica que la autenticación se hará por medio de OTP[13], del mismo modo que el reto MD-5, puede ser contestado con una respuesta de su mismo tipo o con una respuesta tipo Nak.

6. GTC (Generic Token Card) (Type 6): La petición consiste en un texto que se mostrará al usuario y la respuesta en el 'Token Cardñecesario para la autenticación. Puede contestarse de la misma manera que el reto MD-5.

7. Expanded Type (Type 254) : Están pensados para tipos que dependan del proveedor. Su estructura estándar es la siguiente:

8. Experimental (Type 255): El tipo experimental no tiene formato definido y se usa para realizar pruebas sobre el protocolo EAP.

2.2.7. Modos de funcionamiento de EAP

EAP define dos modos de funcionamiento: Funcionamiento Peer to Peer y funcionamiento como pasarela o 'Passthrought'.

[1]MD-5 es un protocolo basado en cifrado de clave privada que consiste en lo siguiente: El servidor, que conoce la clave del cliente, le envía a este un nonce, el cliente cifra el nonce de acuerdo a su clave privada y se lo reenvía al servidor. Si este nonce cifrado consiste con el nonce cifrado que construye el servidor usando la clave del cliente la autenticación tiene éxito, en caso contrario la autenticación falla

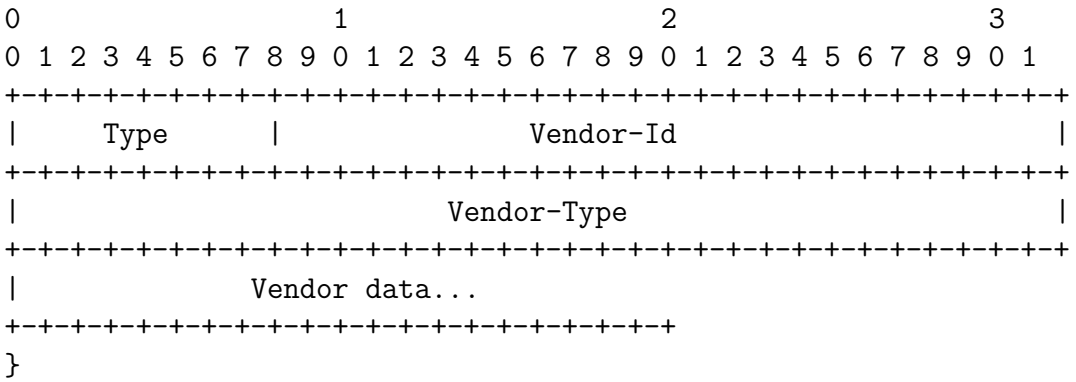


Figura 2.6: Formato de mensaje EAP de tipo Extendido

Funcionamiento Peer to Peer

Dado que EAP es un protocolo extremo a extremo, puede ser requerida una autenticación mutua, ambos extremos pueden actuar como Peers y autenticadores al mismo tiempo (implementando ambos los niveles de protocolo correctos). Este requerimiento es soportado por ciertos métodos EAP, pero no por todos. Hay que tener en cuenta ciertas consideraciones para poder funcionar en este modo:

1. El nivel de enlace debe tener soporte para derivación de claves bidireccional. Niveles de enlace como 802.11 sólo soportan derivación unidireccional.

2. Soporte para resultados en ambos sentidos de la comunicación (Peer autenticado en el servidor, servidor autenticado en el Peer)

Funcionamiento como pasarela

Se trata de una configuración de EAP muy utilizada.

En la configuración como pasarela el autenticador reenvía los paquetes del Peer al Backend Authentication Server y viceversa. En esta configuración, el nodo EAP, no posee nivel 4 (soporte de métodos EAP), por lo tanto no lleva a cabo la autenticación sino sólo un transporte de los métodos desde el cliente al servidor.
Una vez recibido un paquete, el Peer que actúa como pasarela tiene varias opciones: actuar sobre él, descartarlo o reenviarlo. La decisión de reenvío se toma fijándose únicamente en los campos -Código- (Code), -Identificador-

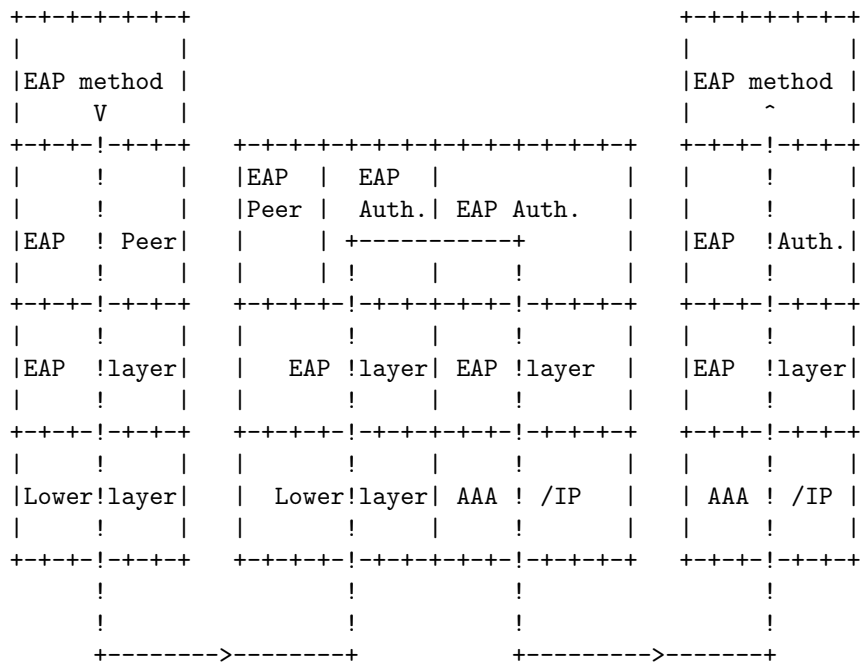


Figura 2.7: Funcionamiento de EAP como Pasarela.
[Como veremos después, el NAS se basa en este esquema para funcionar.]

(Identifier) y -Longitud- (Length).
Esta configuración tiene una serie de ventajas y desventajas:

- Ventajas:
  1) Permite el uso de mecanismos de autenticación sin haber tenido que negociarlos de antemano directamente con el Peer.
  2) Los servidores de acceso a la red no tienen porque entender todos los métodos de autenticación soportados por el Backend authentication Server y pueden actuar como pasarela retransmitiendo los mensajes sin procesarlos.
  3) Ofrece la posibilidad de separación entre el autenticador y el Backend autentication server, que simplifica las decisiones.

- Desventajas:
  1) El uso de distintos niveles de enlace hace que,a veces, se deban realizar modificaciones para usar los métodos.
  2) Si el autenticador está separado del Backend Authentication Server aparecen potenciales complicaciones de seguridad en el enlace entre

estos dos puntos.

2.2.8. Consideraciones de seguridad en EAP

Hasta ahora hemos definido la torre de protocolos EAP, los tipos y formato de los mensajes EAP y los tipos de autenticación que pueden lograrse. Ahora nos centraremos en algunas consideraciones sobre seguridad que es necesario tener en cuenta usando EAP.
En primer lugar se explicarán los distintos tipos de ataque que pueden afectar a EAP, posteriormente se explicarán las pautas de seguridad que deben seguir los métodos EAP que se creen.

Tipos de ataque

EAP puede ser objetivo de los siguiente ataques:

1. Un atacante puede intentar descubrir las identidades de los usuarios interceptando el tráfico de autenticación.
   Para evitar este tipo de ataques, en algunos métodos EAP el intercambio de identidades puede postergarse hasta haber obtenido un canal seguro por el que transmitirlas.

2. Un atacante puede tratar de modificar los mensajes intercambiados.
   Este tipo de ataque puede realizarse contra el campo -Identifier- de los mensajes EAP. La integridad de los mensajes EAP se comprueba a nivel de método, y no a nivel EAP, por lo que este campo es vulnerable, y su modificación puede ocasionar interpretaciones y descartes de mensajes erróneos.
   Para evitar este ataque se recomienda usar los MIC (Message Integrity Check) que definen los métodos EAP.

3. Pueden intentarse ataques de DoS (denegación de servicio) por ejemplo, creando paquetes con identificadores duplicados o reenviando paquetes. En estos casos el Descarte silencioso de paquetes es fundamental.

4. Un atacante puede intentar conectar a un Peer a una red no segura mediante ataques de -hombre en el medio- (man-in-the-middle).
   Esta complicación de seguridad surge por el intercambio postergado de identidades explicado en el punto 1. Se debe a que este desconocimiento de identidades por parte de los extremos puede hacer que el Peer negocie con un atacante en lugar de con el servidor, creando un

túnel con este, y que este atacante, a su vez, se conecte al servidor obteniendo información de ambos extremos de manera sencilla.
Este ataque puede mitigarse de la siguiente forma:

a) Limitar los métodos EAP que pueden utilizarse en cada túnel.

b) Requerir que la autenticación mutua se haga mediante el uso de métodos EAP.

c) Requerir un vínculo criptográfico entre las claves usadas para crear el túnel y las usadas en los métodos EAP que usen el túnel.

d) Evitar el uso de túneles si el método es lo suficientemente seguro por si mismo.

5. Un atacante puede intentar recuperar las claves con ataques de diccionario. Este ataque puede realizarse si se usa un algoritmo débil frente a este tipo de ataques. El uso de un túnel soluciona este problema, pero las complicaciones de man-in-the-middle que implican los túneles aconsejan que este tipo de métodos no se utilicen.

6. Se puede intentar alterar la negociación para que los extremos elijan un método de autenticación menos seguro. Esto se consigue por medio de los mensajes tipo Nak. EAP no controla este tipo de mensajes, pero el nivel superior debe encargarse de realizar esta gestión y evitar el uso indiscriminado de mensajes tipo Nak[2].

7. Se pueden intentarse derivar claves utilizando las debilidades del generador de claves de EAP. Para evitar este ataque es necesario que haya separación criptográfica entre las distintas claves usadas para generar las claves de sesión EAP [3].

8. Se puede insertar información incorrecta o inconsistente en la red. Este tipo de ataque se produce a nivel de enlace, y por lo tanto debe ser solucionado por los protocolos de dicho nivel. No obstante, EAP es vulnerable a este tipo de ataques; por lo tanto, el nivel de enlace debe estar bien elegido.

[2]Veremos en la fase de pruebas del proyecto como el servidor de autenticación ignora los mensajes de tipo Nak para evitar este tipo de ataques

[3]La clave usada en la sesión EAP es la AAA-key, esta clave es derivada de la MSK y la EMSK, ambas son claves de 64 bytes, y deben estar separadas criptográficamente para evitar derivaciones de claves por parte de atacantes.

9. Un Peer puede conectarse a una red hostil sin saberlo. Esto se debe al uso de métodos que sólo autentican a uno de los extremos. Se recomienda no usar esta clase de métodos.

Especificaciones de seguridad de los métodos EAP

En cada definición de un método EAP deben especificarse:

1. Mecanismo: Mecanismos de seguridad empleados. Un ejemplo de mecanismo de seguridad es MD-5

2. Especificaciones de seguridad: usando términos criptológicos. Deben justificarse las metas de seguridad conseguidas.

3. Fortaleza de las claves: La fortaleza de las claves derivadas debe ser estimada. Si la longitud de una clave es N la mejor fortaleza de la clave usando los métodos actuales debería ser 2 elevado a (N-1).

4. Descripción de la jerarquía de claves.

5. Indicación de las vulnerabilidades.

Con estas consideraciones, podemos hacernos una idea del nivel de seguridad alcanzado por el método; y de si es adecuado para nuestros objetivos.

Complicaciones de seguridad adicionales derivadas de separar el servidor de autenticación y el autenticador

Al usar la configuración de pasarela, el autenticador y el servidor de autenticación residen en distintas máquinas, hay varias implicaciones de seguridad:

1. No es posible para el cliente validar la identidad del autenticador usando EAP (ya que la autenticación es entre el y el servidor de autenticación.

2. El protocolo usado entre el autenticador y el servidor de autenticación debe ser suficientemente seguro o bien utilizarse un canal seguro para las comunicaciones entre ellos.

3. Debe proporcionarse autenticación mutua entre el cliente y el autenticador una vez creado el contexto de seguridad.

4. La clave negociada entre el cliente y el servidor de autenticación puede ser enviada al autenticador. Debe proporcionarse un método seguro para esto.

Seguridad en los Mensajes Success y Failure

En EAP, los paquetes de Éxito (Success) o Fracaso (Failure) no reciben ack ni están protegidos en el sentido de preservación de la integridad. Esto puede solucionarse en gran medida mediante la autenticación mutua del servidor y el cliente.
Dependiendo del método y del contexto usados estos mensajes podrán ir protegidos o no, siendo de esta manera vulnerables frente a un atacante o no. Es posible que un atacante cree un paquete de tipo Success después de que el servidor haya autenticado al cliente, pero antes de que el cliente haya autenticado al servidor, si el cliente acepta este paquete e intenta acceder a la red sin estar autenticado la conexión fallará, creándose un ataque de denegación de servicio.
Una forma de evitar este ataque consiste en, si el método utilizado soporta indicaciones de servicio a nivel de enlace, el Peer no se dará por autenticado hasta que reciba una confirmación del servidor.

2.2.9. Conclusiones

- EAP es un protocolo de seguridad pensado para llevar por encima otros protocolos de seguridad o directamente métodos de seguridad.
- EAP puede ir directamente montado sobre protocolos de nivel de red o transporte, pero sus carencias en este aspecto hacen que no sea recomendable.
- EAP puede exportar sus características de seguridad a los protocolos que vayan por debajo de él.

En definitiva, EAP es un protocolo versátil y con buenas características de seguridad, pero que debe ser complementado por otro protocolo de nivel inferior.

2.3. Protocol for Carrying Authentication for Network Access (PANA)

2.3.1. Introducción

Dar seguridad a una red implica controlar el acceso a la misma. La autenticación cliente a red (Client-to-network authentication) proporciona parámetros que son necesarios para hacer política del tráfico que circula

por dicha red, esta autenticación cliente a red para la obtención de estos parámetros es la principal función que realiza PANA[4].

PANA transporta otra serie de protocolos de autenticación (EAP) de manera independiente del nivel de enlace y se encarga de realizar las políticas de control dentro de la red. Por ejemplo, la asignación de nuevas direcciones IP.

PANA transporta EAP, que a su vez transporta otros métodos EAP, por lo tanto PANA es un nivel de enlace (Lower Layer) para EAP. Es decir, PANA en sí mismo no proporciona ninguna autenticación, sino que proporciona mecanismos para transportar otros protocolos que sí la proporcionen.

PANA es el protocolo con el que programaremos el cliente que pretendemos crear, y es protocolo con el que este cliente se comunicará con el servidor de Acceso, que le pondrá en contacto con el servidor de IMS.

2.3.2. Funcionamiento general

Generalmente, el protocolo corre entre un PaC (PANA Access Client) y un PAA (PANA Authorization and Authentication server) con el objetivo de proporcionar autenticación y autorización en una determinada red de acceso. PANA consiste en una serie de peticiones y respuestas, muchas de las cuales pueden ser iniciadas en cualquiera de los dos extremos y otras, sólo pueden ser iniciadas por uno de los dos.
Es un protocolo que funciona sobre UDP, y posee su propio mecanismo de retransmisión de paquetes.

Una sesión de PANA se compone de distintas fases:

1. Fase de autenticación y autorización: Se crea una nueva sesión PANA y se ejecuta EAP entre el PaC y el PAA. Se usa el payload de EAP (que lleva un método de EAP dentro) para realizar la autenticación.

2. Fase de acceso: Si la fase anterior fue correcta el dispositivo obtiene acceso a la red y puede enviar tráfico IP a través de ella[4]. Durante esta fase el PaC y el PAA pueden, opcionalmente, mandar notificaciones PANA para probar si el enlace sigue vivo.

[4]Como se ha visto antes, EAP no es adecuado para transportar tráfico de datos, por lo que una vez utilizado EAP para conseguir la autenticación, los datos viajarán directamente sobre PANA

3. Fase de reautenticación: Durante la fase de acceso, el PAA puede, y el PaC debería iniciar una reautenticación si desean que la sesión se renueve antes de que el tiempo de vida de la sesión expire. Es una subfase de la fase anterior.

4. Fase de terminación: Tanto PAA como PaC pueden elegir terminar la sesión en cualquier momento. Si la sesión termina sin ningún tipo de aviso o mensaje entonces, o bien el tiempo de duración o bien los mensajes de notificación PANA acabarán con la sesión posteriormente.

La protección criptográfica de los mensajes PANA es posible aprovechando el exportado de claves de EAP.

2.3.3. Reglas de procesado en PANA

En esta sección trataremos temas tales como la fragmentación de mensajes en PANA o como se prueba la validez de los mismos.

Fragmentación

PANA no soporta fragmentación, pero puede usar la de IP o la de algunos métodos EAP si es necesario.

Números de secuencia y retransmisión

El PaC y el PAA mantienen dos secuencias de números distintas: una para la siguiente petición entrante y otra para la siguiente petición saliente. Son monótonamente crecientes (de 1 en 1) y tienen 32 bits de longitud, siendo cuentas circulares. Los números de inicio de estas secuencias (ISN) con escogidos aleatoriamente. Un mensaje sólo será aceptado si su número de secuencia coincide con el esperado.
En cuanto a la política de retransmisiones; cada mensaje será retransmitido hasta que la respuesta al mismo sea recibida, se acabe el temporizador o se alcance el número máximo de retransmisiones. Tanto PaC como PAA deben responder a todos los paquetes duplicados que reciban.

PANA securtity association (PANA SA)

Se crea como un atributo de la sesión PANA cuando la autenticación EAP tiene éxito. Culmina con la creación de un MSK que aporta material criptográfico a una sesión PANA.
El tiempo de vida de la PANA SA es el mismo que el de la sesión PANA, y

aporta nuevos atributos a la sesión:
Atributos de la sesión PANA:

- Identificador de la sesión.
- IP y puerto del PaC.
- IP y puerto del PAA.
- Número de secuencia para la siguiente petición entrante.
- Número de secuencia para la siguiente petición saliente.
- Payload del último mensaje transmitido.
- Intervalo de retransmisión.
- Tiempo de vida de la sesión.

PANA SA añade:

- Nonce generado por el PaC.
- Nonce generado por el PAA.
- MSK.
- Identificador del MSK.
- PANA-AUTH-KEY.
- Función pseudoaleatoria.
- Algoritmo para proporcionar integridad.

La PANA-AUTH-KEY es una clave derivada de la MSK.

Autenticación de mensajes en PANA

PANA no proporciona mecanismos de autenticación de mensajes por sí mismo, no obstante, puede heredar claves de los protocolos que transporta, por ejemplo EAP, en ese caso incluirá un atributo Auth-AVP que se usará para encriptar los mensajes, y que se basa en una función de hash.

Validez de un mensaje

Para que un mensaje PANA que se recibe se considere correcto deben cumplirse las siguientes condiciones:

1. Cada campo de la cabecera contiene un valor válido para ese campo.
2. El tipo de mensaje es uno de los esperados, ésto depende del estado de la sesión.
3. El payload del mensaje contiene un conjunto de AVPs[5] que corresponden a algún mensaje PANA.
4. Cada AVP es de un tipo conocido y además de decodifica correctamente.
5. En caso de que se use encriptación, el mensaje se desencripta correctamente.

Si un mensaje no cumple alguno de los puntos anteriores debe ser descartado silenciosamente.

Tiempo de vida de la sesión

El tiempo de vida de la sesión no es un parámetro negociable.
Es enviado por el PAA al PaC. Una vez vencido este temporizador no es posible entrar en fase de reconexión y deberá crearse una conexión nueva desde 0.

2.3.4. Formato de los mensajes

Cabeceras IP y UDP

El único requisito para estos protocolos es que el puerto asignado a PANA es el 716. No obstante, otros puertos pueden ser usados si es necesario.

Formato de los mensajes PANA

El formato general de un mensaje PANA es el siguiente.

- Reserved: 16 bits reservados para uso futuro. Deben fijarse a 0 y ser ignorados en el receptor.

[5]Un AVP (Attribute Value Pair) es un campo de datos que puede ser añadido o no a los mensajes. Pueden contener información sobre enrutamiento, capacidades, etc o sobre autorización o autenticación.

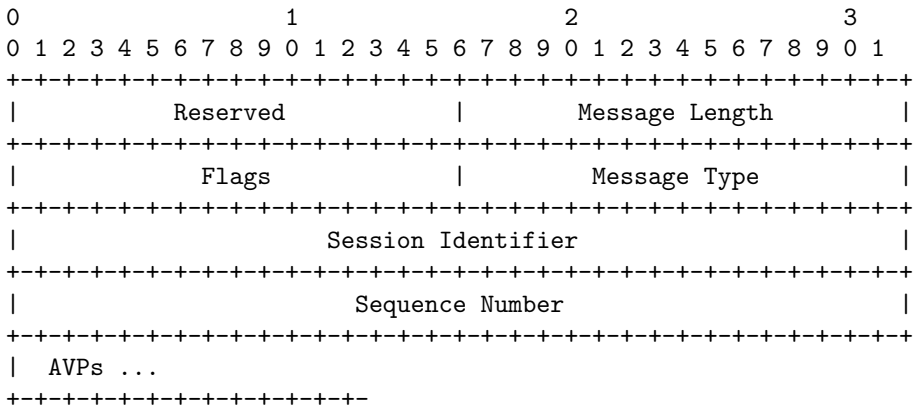


Figura 2.8: Formato de un mensaje PANA

- Message Length: 2 octetos e incluye la cabecera.
- Flags: son dos octetos:

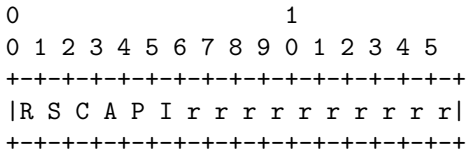


Figura 2.9: Formato de los flags en un mensaje PANA

  - R (Request): Indica que el mensaje es una petición.
  - S (Start): Éste es el primer mensaje durante la fase de autorización y autenticación.
  - C (Complete): Éste es el último mensaje durante la fase de autenticación y autorización.
  - A (Re-Authentication): Petición de reautenticación.
  - P (Ping): Es una petición o respuesta para probar si el enlace sigue disponible.
  - I (IP reconfiguration) : Se requiere que el PaC reconfigure su IP
  - r (reserved): reservados para uso futuro.
- Message Type: 2 octetos, indica el tipo de mensaje
- Session Identifier: 32 bits. Contiene el identificador de la sesión.

- Sequence Number: 32 bits. Contiene el número de secuencia.
- AVP: Un AVP sirve para encapsular información relevante para el mensaje PANA. En la siguiente sección se tratarán más a fondo los AVP.

Formato de los AVPs

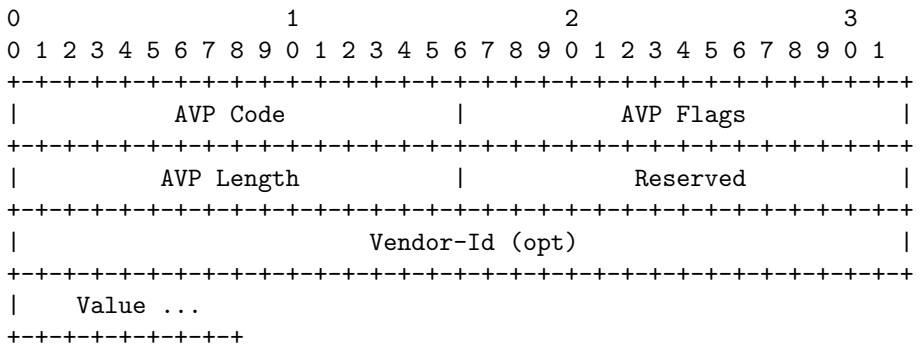


Figura 2.10: Formato de los AVPs de un mensaje PANA

- AVP Code: Hay dos opciones. Si el Vendor-Id está presente, identifica al atributo que le sigue; sino lo está, se refiere a un atributo IETF.
- Flags:

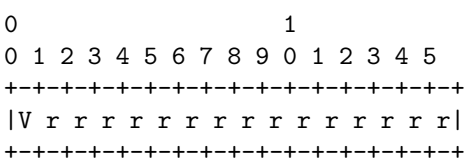


Figura 2.11: Formato de los flags de un AVP de un mensaje PANA

  - V (Vendor): indica si el campo Vendor-Id está o no presente en la cabecera.
  - r (reserved): reservados para uso futuro. Deben ser fijados a 0 e ignorados por el receptor.
- AVP Length: 2 octetos. Los campos AVP Code, AVP Flags, AVP Length, Reserved y Vendor-Id no son tenidos en cuenta en esta longitud.

- Reserved: 2 octetos reservados para uso futuro. Deben ser fijados a 0 e ignorados en el receptor.
- Vendor-Id: Está presente si el flag V está a 1. Tiene 8 octetos e indica una asignación IANA para este Vendor.
- Value: cero o más octetos, contiene la información del atributo.

2.3.5. Tipos de mensajes en PANA

Los tipos de mensajes en PANA se resumen en la siguiente tabla:

Message Name	Abbrev.	Message Type	PaC<->PAA	Ref
PANA-Client-Initiation	PCI	1	-------->	7.1
PANA-Auth-Request	PAR	2	<------->	7.2
PANA-Auth-Answer	PAN	2	<------->	7.3
PANA-Termination-Request	PTR	3	<------->	7.4
PANA-Termination-Answer	PTA	3	<------->	7.5
PANA-Notification-Request	PNR	4	<------->	7.6
PANA-Notification-Answer	PNA	4	<------->	7.7

Cuadro 2.1: Mensajes PANA

- PCI: Es enviado para iniciar una sesión iniciada por el PaC, en este mensaje el identificador y el número de secuencia deben ir a 0.
- PAR: Puede ser enviado por el PaC o por el PAA, no puede tener a la vez los bits S y C a uno.
- PAN: Es la respuesta a un PAR, y, del mismo modo, no puede tener los bits S y C a uno al mismo tiempo.
- PTR: Puede ser enviado tanto por el PAA como por el PaC.
- PTA: Es la respuesta a un PTR.
- PNR: Se usa para solicitar una reautenticación o para testear la vida del enlace. Debe tener uno y solo uno de los bits A (Reautenticación) o P (Ping) puesto a uno.

- PNA: Es la respuesta a un PNR, debe tener a uno los mismos bits que PNR.

Para más detalles sobre estos mensajes, consultar el RFC de PANA (5191).

Tipos de AVPs en PANA

Los Tipos de AVPs en PANA se resumen en la siguiente tabla, junto con los mensajes PANA que pueden llevarlos:

	Message Type					
Attribute Name	PCI	PAR	PAN	PTR	PTA	PNR	PNA
AUTH	0	0-1	0-1	0-1	0-1	0-1	0-1
EAP-Payload	0	0-1	0-1	0	0	0	0
Integrity-Algorithm	0	0+	0-1	0	0	0	0
Key-Id	0	0-1	0-1	0	0	0	0
Nonce	0	0-1	0-1	0	0	0	0
PRF-Algorithm	0	0+	0-1	0	0	0	0
Result-Code	0	0-1	0	0	0	0	0
Session-Lifetime	0	0-1	0	0	0	0	0
Termination-Cause	0	0	0	1	0	0	0

Cuadro 2.2: AVPs PANA

El significado de la tabla es:

- 0: El AVP no debe estar presente en el mensaje.
- 0-1: Una o ninguna instancia puede estar presente en el mensaje
- 1: Una instancia debe estar en el mensaje.
- 0+: cero o más instancias pueden estar presentes.

Y el significado de los AVPs:

- AUTH: Se usa para proteger la integridad de los mensajes PANA. Su longitud varía según el algoritmo utilizado.

- EAP-Payload: Se usa para encapsular el mensaje EAP que actualmente esta siendo intercambiado entre el Peer y el autenticador EAP.
- Integrity-Algorithm: El algoritmo que se usará para computar el AUTH
- Key-Id: contiene un identificador MSK. Es de tipo entero de 32 bits y es asignado por el PAA, es único para cada sesión PANA.
- Nonce: Es un número aleatorio usado en las computaciones de las claves criptográficas. La longitud de estos Nonces es escogida basándose en las funciones pseudoaleatorias (PRF) disponibles.
- PRF- Algorithm: Es el algoritmo usado para combinar las funciones pseudo-aleatorias para derivar una PANA-AUTH-KEY.
- Result-Code: Indica si la autenticación EAP ha sido completada correctamente. Hay 3 posibilidades:
  1. PANA-SUCCESS: Tanto la autenticación como la autorización son correctas.
  2. PANA-AUTHENTICATION-REJECTED: La autenticación ha fallado, y en consecuencia también ha fallado la autorización.
  3. PANA-AUTHORIZATION-REJECTED: La autorización falla incluso aunque la autenticación haya sido correcta.
- Session-LifeTime: Número de segundos que quedan hasta que la sesión expire
- Termination Cause: Se usa para indicar la razón por la que una conexión termina:
  1. LOGOUT: El cliente inicia una desconexión.
  2. ADMINISTRATIVE: El cliente no obtiene acceso o es desconectado por temas administrativos.
  3. SESSION-TIME-OUT: Fin del tiempo asignado a la sesión.

2.3.6. Funcionamiento detallado del protocolo

En la presente sección explicaremos el funcionamiento detallado de las distintas fases expuestas en la secciones anteriores.

Fase de autenticación y autorización

Durante este fase de producirá la autenticación y autorización mutua del PaC y el PAA. Puede ser iniciada por cualquiera de los dos extremos:

- Iniciada por el PaC: Cuando el PaC inicia la sesión envía un mensaje PANA-Client-Initation al PAA. El PAA debe responder la petición con un mensaje PANA-Auth-Request.

- Iniciada por el PAA: Si el PAA conoce la dirección IP del PaC puede enviar un mensaje de tipo PANA-Auth-Request no solicitado.

En ambos casos, el PAA elige un identificador de sesión y se lo envía al PaC en el PANA-Auth-Request. Este identificador debe ser transportado por todos los mensajes subsiguientes entre el PaC y el PAA.
Si una vez recibido el PANA-Auth-Request, el PaC quiere establecer la sesión con el PAA, deberá enviar un PANA-Auth-Answer, en otro caso, descarta silenciosamente el "PANA-Auth-Request".
Estos mensajes deben llevar el bit 'S'(Start) fijado a 1.

Llegados a este punto, PANA necesita establecer mediante un método EAP la función pseudoaletoria (PRF) y los algoritmos usados para la derivación de la PANA-Auth-key y el cálculo del PANA-Auth AVP (explicados más adelante).
Para ello, continuará enviado mensajes tipo PANA-Auth-Request y PANA-Auth-Response.
La negociación es de la siguiente manera: El PAA envía los algoritmos que soporta y el cliente responde con los algoritmos elegidos de entre estos algoritmos propuestos por el servidor.
Podría darse el caso de que tanto el PaC como el PAA intenten iniciar al sesión al mismo tiempo, para solucionar esto el servidor debe descartar el primer mensaje del cliente enviado antes de que el enviará su propia invitación.
El resultado de la fase de autenticación es transportado por el último PANA-Auth-Request enviado desde el PAA al PaC y lleva tanto el resultado de la autenticación EAP como el resultado de la autenticación PANA. Debe ser confirmado por medio de un PANA-Auth-Answer. Estos dos últimos mensajes deben tener el flag C (complete) a 1.
En la siguiente figura podemos ver este intercambio de mensajes de manera de tallada.
Como se ha indicado, estamos accediendo a una red segura, por lo que es posible que el PaC deba reconfigurar su IP para ajustarse a las IPs que

```
PaC       PAA  Message(sequence number)[AVPs]
-----------------------------------------------------------------------
   ----->     PANA-Client-Initiation(0)
   <-----     PANA-Auth-Request(x)[PRF-Algorithm,Integrity-Algorithm]
                                        // The 'S' (Start) bit set
   ----->     PANA-Auth-Answer(x)[PRF-Algorithm, Integrity-Algorithm]
                                        // The 'S' (Start) bit set
   <-----     PANA-Auth-Request(x+1)[Nonce, EAP-Payload]
   ----->     PANA-Auth-Answer(x+1)[Nonce] // No piggybacking EAP
   ----->     PANA-Auth-Request(y)[EAP-Payload]
   <-----     PANA-Auth-Answer(y)
   <-----     PANA-Auth-Request(x+2)[EAP-Payload]
   ----->     PANA-Auth-Answer(x+2)[EAP-Payload]
                                         // Piggybacking EAP
   <-----     PANA-Auth-Request(x+3)[Result-Code, EAP-Payload,
                                   Key-Id, Session-Lifetime, AUTH]
                                      // The 'C' (Complete) bit set
   ----->     PANA-Auth-Answer(x+3)[Key-Id, AUTH]
                                      // The 'C' (Complete) bit set
```

Figura 2.12: Intercambio de genérico de mensajes PANA para conseguir una autenticación

pueden entrar en dicha red, en ese caso, en los mensajes enviados por el PAA el bit 'I'(reconfiguración de IP) estará a 1.

Fase de Acceso

Una vez terminada la fase de autenticación y autorización el dispositivo entra en fase de acceso, durante la cual se intercambiará tráfico IP entre los dos extremos. Durante esta fase pueden enviarse mensajes con el bit 'P'(Ping) a uno para probar si el enlace sigue vivo. Tanto el PaC como el PAA pueden enviar estos mensajes. Esto se denomina test de vida.
La tasa de test de vida del enlace ha de estar limitada por la implementación para no sobrecargar el enlaces. No hay una negociación sobre el valor de este temporizador, pero en general, serán fijados a niveles que no sean perjudiciales para las comunicaciones.
Tras una serie de pings no contestados el enlace se dará por caído y la conexión se interrumpirá.

Fase de reautenticación

Sirve para alargar el tiempo de vida de una sesión una vez acabado el tiempo negociado en la fase de autenticación y autorización. Si la reauten-

ticación tiene éxito el enlace volverá a la fase de acceso, de lo contrario la conexión se interrumpirá.

Cuando el PaC inicia una reautenticación envía un PANA-Notification-request con el bit A fijado a uno y con el identificador de sesión que se le había asignado a dicha sesión. El PAA responderá con una PANA-Auth-Request con el bit A a uno para empezar una nueva autenticación.
Si el identificador de sesión que recibe el PAA no corresponde a ninguna de las sesiones que tenía establecida entonces descarta el paquete silenciosamente.

La reautenticación puede ser igualmente iniciada por el PAA, en ese caso, este envía directamente el PANA-Auth-Request con el identificador de la sesión.
La reautenticación no debe reiniciar los números de secuencia.
En la siguiente figura podemos observar este proceso, iniciado por el PaC

```
PaC      PAA  Message(sequence number)[AVPs]
-----------------------------------------------------------------------
 -------->    PANA-Notification-Request(q)[AUTH]
                            // The 'A' (re-Authentication) bit set
 <--------    PANA-Notification-Answer(q)[AUTH]
                            // The 'A' (re-Authentication) bit set
 <--------    PANA-Auth-Request(p)[EAP-Payload, Nonce, AUTH]
 -------->    PANA-Auth-Answer(p)[AUTH, Nonce]
 -------->    PANA-Auth-Request(q+1)[EAP-Payload, AUTH]
 <--------    PANA-Auth-Answer(q+1)[AUTH]
 <--------    PANA-Auth-Request(p+1)[EAP-Payload, AUTH]
 -------->    PANA-Auth-Answer(p+1)[EAP-Payload, AUTH]
 <--------    PANA-Auth-Request(p+2)[Result-Code, EAP-Payload,
                                     Key-Id, Session-Lifetime, AUTH]
                                     // The 'C' (Complete) bit set
 -------->    PANA-Auth-Answer(p+2)[Key-Id, AUTH]
                                     // The 'C' (Complete) bit set
```

Figura 2.13: Intercambio de genérico de mensajes PANA para conseguir una reautenticación

Fase de finalización

Puede ser iniciada por cualquiera de los extremos. Intercambiando los mensajes PANA-Termination-Request y PANA-Termination-Answer se da

por concluida la sesión PANA de manera controlada.

2.3.7. Consideraciones de seguridad

Medidas generales de seguridad en PANA

- Números de secuencia monótonamente crecientes inicializados de manera aleatoria: Este esquema evita el "spoofing"de la red y combinado con los identificadores de sesión (que son únicos para cada sesión) evitan los ataques de tipo replay (una sesión con un identificador rara vez llevará la misma cuenta que otra con otro identificador).
- La red PANA define además los EP (Enforcement Point) que filtran el tráfico enviado por el PaC. En combinación con un método EAP que permita la derivación de claves proporciona protección de autenticación e integridad de los mensajes mediante una PANA SA.
- Esta protección criptográfica evita además los ataques de "hombre en el medio", inserción de mensajes, o modificación o repetición de mensajes. Todo mensaje que no pasa la verificación de AUTH es descartado silenciosamente.

Vulnerabilidades de seguridad

- Intercambio inicial: Esta expuesto a ataques de "spoofing", ya que estos mensajes no están autenticados ni tienen su integridad protegida.
- Métodos EAP: La fortaleza o debilidad del método EAP utilizado puede hacer que el método sea vulnerable a ataques de diccionario. Como vimos anteriormente en el resumen de EAP, la capa del nivel de aplicación, que actúe por encima de EAP será la encargada de evitar que un atacante consiga que se negocien métodos EAP débiles.
- Claves criptográficas: No hay debilidades adicionales aquí, de la MSK se derivan la PANA-AUTH-KEY para la PANA SA única para cada sesión.
- Test de tiempo de vida: Este tipo de mensajes, que obligan al PAA a contestar con el tiempo de vida que le queda a la sesión pueden ser utilizados por un atacante para saturar la red.
  Cuando se utilizan dentro de una PANA SA se evita que se abuse de ellos. Es recomendable enviarlos siempre dentro de este contexto. Puede ser necesario no permitir el uso de este tipo de mensajes en sesiones PANA que no sean SA,

- Terminación temprana de una sesión: Es el mismo caso que el anterior, se recomienda que las terminaciones de sesión se produzcan en el contexto de una PANA SA, para evitar terminaciones de sesión inducidas por un atacante.

2.3.8. Conclusiones

- PANA es un protocolo de nivel inferior para otros protocolos de seguridad, especialmente EAP.

- PANA da soporte para retransmisiones, control de tráfico, control de acceso, etc.

- PANA puede usar material criptográfico, pero debe heredarlo de un protocolo de nivel superior.

2.4. Transport Layer Security Protocol (TLS)

2.4.1. Introducción

El objetivo de TLS[5] es proporcionar integridad y confidencialidad entre dos aplicaciones que se estén comunicando.
Es decir, proporcionar una comunicación tunelada entre ambos extremos. Se trata de un protocolo que proporciona seguridad salto a salto, y no extremo a extremo.
TLS es transparente a los protocolos que viajan por encima de él, y se usa para encapsular otros protocolos de nivel más alto, como por ejemplo EAP. En este sentido, TLS realiza una función similar a la que realiza IPSec[9].
TLS no será usado directamente en este proyecto, sino que se utilizará EAP-TLS[1], que está basado en TLS, aunque aprovechando la estructura que proporciona TLS para transportar métodos de autenticación.
TLS se subdivide en dos partes: TLS Record Protocol y TLS Handshake Protocol. TLS Record Protocol está en el nivel más bajo, justo por encima de un algún protocolo de nivel de transporte, por ejemplo, TCP o de algún otro protocolo de seguridad.

El Record Protocol proporciona dos propiedades básicas a una comunicación:

1. La conexión es privada, esto se consigue usando cifrado de clave privada o simétrica. Se basan en el uso de un secreto compartido.

2. La conexión tiene protegida la integridad. Se utilizan funciones tipo hash seguras.

Y adicionalmente, el Handshake protocol proporciona:

1. Autenticación de cada Peer, usando criptografía asimétrica o de clave pública.

2. La negociación del secreto compartido usado en el Record Protocol es segura.

3. La negociación tiene la integridad protegida, y ningún atacante puede modificar la negociación sin ser detectado.

Los objetivo que se pretenden lograr, por orden de prioridad, con TLS son:

1. Seguridad en la comunicación entre 2 extremos, basada en criptografía.

2. Interoperabilidad entre distintas implementaciones.

3. Extensibilidad según las necesidades futuras.

4. Eficiencia, sobre todo en los cálculos de las operaciones criptográficas, que tienen un alto consumo de CPU.

2.4.2. TLS Record Protocol

Este protocolo se encarga del nivel más bajo de TLS. El protocolo coge los mensajes que van a ser transmitidos, los fragmenta en bloques manejables, opcionalmente comprime los bloques; aplica una función tipo MAC, encripta los datos y transmite el resultado.
Del mismo modo los datos recibidos son desencriptados, verificados, descomprimidos, reensamblados y transmitidos al nivel superior.
El TLS Record Protocol es el encargado de mantener el estado de conexión.

Estados de conexión

Un estado de conexión es un entorno de operación para el TLS Record Protocol. En él se especifican los algoritmos de MAC, de encriptación y de compresión que se están usando.
Los parámetros de seguridad del protocolo se especifican según las siguientes variables:

- Connection end: Indica si la entidad es el cliente o el servidor de la conexión.
- PRF algorithm: Algoritmo para sacar claves a partir del secreto compartido.
- Bulk Encrypthion Algorithm: Algoritmo usado para la encriptación.
- MAC algorithm : Algoritmo para la función MAC.
- Compression algorithm: Algoritmo usado para la compresión de datos.
- Master Secret: secreto compartido de 48 bytes.
- Client Random: Número aleatorio generado por el cliente, de 32 bytes.
- Server Random: Número aleatorio generado por el servidor, de 32 bytes.
- Compression state: Estado actual del algoritmo de compresión.
- Cipher state: Estado actual del algoritmo de encriptación.
- Sequence number: Número de secuencia mantenido por separado para la lectura y la escritura (del mismo modo que en PANA), debe iniciarse a cero[6] en cada nueva conexión e irse incrementando con el paso de los mensajes.

2.4.3. TLS Handshake Protocol

Este es el protocolo encargado de la negociación de la sesión TLS, es decir, es el que se encarga de realizar TLS en sí, a diferencia del Record Protocol que se encargaba de transportarlo.
Una sesión TLS tiene los siguientes parámetros:

- Session Identifier.
- Peer certificate.
- Compression Method.
- Cipher Spec: Especifica la función pseudoaleatoria (PRF) usada para generar el material de claves.

[6]Nótese que esto es distinto a como se hacía en PANA, en PANA este número comenzaba en un valor aleatorio

- Master secret: Secreto compartido de 48 bytes.
- Is resumable: Flag que indica si la sesión puede usarse para crear nuevas sesiones.

Tipos de mensajes de TLS Handshake Protocol

- Change Cipher Spec Protocol: Envía un solo byte de valor 1 encriptado y comprimido. Este mensaje es enviado por cliente y servidor para indicar que los siguientes mensajes serán encriptados con el nuevo algoritmo negociado.
- Alert Protocol: Mensajes de alerta intercambiados, pueden tener distinta severidad, llegando en la mayoría de los casos, a terminar la sesión TLS.
- Mensajes de cierre de conexión: El cliente y el servidor deben saber que la conexión está terminando para evitar ataques de truncamiento. Para ello se envía el siguiente mensaje:
  - Closing Message: Indica que quien lo envía no mandará más mensajes durante la actual sesión. Ambos lados de la conexión deben enviarlo antes de cerrar su flujo de lectura de la conexión.
- Mensajes de error: Cuando un error se detecta, la parte que lo ha detectado envía la alerta a la otra parte. Si el error es fatal, la conexión se cierra y ambos lados deben olvidar los datos (identificador, etc) de la conexión. Si el mensaje es de advertencia, la conexión no se cerrará, pero la parte que lo recibe puede elegir seguir con la conexión normalmente o enviar un error fatal a la otra parte. Hay varios mensajes de error definidos, se recomienda ver el RFC[5] para más detalles.
- Mensajes tipo Hello: Son utilizados para intercambiar las posibilidades de seguridad del cliente y el servidor. Cuando la conexión empieza los protocolos de hash, encriptación y MAC son fijados como nulos y los mensajes de Hello los negocian. Hay varios tipos:
  1. Hello Request: Puede ser enviado por el servidor cuando este quiera y significa que el cliente debe empezar el proceso de negociación de los parámetros. Recibido este mensaje el cliente contestará o lo descartará si ya esta iniciando una sesión o no quiere volver a negociar los parámetros de la misma.

2. Client Hello: Cuando un cliente se conecta por primera vez a un servidor debe enviar éste como primer mensaje. También lo enviará en respuesta a un Hello Request o por iniciativa propia si quiere volver a negociar los parámetros de una conexión. Este mensaje incluye un campo de tiempo de conexión que el cliente desea. El cliente manda una lista con los protocolos que soporta por orden de preferencia en unos "sets de cifrado"(cipher suite) que incluyen protocolos de MAC, de cifrado, tipos de clave y el PRF, el servidor elegirá una de las cipher suites pasadas o si no hay opciones buenas cerrará la conexión y enviará un warning.
3. Server Hello: El servidor responderá con este mensaje a un Client Hello cuando sea capaz de elegir un set de algoritmos para la comunicación, sino responderá con un failure alert.
4. Extensiones: El cliente puede enviar extensiones en su Client Hello, en el ServerHello solo podrán aparecer extensiones que hayan aparecido en el Client Hello. No puede haber más de una extensión del mismo tipo.
   Estas extensiones deben especificar su funcionalidad tanto durante la creación de una nueva sesión como cuando se usan durante una renegociación de una sesión ya existente. Son usadas, en la medida de lo posible, para solventar problemas de seguridad que se encuentren durante los intercambios de mensajes iniciales.

- Server Certificate: El servidor debe enviar este mensaje siempre que el intercambio de claves requiera procesos de autenticación. Debe ir justo después del ServerHello. Si el Client Hello incluía una extensión para el algoritmo de cifrado, entonces estos certificados deben estar construidos con ese algoritmo concreto.

- Mensaje de intercambio de claves del servidor: Este mensaje se envía inmediatamente después del certificado del servidor, solo se envía si el certificado del servidor no contenía información suficiente para permitir al cliente el intercambio de claves maestras. El mensaje, por tanto, contiene información para permitir al cliente intercambiar estas claves maestras. Si en el Client Hello había especificado algún algoritmo de cifrado, el algoritmo de cifrado seleccionado debe ser ese en concreto.

- Certificate Request: Un servidor no anónimo puede solicitar el envío de un certificado por parte del cliente

- ServerHello done: Lo envía el servidor para indicar el fin del ServerHello y sus mensajes asociados. Indica que el servidor ha terminado los pasos

previos al intercambio de claves y que el cliente puede proceder con la fase de intercambio de claves.

- Client Certificate: Es el primer mensaje que el cliente puede enviar después del ServerHello Done, sólo se envía si el servidor requiere un certificado, si el cliente no posee certificados puede enviar un certificado en blanco, ante el cual el servidor podrá o bien continuar con la comunicación sin certificado o bien interrumpirla.
- Client Key Exchange Message: Este mensaje es enviado por el cliente justo después del Client Certificate, si el anterior no fue enviado se convierte en el primer mensaje que se envía. Este mensaje fija el secreto compartido maestro de la comunicación.
- Certificate Verify: Se usa para dar verificación explícita al certificado de un cliente.
- Finished: Se usa para verificar que el intercambio de claves y el proceso de verificación fueron correctos.

Funcionamiento del protocolo

Los pasos seguidos por el TLS Handshake protocol son los siguientes:

- Intercambiar los mensajes Hello para fijar los protocolos e intercambiar los números aleatorios.
- Intercambiar los datos necesarios para permitir al servidor y al cliente aceptar la primera clave.
- Intercambiar certificados y claves para la autenticación mutua.
- Generar el secreto compartido maestro.
- Dar los parámetros de seguridad al TLS Record Layer.
- Verificar al cliente y al servidor que el otro extremo ha calculado los mismos parámetros y que no ha habido ningún atacante en medio de la comunicación.

Un tipo de ataque muy común consiste en que el atacante actuando como hombre en el medio haga seleccionar un método de seguridad poco seguro, este riesgo intenta minimizarse, pero es posible encontrarse con ataques de este tipo. Para minimizar su efecto es importante que los niveles superiores no acepten enviar sus datos por conexiones menos seguras de lo que se requiera. En la siguiente figura tenemos resumido este protocolo:

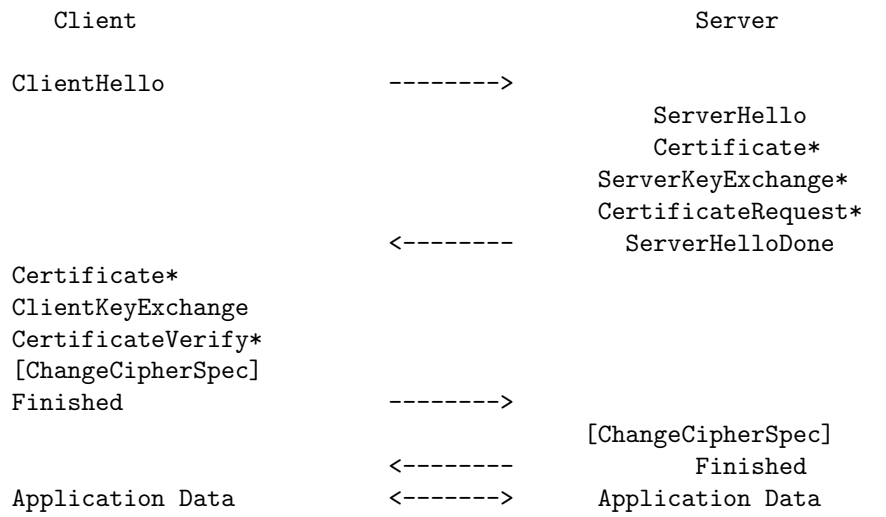


Figura 2.14: Intercambio de mensajes TLS

2.4.4. Conclusiones

- TLS se basa en criptografía asimétrica.
- TLS proporciona seguridad salto a salto.
- TLS se divide en dos subniveles, de forma parecida a como lo hace IPSec[9]

En resumen, TLS es útil para conseguir seguridad salto a salto basada en certificados.

2.5. Diameter Base Protocol

2.5.1. Introducción

Diameter[16] es un protocolo de AAA (Authentication,Authorization and Accounting) pensado para proporcionar sus servicios a redes de tipo IP, incluidas las redes móviles.
Es el protocolo que emplea IMS para sus comunicaciones, y, por lo tanto, tiene gran importancia en el presente proyecto. Los protocolos de autorización y accounting (AAA) tales como RADIUS[3],fueron desarrollados inicialmente para redes de tipo PPP. Con el paso del tiempo y con el crecimiento de internet, estos protocolos se han quedado "pequeños"para las demandas actuales

tanto en complejidad como en densidad. Las nuevas necesidades se resumen en estas:

- Control de caídas: RADIUS no definía ningún mecanismo de Acks, por lo tanto este control dependía de la aplicación, por contra en Diameter sí se definen estos Acks a nivel de transporte.

- Seguridad a nivel de transmisión: RADIUS define un nivel de aplicación con seguridad e integridad que es sólo requerido en los mensajes de respuesta, además, en RADIUS el uso de IPSec está muy limitado. Diameter obliga a que IPSec[9] sea soportado, y TLS es opcional.

- Consideraciones de retransmisión: RADIUS no define una política de retransmisiones y va montado sobre UDP, por lo que estas varían de unas aplicaciones a otras. Por contra, Diameter actúa sobre TCP o sobre SCTP, proporcionando una buena política de retransmisiones.

- Soporte para agentes: RADIUS no define soporte para agentes de red, tales como proxys. Está limitación es corregida en Diameter.

- Soporte para mensajes iniciados por el servidor: En RADIUS, es opcional el soporte para este tipo de mensajes, haciendo complicadas funciones como, por ejemplo, la desconexión iniciada por el servidor. En Diameter es obligatoria la implementación del soporte para mensajes iniciados por el servidor.

- Negociación de capacidades: RADIUS no soporta negociación de capacidades ni flags de obligatorio o no obligatorio en los campos de sus mensajes, este hecho puede hacer que en muchos casos no sea posible la comunicación incluso cuando ambos extremos poseen implementaciones similares. Diameter sí soporta la negociación de capacidades y los flags en sus AVPs.

- Descubrimiento de Peers y configuración: RADIUS requiere que los Peer con los que se comunicará sean especificados de manera manual. Diameter permite el descubrimiento de los mismos por medio de protocolos como DNS.

Diameter surge para cumplir con esta necesidades, que no eran cubiertas por sus antecesores.
Diameter proporciona además las siguientes capacidades:

1. Uso de AVPs.

2. Negociación de capacidades.

3. Notificación de errores.

4. Extensibilidad, a través de la creación de nuevos AVPs, como, por ejemplo, Diameter-EAP, que es una extensión de Diameter.

5. Servicios básicos para la gestión de sesiones.

Los AVPs dan soporte a las siguientes funciones dentro del protocolo Diameter básico:

1. Transporte de la información de autenticación del usuario.

2. Transporte de la información de autorización específica del servicio.

3. Intercambio de capacidades y necesidades referidas a la red en cuanto al uso de recursos (usado para planificar dimensionamientos de capacidades)

4. Intercambio de información sobre el uso de recursos.

5. Redireccionamiento y reenrutamiento de mensajes Diameter según la jerarquía de servidores.

El protocolo Diameter básico proporciona las mínimas capacidades requeridas a un protocolo de tipo AAA. Puede usarse para propósitos de accounting únicamente, o, de manera más común, puede usarse en combinación con una aplicación Diameter.
Diameter es un protocolo Peer-to-Peer, en el sentido de que cualquier nodo puede iniciar la comunicación. Además, se permite el uso de agentes, que son nodos que no autorizan ni autentican los mensajes localmente, esta categoría incluye, por ejemplo, proxys y relays.

2.5.2. Definiciones y vocabulario en Diameter

- Diameter Client: Un cliente de Diameter es un extremo de la red que proporciona control de acceso, es un Diameter client, por ejemplo, un NAS (Network Access Server).

- Diamenter Node: Es cualquier nodo de la red que implementa Diameter.

- Diameter Agent: Es un nodo que implementa Diameter y que redirige las peticiones, por ejemplo, un proxy.

- Diameter Peer: Es un nodo de la red al que, dado un Diamenter Node concreto, ese nodo tiene conectividad directa.
- Diameter Server: Es el nodo que implementa y proporciona las funciones de AAA para un determinado Realm[7].
- Network Access Identifier (NAI): Es una estructura usada por Diameter que contiene [nombre de usuario]@[Realm]. El nombre de usuario es usado para propósitos de autenticación y autorización mientras que el Realm se usa para propósitos de enrutado. Se requiere que los Realm sean únicos y registrados en algún servidor DNS.
- Usuario: La entidad que requiere un servicio, en soporte de la cual Diameter crea peticiones.

2.5.3. Panorámica del protocolo

El protocolo Diameter básico contiene estas funcionalidades:

1. Intercambio de capacidades.
2. Funcionalidad de Watchdog
3. Envío de mensajes.
4. Abandono de la conexión por parte de los Peers.

Las comunicaciones Diameter comienzan con un Peer enviando mensajes al otro Peer. Estos mensajes contendrán los AVPs correspondientes al tipo de aplicación que se está utilizando y además deben contener un AVP con el número de sesión: "Session-Id AVP". Este Session-Id será empleado en todos los mensajes de la sesión para identificar la sesión del usuario. Diameter empleará el -Result-Code AVP- para indicar el resultado de los mensajes que reciba, pudiendo indicar este AVP resultado de éxito, de error o pidiendo que la comunicación continúe.

Todo Peer de Diameter debe soportar el protocolo Diameter básico (Diameter Base Protocol). Adicionalmente pueden soportar cualquier otra aplicación Diameter, pero las aplicaciones soportadas deben estar completas. Se definen dos aplicaciones iniciales para Diameter, -NASREQ- y -Movile Ipv4[18]-. Un servidor de Diameter que no soporte estas 2 aplicaciones se

[7]Realm es traducido como dominio en el presente documento, y ambos son usado indistintamente en el desarrollo del mismo.

definirá como "Diameter X Server"siendo X la aplicación soportada por el mismo. Todos los agentes de redirección, tales como proxys o relays, deben transportar el protocolo de manera transparente entre los extremos que sí procesarán las peticiones.

2.5.4. Nivel de transporte y seguridad en Diameter

Diameter puede ir sobre TCP[14] o sobre SCTP[19][8] como protocolos de nivel de transporte, ambos sobre el puerto 3868. Los clientes sólo tienen que soportar uno de los protocolos, aunque opcionalmente pueden soportar ambos, sin embargo, los servidores deben soportar los dos protocolos. Los Peers pueden iniciar la conexión en otro puerto en el que estén preparados para recibir conexiones pero adicionalmente deben estar preparados para recibir mensajes en el puerto 3868. Cada conexión Diameter utilizará una sola conexión de nivel de transporte, a menos que existan varias instancias del Peer comunicándose simultáneamente.
Si el Peer no indica que protocolo de transporte usará, debe intentarse primero la conexión SCTP, y posteriormente, TCP.
El temporizador Tc se encarga de reintentar las conexiones fallidas, por defecto dicho temporizador estará fijado en 30 segundos.
Si una conexión Diameter recibe mensajes con errores en el nivel de transporte la conexión debe reiniciarse en el nivel de transporte, empleando para ello el bit RST de TCP o un mensaje de tipo ABORT en SCTP.

[8]Stream Control Transmission Protocol (SCTP) es un protocolo de comunicación de capa de transporte desarrollado por el grupo SIGTRAN el año 2000. El protocolo está especificado en la RFC2960, y la RFC3286.
SCTP es una alternativa a los protocolos de transporte TCP y UDP. Provee confiabilidad, control de flujo y secuenciación como TCP. Sin embargo, SCTP opcionalmente permite el envío de mensajes fuera de orden y a diferencia de TCP, SCTP es un protocolo orientado al mensaje (similar al envío de datagramas UDP).
Las ventajas de SCTP son:

- Capacidad de Multihoming, en la cual uno (o dos) de los extremos de una asociación (conexión) pueden tener más de una dirección IP. Esto permite reaccionar en forma transparente ante fallos en la red.
- Entrega de los datos en fragmentos que forman parte de flujos independientes y paralelos, eliminando así el problema de head of the line blocking que sufre TCP.
- Es capaz de seleccionar y monitorizar caminos, seleccionando un camino primario y verificando constantemente la conectividad de cada uno de los caminos alternativos.
- Mecanismos de validación y asentimiento como protección ante ataques por inundación, proveyendo notificación de trozos de datos duplicados o perdidos.

En cuanto a la seguridad, los Clientes Diameter, así como los agentes de movilidad (proxys, relays, etc) y los Network Access Servers (NAS) deben soportar IPsec[9], y además pueden opcionalmente soportar TLS[5], los servidores Diameter deben soportar ambos protocolos.

2.5.5. Identificadores de Aplicación (Application Id)

Cada aplicación Diameter debe tener asignado un identificador de aplicación (Application-Id). Durante la negociación de capacidades, los Peer informarán sobre las aplicaciones que soportan.
En el protocolo básico se definen los siguientes Application-Ids:

- Diameter Common Messages 0
- NASREQ 1
- Mobile-IP 2
- Diameter Base Accounting 3
- Relay 0xffffffff

Adicionalmente, se definen el resto de aplicaciones, empezando desde el número 4. Por ejemplo, se define:

- Diameter-EAP 5

En caso de haber varios servidores de Diameter, los Proxys y Relays son los encargados de encontrar un servidor que soporte la aplicación de un mensaje Diameter determinado, si no hay ninguno disponible se devuelve un error del tipo DIAMETER-UNABLE-TO-DELIVER encapsulado dentro del Result-Code AVP.

2.5.6. Conexión vs Sesión

Es importante distinguir los términos de Conexión y de Sesión dentro del marco de Diameter, para ello, el siguiente gráfico ilustra bastante bien las diferencias.
Es decir, las conexiones se crean entre Peers Diameter y la sesión se crea entre un cliente y un servidor. Cada una de estas sesiones tiene un identificador de sesión único.
Es importante destacar que no hay ninguna relación entre conexiones y sesiones, y que los datos de múltiples sesiones pueden estar viajando a través

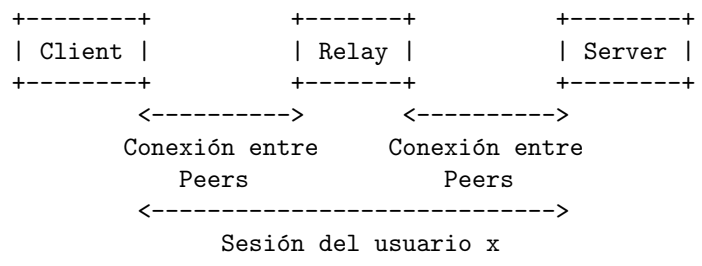


Figura 2.15: Diferencias entre sesión y conexión en Diameter

de la misma conexión.

2.5.7. Los agentes Diameter

En Diameter se definen una serie de agentes que pueden intervenir en el enrutado de los paquetes. Estos agentes son: Relay, Proxy, Redirect y Translation. Los agentes son útiles para el protocolo por los siguiente motivos:

1. Permiten la administración de sistemas y grupos configurables.
2. Permiten concentrar las peticiones dirigidas a un grupo de NAS[9]
3. Pueden añadir información a los mensajes en forma de AVPs
4. Pueden usarse para hacer balanceo de carga.
5. Permiten dirigir las peticiones de autenticación al servidor indicado en redes complejas.

Los agentes, además, mantienen información sobre el estado de la sesión, lo cual permite, entre otras cosas, lo siguiente:

1. Transformaciones de protocolos; por ejemplo de RADIUS a Diameter y viceversa.
2. Limitación de los recursos concedidos a un determinado usuario.

Existen varios tipos de agentes, que serán definidos a continuación.

[9]NAS: Network Access Server. Se trata de un servidor que actúa de intermediario y de punto de acceso a la red.

Agentes de tipo Relay

Los agentes de tipo Relay toman decisiones de enrutamiento basándose en una tabla de Realms soportados y de Peer conocidos.

Los relays permiten agregar peticiones de varios servidores NAS[15] cercanos geográficamente, y son útiles ya que permiten evitar que los NAS se comuniquen con servidores en otros dominios, ya que esto conlleva una serie de configuraciones de seguridad que son costosas. Los agentes de Relay, como ya se ha explicado, sólo pueden modificar información de enrutamiento mediante la inserción o la eliminación de AVPs (pero no modificando los existentes).

Agentes de tipo Proxy

Del mismo modo que los Relays, los Proxys enrutan los mensajes de acuerdo a unas tablas de enrutamiento que se definirán más adelante.
Para poder aplicar estas políticas los Proxys necesitan tener implementadas las aplicaciones de Diameter que vayan a emplear.
Los Proxys tienen capacidad para modificar mensajes y aplicar políticas de control en los mismos.

Agentes de tipo Redirect

Son útiles en escenarios donde el enrutamiento debe realizarse de manera centralizada.
Los agentes de tipo Redirect devuelven al resto de Agentes Diameter la información necesaria para que estos puedan comunicarse directamente con el Peer que desean establecer conexión.
No pueden modificar ningún mensaje.
Como no reciben ningún mensaje de tipo respuesta no necesitan mantener ningún estado de sesión.
Los agentes Redirect no necesitan implementar ninguna aplicación de Diameter ya que sólo toman decisiones de enrutamiento. En la siguiente figura se ilustra un ejemplo de comunicación con agentes Redirect.

Nota: Para leer la figura seguir el orden de los números de los mensajes.

Agentes de tipo Translation

Los agentes de tipo Translation se encargan de dar soporte para la transformación entre varios protocolos, por ejemplo Diameter a RADIUS.

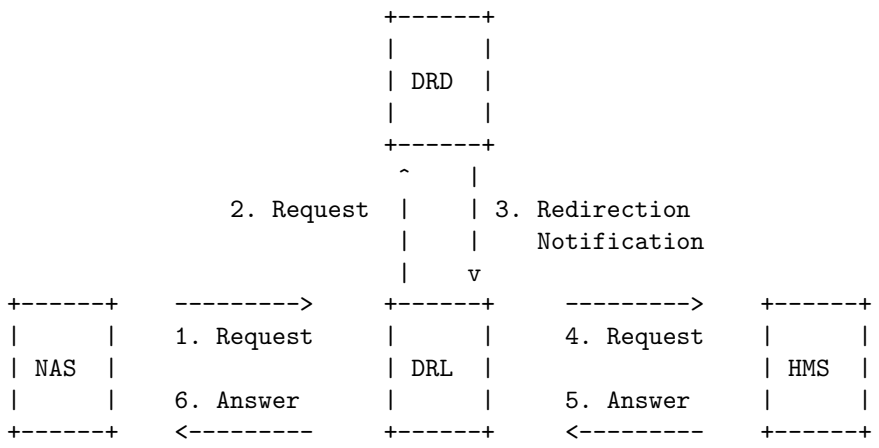


Figura 2.16: Funcionamiento de un agente Redirect en Diameter

Deben mantener información sobre el estado de la sesión y deben implementar al menos la base de los protocolos que traduzca.
En la siguiente figura vemos un ejemplo de su uso.

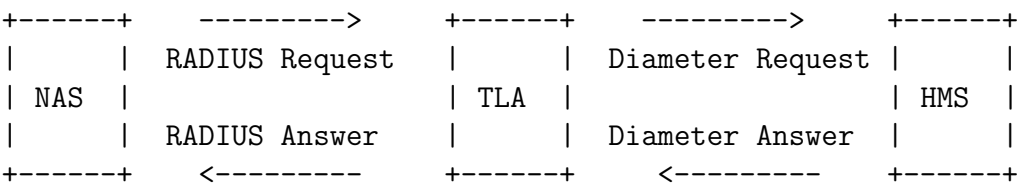


Figura 2.17: Funcionamiento de un agente Translation en Diameter

2.5.8. El enrutamiento en Diameter: Los Realms

El enrutamiento en Diameter se basa en los llamados Realms, que son cadenas de caracteres que identifican a un determinado dominio. Aparecen precedidos por un símbolo @.
Las búsquedas no requieren que haya coincidencia exacta, sino que se basan en la coincidencia más larga empezando desde la derecha.
La tabla de enrutamiento basada en Realms contiene[10]:

- Nombre del Realm: Clave primaria para el enrutamiento.

[10]Durante el desarrollo del proyecto, y, especialmente en el anexo dedicado a los ficheros de configuración, puede observarse como las tablas de enrutamiento usadas implementan esta estructura.

- Identificador de aplicación: La aplicación queda definida por el Vendor Id y la application Id, para todas las aplicaciones definidas en la IETF el Vendor Id debe ser fijado a 0. El application ID dependerá de la aplicación que se esté usando y se usará como segunda clave para el enrutamiento.

- Acción local (Local Action) : Este campo indica como deben ser tratados los mensajes:

  1. LOCAL: Los mensajes se procesan localmente y no es necesario reenviarlos a otro Peer.
  2. RELAY: Los mensajes marcados de esta forma deben ser reenviados al siguiente salto de la red sin modificar ningún AVP que no sea de propósito únicamente de enrutamiento.
  3. PROXY: Los mensajes así marcados serán reenviados al siguiente salto, pero el Proxy puede aplicar políticas de enrutamiento e incluir nuevos AVPs en el mensaje.
  4. REDIRECT: Estos mensajes son reenviados a quien los envió añadiéndoles la identidad del home Server[11].

- Static or Dynamic: indica si la ruta es estática o es descubierta dinámicamente.

- Tiempo de expiración (Expiration Time): Indica el tiempo de vida de una ruta establecida dinámicamente.

Todo agente Diameter debe soportar al menos uno de los modos de Acción Local y la tabla de enrutamiento debe contener una entrada por defecto a la que serán enrutados los mensajes que no puedan ser encaminados siguiendo estos parámetros.

Procesado de los mensajes

Cuando se recibe un mensaje, este es procesado de la siguiente manera:

- Si el mensaje tiene como destino al Servidor local entonces se procesa en dicho servidor.

- Si el mensaje va destina a un servidor con el cual el Peer local tiene conexión directa entonces el mensaje es redirigido a dicho servidor.

[11] Identificador del servidor (Server Identifier): Se trata de uno o más servidores a los que el mensaje está destinado. Los servidores deben aparecer también en la "Peer Table".

- Si el mensaje contiene un Realm que no es el local y al cual el Peer actual no puede acceder directamente entonces el mensaje es reenviado de acuerdo a la tabla de routing.
- Si ninguno de estos procedimientos tiene éxito entonces se devuelve un mensaje de error con el campo Result-Code indicando DIAMETER-UNABLE-TO-DELIVER.

Una vez recibida una respuesta destinada al Peer local, debe buscarse el indicador Hop-by-Hop en una tabla de respuestas pendientes, y dicho identificador debe ser eliminado de la tabla para indicar que la respuesta ya ha sido recibida.

2.5.9. Estructura de los Mensajes en Diameter

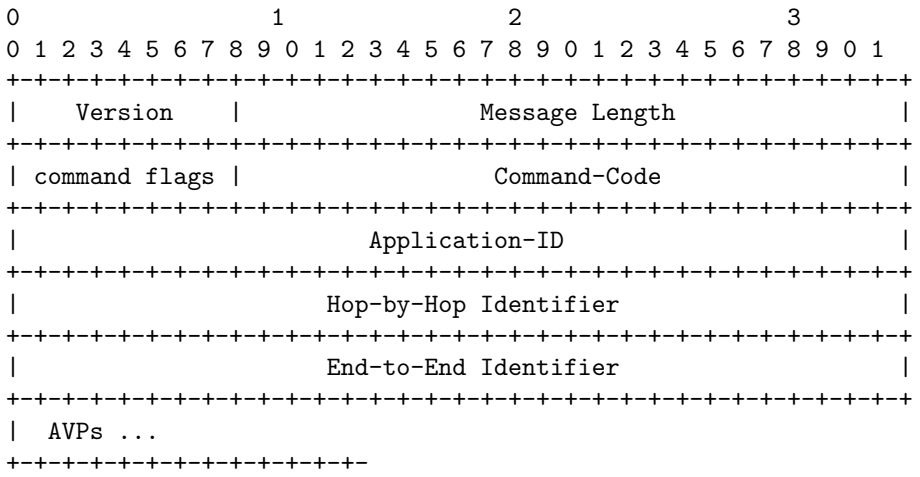
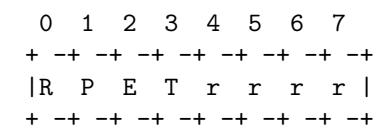


Figura 2.18: Estructura de un mensaje Diameter

- Version: Debe fijarse a 1, ya que la versión actual de Diameter es la 1.
- Message Length: 3 octetos e incluye la longitud de las cabeceras.
- Command Flags: 8 bits según el siguiente esquema:



1. R: Request, si está a 1, el mensaje es una petición
2. P: Proxiable, si está a 1, el mensaje puede ser procesado por un Proxy, Relay o Redirect, si no, el mensaje debe obligatoriamente procesarse localmente.
3. E: Error, si está a 1 el mensaje contiene AVPs que indican errores.
4. T: Mensaje potencialmente duplicado: Indica que el mensaje está siendo retransmitido por no haber recibido un Ack para el mismo, y que, por lo tanto, es posible que se trate de un mensaje duplicado
5. r: reservados. Son flags reservados para uso futuro y deben estar fijados a 0 y ser ignorados por el receptor.

- Command Code: 3 octetos que indican el código del comando Diameter usado.
- Aplication-Id: 4 octetos identificando la aplicación Diameter que es aplicable al mensaje en cuestión.
- Hop-by-Hop identifier: 32 bits sin signo usados para unir las respuestas y las peticiones. Debe ser único para cada conexión y para cada instante de tiempo, además de serlo también para los posibles rearranques de la comunicación. Normalmente se trata un número monótonamente creciente.
- End-to-End Identifier: 32 bits sin signo usados para detectar mensajes duplicados, los 12 primeros bits son una marca de tiempo y los otros 20 bits son un número aleatorio. Este identificador es único y debe perdurar durante los siguientes 4 minutos incluso si hay rearranques en la comunicación. Nunca debe ser modificado por agentes de ningún tipo. Los mensajes duplicados detectados con este campos serán descartados silenciosamente.
- AVP: contiene todos los datos relevantes para el mensaje Diameter en función de la aplicación Diameter que se esté usando.

Commmand Codes (Códigos de Comando)

Determinan la acción que debe llevarse a cabo para un determinado mensaje. Cada mensaje debe contener uno y sólo un Command Code.
En el protocolo Diameter Básico se definen los siguientes Command Codes:

Command-Name(nombre)	Abreviatura	Código
Abort-Session-Request	ASR	274
Abort-Session-Answer	ASA	274
Accounting-Request	ACR	271
Accounting-Answer	ACA	271
Capabilities-Exchange- Request	CER	257
Capabilities-Exchange-Answer	CEA	257
Device-Watchdog-Request	DWR	280
Device-Watchdog-Answer	DWA	280
Disconnect-Peer-Request	DPR	282
Disconnect-Peer-Answer	DPA	282
Re-Auth-Request	RAR	258
Re-Auth-Answer	RAA	258
Session-Termination-Request	STR	275
Session-Termination- Answer	STA	275

Cuadro 2.3: Command Codes en Diameter Base Protocol

AVPs

Cada AVP tiene una estructura y una función que dependen de su definición, no obstante siguen una estructura general:

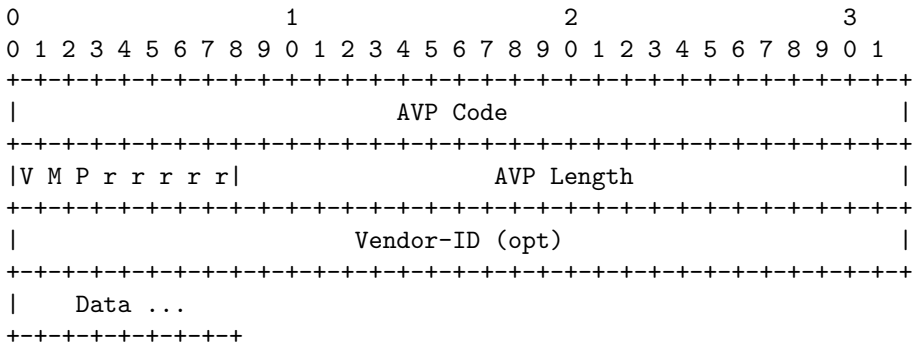


Figura 2.19: Estructura de un AVP en Diameter Base Protocol

- AVP Code: Identifica el AVP de manera inequívoca, cabe destacar que los códigos desde el 0 al 256 están reservados para la compatibilidad con RADIUS.
- Flags:
  1. V : Indica si el campo opcional Vendor-ID está incluido en el AVP.

2. P: Indica la necesidad de encriptación para la seguridad extremo a extremo.
3. M: Indica si el campo es Mandatory, es decir, si el AVP debe ser soportado por los Peers que intervengan en la conexión.
4. r: Son bits reservados para su uso futuro que deben ser fijados a 0 e ignorados por el receptor.

- AVP-Length: longitud del AVP. 3 octetos incluyendo la cabecera.
- Vendor-ID: Este campo es opcional. Sirve para enrutamiento interno dentro del servidor de aplicaciones propias del servidor.
- Data: El campo Data tiene 0 o más octetos y contiene información adicional y específica del AVP.
  Puede tener los siguientes formatos:
  - Octect String: Son datos de longitud arbitraria.
  - Integer 32
  - Integer 64
  - Unsigned 32
  - Unsigned 64
  - Float 32
  - Float 64
  - Grouped: Indica que se trata de una secuencia de AVPs
  - Address: Indica una dirección de 32 bits (IPv4) o de 128 bits (IPv6), los primeros 2 octetos del campo indican el tipo de dirección.
  - Time: Marcas de tiempo en formato NTP[11].
  - ITF-8 String: Es una cadena legible por humanos usando la codificación ISO-10646-1. No deben usarse datos que salgan fuera de dicha codificación.
  - Diameter Identity: Se usa para identificar un determinado nodo Diameter.
  - Diameter URI: Consultar RFC[16].
  - Enumerated: Contiene una lista de valores y su interpretación.
  - IPFilterRule: Se usa para filtrar paquetes según parámetros IP.
  - QoSFilterRule: Se usa para filtrar paquetes según parámetros de calidad de servicio.

AVPs contenidos en Diameter Base Protocol

Hay una gran cantidad de AVPs que forman parte de Diameter Base Protocol, se recomienda consultar el RFC[16] para más detalles.

2.5.10. AVPs y Diameter Commands

Cada comando de Diameter (Diameter Command) puede llevar una serie de AVPs, en el RFC de Diameter[16], hay una tabla que indica qué AVPs puede llevar cada comando y en qué cantidad.
Se recomienda consultar este documento para más detalles.

2.5.11. Diameter Peers: Intercambio de capacidades, desconexión y Wachtdog

Como se ha explicado antes, un Diameter Peer es un nodo de la red al que, dado un Diamenter Node concreto, ese nodo tiene conectividad directa. En esta sección se tratarán la conexión, desconexión, descubrimiento de nuevos Peers e intercambio de capacidades entre Peers.

Conexión de Peers

Un nodo Diameter suele tener varios Peers a los que es capaz de comunicarse, sin embargo, no es económicamente rentable mantener conexión con todos ellos al mismo tiempo.
Un Nodo Diameter debería tener conexión con un máximo de dos Peers de un determinado Realm simultáneamente, que serán conocidos como Peer Primario y Peer secundario. De esta forma, las peticiones destinadas a un determinado Realm serán enrutadas al Peer primario, y, en caso de error de éste, se encaminarán al secundario. También es posible cambiar esta configuración con el objetivo de hacer balanceo de carga.

Hay dos posibles eventos que provocan que un Peer cambie de estado conectado a desconectado o viceversa:

1. Se reciben 3 mensajes correctos de tipo Wacthdog desde el Peer en cuestión: El Peer pasa a considerarse conectado.

2. El Peer ya no es alcanzable desde el protocolo de nivel de transporte: El Peer pasa a considerarse desconectado.

Descubrimiento de nuevos Peers Diameter (Diameter Peer Discovery)

Se soportan varias opciones para el descubrimiento de Peers:

1. Consulta de una lista estática de Peers.
2. Uso de SLP[6][12]
3. Usando NAPTR[10] Records[13]
4. Si no hay ningún NAPTR Record entonces se utilizan otra serie de Records de DNS[12] específicos de servidor o generales.

Un Peer descubierto dinámicamente implicará una actualización en la tabla de Peers.

Intercambio de capacidades

Una vez la conexión a nivel de transporte entre dos Peers está establecida deben realizar un intercambio de capacidades a nivel Diameter. Para ello uno de los Peers, el que pretende realizar la conexión, enviará un mensaje tipo CER indicando sus aplicaciones soportadas.
En caso de no haber ninguna aplicación en común el otro Peer enviará un CEA con un AVP de tipo Result-Code"indicando DIAMETER-NO-COMMON-APPLICATIONS y debe desconectarse la conexión a nivel de enlace.
De la misma forma, si no existe ningún mecanismo de seguridad en común el contenido del Result-Code"será DIAMETER-NO-COMMON-SECURITY, y se producirá una desconexión a nivel de transporte.
Los CERs recibidos de Peers desconocidos deben ser descartados silenciosamente o bien enviar de nuevo un mensaje con Result-Code"DIAMETER-UNKNOWN-PEER.
Los mensajes de tipo CER y CEA no deben ser redirigidos por servidores Proxys, Relays o Redirect. En caso de no poder ser entregados a su destino, se enviará un mensaje con el Result-Code"indicando DIAMETER-UNABLE-TO-DELIVER.

[12]SLP: Service Location Protocol: Se trata de un protocolo de detección de servicios, aunque no se trata de un estándar del IETF

[13]NAPTR: Naming Authority Pointer DNS Resource Rec: Es un Resource Record"de DNS[12] que especifica una regla basada en una expresión regular tal que aplicada a una determinada cadena produce una nueva etiqueta de dominio que será empleada para enrutamiento y mapeado.

Desconexión de Peers

En caso de desconexión en el nivel de transporte, el Peer asumirá que ha habido una perdida de conectividad con el otro Peer, o bien que este ha reiniciado, en estos casos debe reintentarse la reconexión periódicamente.
Hay además una serie de mensajes que son intercambiados por los Peers para llevar a cabo una desconexión:

1. Disconnect-Peer-Request (DPR)
2. Disconnect-Peer-Answer (DPA)

Mensajes de intercambio de capacidades y desconexión

Para llevar a cabo el intercambio de capacidades y la desconexión los Peers se intercambian una serie de mensajes definidos en la sección anterior (CER,CEA, DPR y DPA).
En esta sección se explicará la estructura de estos mensajes.

- Capabilities-Exchange-Request (CER)

  Este mensaje es enviado para el intercambio local de capacidades, su Command- Code es 257 y debe tener el bit R a "1".
  El formato del mensaje es:

```
<CER> ::= < Diameter Header: 257, REQ >
          { Origin-Host }
          { Origin-Realm }
        1* { Host-IP-Address }
          { Vendor-Id }
          { Product-Name }
          [ Origin-State-Id ]
         * [ Supported-Vendor-Id ]
         * [ Auth-Application-Id ]
         * [ Inband-Security-Id ]
         * [ Acct-Application-Id ]
         * [ Vendor-Specific-Application-Id ]
          [ Firmware-Revision ]
         * [ AVP ]
```

Figura 2.20: Mensaje CER de Diameter

- Capabilities-Exchange-Answer (CEA) Es enviado en respuesta a un CER, su Command-Code es 257 y debe tener el bit R a "0". El formato de este mensaje está descrito en la siguiente figura.

```
<CEA> ::= < Diameter Header: 257 >
            { Result-Code }
            { Origin-Host }
            { Origin-Realm }
          1* { Host-IP-Address }
            { Vendor-Id }
            { Product-Name }
            [ Origin-State-Id ]
            [ Error-Message ]
           * [ Failed-AVP ]
           * [ Supported-Vendor-Id ]
           * [ Auth-Application-Id ]
           * [ Inband-Security-Id ]
           * [ Acct-Application-Id ]
           * [ Vendor-Specific-Application-Id ]
             [ Firmware-Revision ]
           * [ AVP ]
```

Figura 2.21: Mensaje CEA de Diameter

- Disconnect-Peer-Request (DPR): Command-Code 282 y bit R a "1", es enviado por un Peer para manifestar si intención de cortar la conexión a nivel de transporte.

```
<DPR>  ::= < Diameter Header: 282, REQ >
           { Origin-Host }
           { Origin-Realm }
           { Disconnect-Cause }
```

Figura 2.22: Mensaje DPR de Diameter

- Disconnect-Peer-Answer (DPA): Command-Code 282 y bit R a "0", se envía en respuesta a un DPR.

El nivel de enlace y los mensajes Watchdog

Debido a las características del protocolo[14] es importante detectar los fallos en el nivel de enlace lo antes posible, para evitar mensajes enviados a Peers que ya no están conectados y retardos innecesarios. Los mensajes de tipo watchdog son utilizados con este fin.

[14]En especial, el gran tamaño que pueden alcanzar las redes.

```
<DPA>  ::= < Diameter Header: 282 >
         { Result-Code }
         { Origin-Host }
         { Origin-Realm }
         [ Error-Message ]
       * [ Failed-AVP ]
```

Figura 2.23: Mensaje DPA de Diameter

- Device-Wacthdog-Request(DWR)
  Command Code 280 y bit R a "1", es un mensaje que no debe ser reenviado a un Peer alternativo. Se envía cuando no hay tráfico enviándose entre los Peers y siguiendo un temporizador.

```
<DWR>  ::= < Diameter Header: 280, REQ >
         { Origin-Host }
         { Origin-Realm }
         [ Origin-State-Id ]
```

Figura 2.24: Mensaje DWR de Diameter

- Device-Watchdog-Answer (DWA) Command Code 280 y bit R a "0", es la respuesta a un mensaje de tipo DWR.

```
<DWA>  ::= < Diameter Header: 280 >
           { Result-Code }
           { Origin-Host }
           { Origin-Realm }
           [ Error-Message ]
         * [ Failed-AVP ]
           [ Original-State-Id ]
```

Figura 2.25: Mensaje DWA de Diameter

2.5.12. Manejo de errores

Hay dos tipos de errores en Diameter: Los errores de protocolo, que son aquellos que ocurren a nivel de protocolo básico, y los errores de aplicación que son los que ocurren a nivel de una determinada aplicación de Diameter. Los AVPs de error sólo deben aparecer en respuestas y sólo en aquellas que

tengan el bit E a "1". El Result-Code AVP indicará el error encontrado. En las siguientes figuras tenemos un ejemplo de error de protocolo y otro de error de aplicación:

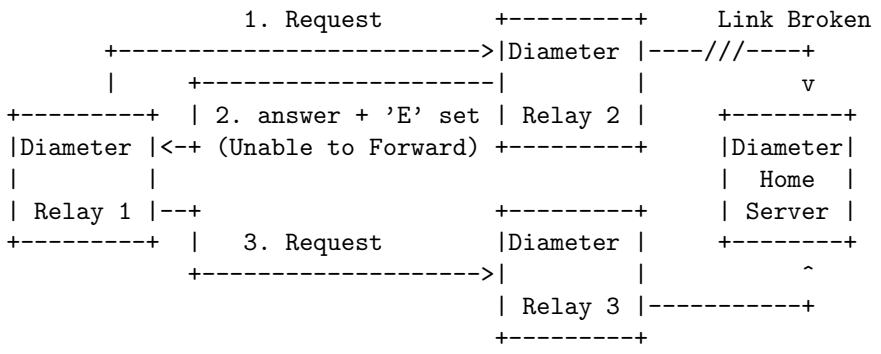


Figura 2.26: Ejemplo de error de protocolo en Diameter

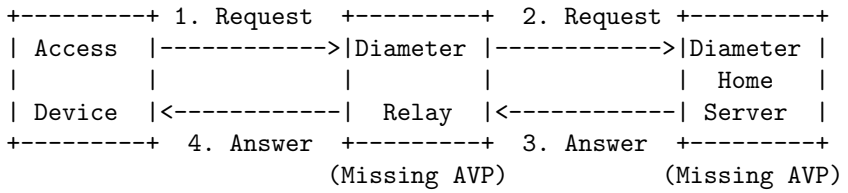


Figura 2.27: Ejemplo de erro de aplicación en Diameter

Para leer las figuras seguir el orden de los mensajes.

Result-Code y Error-Message, su funcionamiento en errores

El Result-Code AVP, como ya hemos visto anteriormente, tiene AVP-Code 268 e indica si una petición tiene éxito o por el contrario da un resultado de error.
Puede ser de alguno de estos tipos:

- 1xxx : Informativo. Indican que la petición no puede ser satisfecha aún y requiere acciones adicionales para satisfacerse. Destaca el DIAMETER-MULTI-ROUND-AUTH (1001) que indica que el mecanismo de autenticación usado requiere más mensajes para completarse.
- 2xxx: Éxito (Success). Hay dos: DIAMETER-SUCCESS (2001) y DIAMETER-LIMITED-SUCCESS (2002), este último indica que se requiere un procesado adicional para dar servicio al usuario.

- 3xxx: Errores de protocolo. Hay 10: DIAMETER-COMMAND-UNSUPPORTED(3001), DIAMETER-UNABLE-TO-DELIVER(3002), DIAMETER-REALM-NOT-SERVED(3003), DIAMETER-TOO-BUSY (3004), DIAMETER-LOOP-DETECTED (3005), DIAMETER-REDIRECT-INDICATION(3006), DIAMETER-APPLICATION-UNSUPPORTED (3007), DIAMETER-INVALID-HDR-BITS (3008), DIAMETER-INVALID-AVP-BITS (3009), DIAMETER-UNKNOWN-PEER (3010)

- 4xxx: Fallos transitorios. Indican que la conexión no puede completarse en ese momento. Hay 3: DIAMETER-AUTHENTICATION-REJECTED (4001), DIAMETER-OUT-OF-SPACE (4002), ELECTION-LOST (4003)

- 5xxx: Fallos permanentes. Existen 17 tipos de errores. Consultar el RFC[16] para más detalles.

Por su lado, el Error-Message AVP es un mensaje de error legible por humanos, no lleva ningún código de error y nunca debe ser obligatorio.

2.5.13. La Máquina de estados de Diameter

La máquina de estados debe ser implementada por todos los Peers de Diameter.

En la siguiente figura tenemos el funcionamiento de dicha máquina. Las distintas acciones están separadas por comas. La "I"indica el Peer que inicia la conexión, mientras que la R"indica el Peer que responde. Los estados de la tabla pueden estar en estado cerrado (Closed), I-Open o R-Open.

estado	eventos	acciones	próximo estado
Closed	Start	I-Snd-Conn-Req	Wait-Conn-Ack
	R-Conn-CER	R-Accept, Process-CER, R-Snd-CEA	R-Open
Wait-Conn-Ack	I-Rcv-Conn-Ack	I-Snd-CER	Wait-I-CEA
	I-Rcv-Conn-Nack	Cleanup	Closed
	R-Conn-CER	R-Accept, Process-CER	Wait-Conn-Ack/ Elect
	Timeout	Error	Closed
Wait-I-CEA	I-Rcv-CEA	Process-CEA	I-Open
	R-Conn-CER	R-Accept,	Wait-Returns

		Process-CER, Elect	
	I-Peer-Disc	I-Disc	Closed
	I-Rcv-Non-CEA	Error	Closed
	Timeout	Error	Closed
Wait-Conn-Ack/ Elect	I-Rcv-Conn-Ack	I-Snd-CER,Elect	Wait-Returns
	I-Rcv-Conn-Nack	R-Snd-CEA	R-Open
	R-Peer-Disc	R-Disc	Wait-Conn-Ack
	R-Conn-CER	R-Reject	Wait-Conn-Ack/ Elect
	Timeout	Error	Closed
Wait-Returns	Win-Election	I-Disc,R-Snd-CEA	R-Open
	I-Peer-Disc	I-Disc, R-Snd-CEA	R-Open
	I-Rcv-CEA	R-Disc	I-Open
	R-Peer-Disc	R-Disc	Wait-I-CEA
	R-Conn-CER	R-Reject	Wait-Returns
	Timeout	Error	Closed
R-Open	Send-Message	R-Snd-Message	R-Open
	R-Rcv-Message	Process	R-Open
	R-Rcv-DWR	Process-DWR, R-Snd-DWA	R-Open
	R-Rcv-DWA	Process-DWA	R-Open
	R-Conn-CER	R-Reject	R-Open
	Stop	R-Snd-DPR	Closing
	R-Rcv-DPR	R-Snd-DPA, R-Disc	Closed
	R-Peer-Disc	R-Disc	Closed
	R-Rcv-CER	R-Snd-CEA	R-Open
	R-Rcv-CEA	Process-CEA	R-Open
I-Open	Send-Message	I-Snd-Message	I-Open
	I-Rcv-Message	Process	I-Open
	I-Rcv-DWR	Process-DWR, I-Snd-DWA	I-Open
	I-Rcv-DWA	Process-DWA	I-Open
	R-Conn-CER	R-Reject	I-Open
	Stop	I-Snd-DPR	Closing
	I-Rcv-DPR	I-Snd-DPA, I-Disc	Closed
	I-Peer-Disc	I-Disc	Closed
	I-Rcv-CER	I-Snd-CEA	I-Open
	I-Rcv-CEA	Process-CEA	I-Open
Closing	I-Rcv-DPA	I-Disc	Closed

R-Rcv-DPA	R-Disc	Closed
Timeout	Error	Closed
I-Peer-Disc	I-Disc	Closed
R-Peer-Disc	R-Disc	Closed

Tipos de Eventos:

- Start: La conexión con el Peer debe iniciarse.
- R-Conn-CER: La conexión de nivel de transporte ha sido iniciada.
- Rcv-Conn-Ack: Ack de nivel de transporte.
- Rcv-Conn-Nack: Nack de nivel de transporte.
- Timeout: Expiración de algún temporizador.
- Rcv-CER: Recibido CER.
- Rcv-CEA: Recibido CEA.
- Rcv-Non-CEA: Recibido un mensaje distinto de CEA.
- Peer-Disc: Evento de desconexión recibido desde el Peer.
- Rcv- DPR: Mensaje DPR recibido.
- Rcv- DPA: Mensaje DPA recibido.
- Win- election: El Peer local ha ganado una elección.
- Send- Message: Mensaje enviado.
- Rcv- Message: Mensaje recibido (distinto de CER, CEA, DPR, DPA)
- Stop: Señal de Stop para la aplicación.

Tipos de acciones:

- Snd-Conn-Req: Conexión de transporte iniciada con el Peer.
- Accept: La conexión asociada con el R-Conn-CER es aceptada.
- Reject: La conexión asociada con el R-Conn-CER es rechazada.
- Proccess-CER: El CER asociado con el R-Conn-CER es procesado.
- Snd-CER: El mensaje CER es enviado al Peer.

- Snd-CEA: El mensaje CEA es enviado al Peer.
- Cleanup: Si se necesita, la conexión puede cerrarse.
- Error: La conexión ha sido desconectada a nivel de transporte.
- Process-CEA: El mensaje CEA asociado con el R-Conn-CER es procesado.
- Snd-DPR: DPR enviado al Peer.
- Snd- DPA: DPA enviado al Peer.
- Disc: El nivel de enlace se desconecta y los recursos son liberados.
- Elect: Se produce una elección.
- Snd-Message: Se envía un mensaje.
- Snd-DWR: Se envía un DWR.
- Snd-DWA: Se envía un DWA.
- Proccess-DWR: Se procesa un DWR.
- Process- DWA: Se procesa un DWA.
- Process: Un mensaje es procesado.

2.5.14. Sesiones de usuario y temporizadores en Diameter

Diameter puede proporcionar dos tipos de servicios a sus aplicaciones y de esta forma, puede proporcionar dos tipos de sesión de usuario distintas. La primera basada en autenticación y autorización (Authentication and Authorization) y haciendo uso opcional de Accounting y la segunda, que sólo hace uso de Accounting.
Para poder distinguir estos tipos de servicios, Diameter utiliza el AVP Auth-Request-Type (código 274) que puede tomar los valores: AUTHENTICATE-ONLY (1), AUTHORIZE-ONLY(2), AUTHORIZE-AUTHENTICATE (3).

Cuando un servidor autoriza a un cliente por un periodo determinado de tiempo es necesario que incluya en sus respuestas un AVP de Authorization Life Time (código 291) indicando el número de segundos que el usuario podrá hacer uso de los recursos, un valor de 0 indica que se requiere una

reautenticación automáticamente según se de acceso al cliente. Ésto es útil, por ejemplo, si se están usando múltiples métodos de autenticación.
También es posible incluir un AVP de Auth-Grace-Period que se combina con el Auth-Life-Time para que el usuario pueda renovar su sesión.
Un dispositivo que no espera enviar una petición de reautorización o una petición de desconexión puede incluir un AVP Auth-Session-State (código 277) con el valor NO-STATE-MAINTAINED(1), en contraposición al otro posible valor de este AVP (STATE-MAINTAINED (0)).

Existen unas máquinas de estados para las sesiones de Authorization, Authentication and Accounting, pero por motivos de espacio no se incluyen en este documento. Consultar el RFC[16] (secciones 8.1 y 8.2)para más información.

Terminación de la sesión

Cuando un usuario decide terminar una sesión de la que posee un estado mantenido es obligatorio enviar un Session Termination Request (STR) al servidor para notificar que la sesión no estará activa durante más tiempo.
Un servidor que reciba este mensaje debe liberar los recursos asociados con el Session-Id especificado y devolver un Session Termination Answer (STA). Por otro lado, el servidor debe también liberar recursos cuando el Authorization-Life-Time y el Grace-Period expiren.

Para mantener una sesión que ya estaba activa se usarán los mensajes Re-Auth-Request (RAR) y Re-Auth-Answer (RAA).

Interrupción de sesiones

El Servidor Diameter puede requerir al servidor de acceso que aborte una determinada sesión enviando un Abort-Session-Request (ASR). Ésto puede deberse a que, por las razones que sea, el servidor decida detener una sesión que previamente autorizó. El servidor de acceso debe contestar con un Abort-Session-Answer (ASA). En caso de estar usando agentes de redirección, tales como proxys o relays, es posible configurar el servidor de acceso para que sólo acepte ASRs de unos determinados agentes.

2.5.15. Conclusiones

- Diameter es un marco de funciones básicas, dentro de las cuales pueden crearse distintas aplicaciones de autenticación.

- Diameter aporta una serie de mensajes útiles para las aplicaciones que llevará, pero no es capaz de realizar autenticaciones por sí mismo.
- Diameter da soporte para la creación y destrucción de sesiones, el intercambio de capacidades, la negociación de protocolos, etc.

En resumen, Diameter Base Protocol, tal y como su nombre indica, proporciona la base necesaria para el uso de aplicaciones de autenticación.

2.6. Diameter-EAP

2.6.1. Introducción

Como hemos visto EAP proporciona un mecanismo para llevar distintos métodos de autenticación. Ahora veremos como puede llevarse EAP sobre Diameter, orientando dicho uso a la comunicación entre un NAS (Network Access Server o servidor de acceso a una red) y un servidor AAA o servidor de autenticación.
En nuestro caso, Diameter-EAP será empleado por el servidor de Acceso para llevar las peticiones del cliente hasta el servidor de IMS. Como veremos durante el desarrollo del proyecto, el servidor de IMS no tenía esta interfaz programada (aunque en un principio se pensó que sí). Por lo tanto, fue necesario programarla para seguir adelante con el desarrollo.

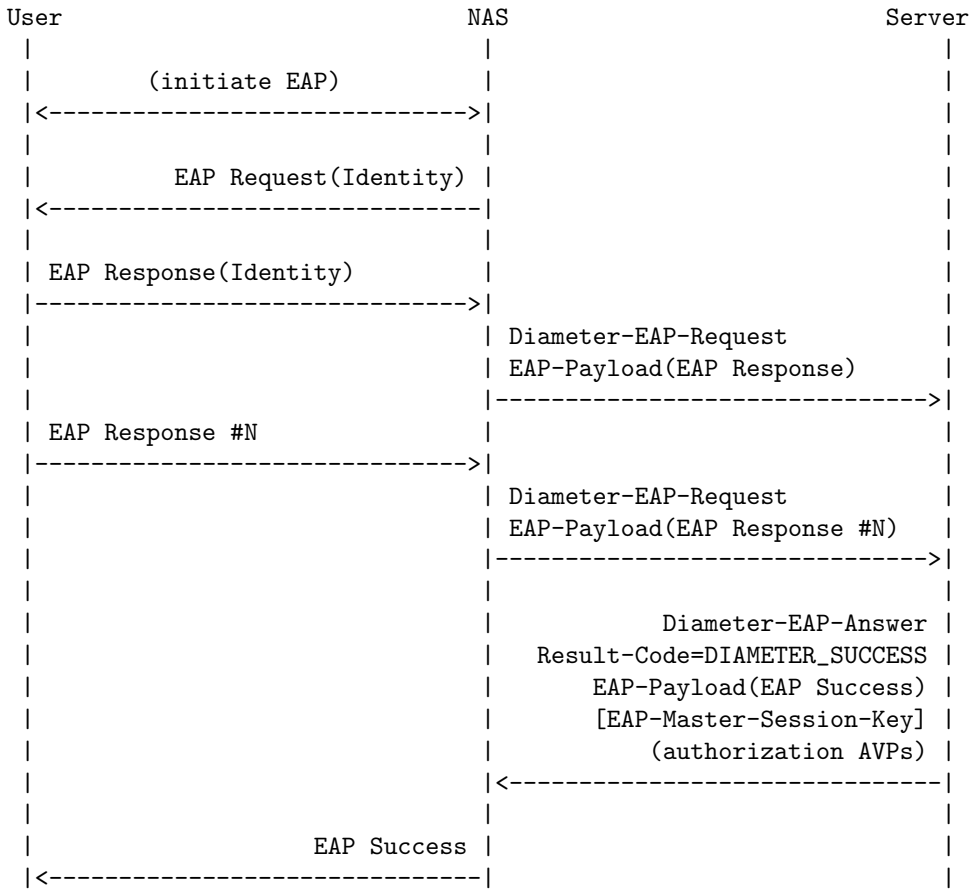
Diameter-EAP[17] es una aplicación que va montada sobre Diameter Base Protocol, es decir, es una ampliación de dicho protocolo.

2.6.2. Soporte de Diameter-EAP sobre Diameter Base Protocol

Los servidores de Diameter que vayan a soportar Diameter-EAP deben contener un Application ID de 5, en caso de no tener dicha aplicación como soportada enviarán un mensaje del tipo DIAMETER-APPLICATION-UNSUPPORTED.
Hay que tener cuidado con los ataques que intentan conseguir que se negocie un tipo de autenticación menos segura en este punto.

2.6.3. Panorámica del Protocolo

En la siguiente figura podemos observar como actúa Diameter-EAP en un caso general:



2.6.4. Códigos de comando añadidos

Los siguientes códigos de comando deben ser soportados por las implementaciones de Diameter que quieran soportar Diameter-EAP:

```
Diameter-EAP Request          Código: 268      Abreviatura: DER
Diameter-EAP Answer           Código: 268      Abreviatura: DEA
```

2.6.5. Mensajes Diameter-EAP: DER y DEA

Diameter-EAP Request: DER

Este mensaje lleva el flag R a 1 y es enviado por el cliente hacia el servidor. Contiene datos EAP. Debe contener un EAP-Payload (código 462) con el estado actual de la autenticación EAP.
El formato del mensaje es el siguiente:

```
<Diameter-EAP-Request> ::= < Diameter Header: 268, REQ, PXY >
                           < Session-Id >
```

```
  { Auth-Application-Id }
  { Origin-Host }
  { Origin-Realm }
  { Destination-Realm }
  { Auth-Request-Type }
  [ Destination-Host ]
  [ NAS-Identifier ]
  [ NAS-IP-Address ]
  [ NAS-IPv6-Address ]
  [ NAS-Port ]
  [ NAS-Port-Id ]
  [ NAS-Port-Type ]
  [ Origin-State-Id ]
  [ Port-Limit ]
  [ User-Name ]
  { EAP-Payload }
  [ EAP-Key-Name ]
  [ Service-Type ]
  [ State ]
  [ Authorization-Lifetime ]
  [ Auth-Grace-Period ]
  [ Auth-Session-State ]
  [ Callback-Number ]
  [ Called-Station-Id ]
  [ Calling-Station-Id ]
  [ Originating-Line-Info ]
  [ Connect-Info ]
* [ Framed-Compression ]
  [ Framed-Interface-Id ]
  [ Framed-IP-Address ]
* [ Framed-IPv6-Prefix ]
  [ Framed-IP-Netmask ]
  [ Framed-MTU ]
  [ Framed-Protocol ]
* [ Tunneling ]
* [ Proxy-Info ]
* [ Route-Record ]
* [ AVP ]
```

Los AVPs etiquetados con {} son obligatorios y los etiquetados con [ ] son opcionales.

2.6.6. Diameter EAP Answer: DEA

Este mensaje debe llevar el bit R a 0 y es enviado por el servidor hacia el Peer por uno de los siguientes motivos:

1. El mensaje es parte de un DIAMETER-MULTI-ROUND-AUTH que

indica que el servidor espera más mensajes EAP del cliente.

2. El servidor comunica al cliente que el proceso de autenticación ha sido realizado correctamente, este mensaje debe llevar obligatoriamente un AVP Result-Code indicando EAP-Success.

3. El servidor comunica al cliente que la autenticación ha fallado. Debe incluirse de nuevo el Result-Code indicando que la autenticación ha fallado.

El formato del mensaje es el que sigue:

```
<Diameter-EAP-Answer> ::= < Diameter Header: 268, PXY >
                          < Session-Id >
                          { Auth-Application-Id }
                          { Auth-Request-Type }
                          { Result-Code }
                          { Origin-Host }
                          { Origin-Realm }
                          [ User-Name ]
                          [ EAP-Payload ]
                          [ EAP-Reissued-Payload ]
                          [ EAP-Master-Session-Key ]
                          [ EAP-Key-Name ]
                          [ Multi-Round-Time-Out ]
                          [ Accounting-EAP-Auth-Method ]
                          [ Service-Type ]
                        * [ Class ]
                        * [ Configuration-Token ]
                          [ Acct-Interim-Interval ]
                          [ Error-Message ]
                          [ Error-Reporting-Host ]
                        * [ Failed-AVP ]
                          [ Idle-Timeout ]
                          [ Authorization-Lifetime ]
                          [ Auth-Grace-Period ]
                          [ Auth-Session-State ]
                          [ Re-Auth-Request-Type ]
                          [ Session-Timeout ]
                          [ State ]
                        * [ Reply-Message ]
                          [ Origin-State-Id ]
                        * [ Filter-Id ]
                          [ Port-Limit ]
                          [ Callback-Id ]
                          [ Callback-Number ]
                          [ Framed-Appletalk-Link ]
                        * [ Framed-Appletalk-Network ]
                          [ Framed-Appletalk-Zone ]
```

```
* [ Framed-Compression ]
  [ Framed-Interface-Id ]
  [ Framed-IP-Address ]
* [ Framed-IPv6-Prefix ]
  [ Framed-IPv6-Pool ]
* [ Framed-IPv6-Route ]
  [ Framed-IP-Netmask ]
* [ Framed-Route ]
  [ Framed-Pool ]
  [ Framed-IPX-Network ]
  [ Framed-MTU ]
  [ Framed-Protocol ]
  [ Framed-Routing ]
* [ NAS-Filter-Rule ]
* [ QoS-Filter-Rule ]
* [ Tunneling ]
* [ Redirect-Host ]
  [ Redirect-Host-Usage ]
  [ Redirect-Max-Cache-Time ]
* [ Proxy-Info ]
* [ AVP ]
```

De nuevo los AVPs etiquetados con {} son obligatorios.

2.6.7. Nuevos AVPs

Diameter EAP añade a los AVPs de Diameter los siguientes nuevos AVPs:

- EAP-Payload (462): Se usa para encapsular el paquete EAP que se está intercambiando actualmente entre el cliente y el servidor. Es de tipo OctetString. El RFC no especifica nada sobre la estructura de este AVP.
- EAP-Reissued-Payload AVP (463)
- EAP-Master-Session-Key-AVP (464): Contiene el material criptográfico para proteger la comunicación entre el cliente y el NAS. Es de tipo OctetString
- Accounting-EAP-Auth-Method-AVP (465): Se usa para tipos expandidos.

Tabla de aparición de los AVPs en DER y DEA

Del mismo modo que sucedía en los comandos de Diameter Base Protocol, los mensajes de tipo DER y DEA pueden llevar una serie de AVPs, en

una cantidad determinada. En el RFC[17] hay una tabla de aparción de los AVPs en estos mensajes. Se recomienda consultar dicho documento para más información.

2.6.8. Consideraciones de seguridad

Trataremos ahora los distintos compromisos de seguridad que pueden aparecer en los escenarios de Diameter-EAP:

Las comunicaciones salto a salto dentro de Diameter-EAP pueden protegerse por métodos de tunelado estándar como TLS o IPsec.
Estos mecanismos proporcionan protección frente a ataques de modificación e inserción de mensajes o hombre en el medio entre otros. Sin embargo, también deben tenerse en cuenta consideraciones de identidad, es decir, decidir si un mensaje Diameter procedente de un cliente autenticado debe ser o no aceptado. Este mecanismo puede basarse en listas de acceso locales, intercambio de certificados, etc.
La protección salto a salto combinada con las consideraciones de identidad proporcionan una buena seguridad frente a atacantes externos, excepto para ataques de denegación de servicio por parte de clientes o NAS que han logrado la autorización y que abusan de sus funciones. En general no es posible controlar este tipo de ataques al 100 %.

2.6.9. Conclusiones

- Diameter-EAP es una aplicación de Diameter Base Protocol, y utiliza todas las funciones definidas en este.
- Diameter-EAP se utiliza para llevar EAP sobre Diameter. Como ya se especificó, se recomienda que EAP[2] no vaya directamente sobre protocolos de nivel de transporte. Diameter-EAP es una de las posibles alternativas.

2.7. EAP-TLS

2.7.1. Introducción

El protocolo EAP es capaz de transportar diversos métodos de autenticación sobre diversos protocolos de transporte. Sin embargo, no ofrece autenticación mutua.
Sería recomendable aportar algún método que proporcione autenticación

mutua y derivación de claves, y aquí es donde es útil EAP-TLS[1], que proporciona derivación de claves y autenticación mutua basada en certificados digitales.

EAP-TLS se basa en TLS (criptografía asimétrica, protección salto a salto...) para proporcionar autenticación mutua a EAP. Es por este motivo por el que su uso será útil en el presente proyecto.

2.7.2. Panorámica del protocolo

La comunicación en EAP-TLS comenzará del mismo modo que en EAP: El cliente enviará su identidad al servidor, y este contestará con un EAP-TLS/start packet, que no es más que un paquete EAP con EAP-Type = EAP-TLS (13) que encapsulará el mensaje Client-Hello de TLS. A partir de este momento la comunicación será como la descrita en el protocolo TLS. En la siguiente figura tenemos un resumen de dicha comunicación:

```
Authenticating Peer           Authenticator
-------------------           -------------
                              <- EAP-Request/
                              Identity
EAP-Response/
Identity (MyID) ->
                              <- EAP-Request/
                              EAP-Type=EAP-TLS
                              (TLS Start)
EAP-Response/
EAP-Type=EAP-TLS
(TLS client_hello)->
                              <- EAP-Request/
                              EAP-Type=EAP-TLS
                              (TLS server_hello,
                              TLS certificate,
                          [TLS server_key_exchange,]
                            TLS certificate_request,
                            TLS server_hello_done)
EAP-Response/
EAP-Type=EAP-TLS
(TLS certificate,
TLS client_key_exchange,
TLS certificate_verify,
TLS change_cipher_spec,
TLS finished) ->
                              <- EAP-Request/
                              EAP-Type=EAP-TLS
                           (TLS change_cipher_spec,
                              TLS finished)
EAP-Response/
```

```
EAP-Type=EAP-TLS ->
                                <- EAP-Success
```

2.7.3. Fragmentación

Un solo TLS Record puede ocupar 16384 bytes, y un mensaje TLS con un certificado puede llegar a ocupar unos 16MB, como consecuencia de esto, EAP-TLS debe proporcionar un mecanismo de fragmentación y defragmentación[15]. Esta funcionalidad no era proporcionada por EAP, pero puede añadirse fácilmente incluyendo un nuevo campos de flags:
Estos flags incluyen:

- La longitud incluida, L
- Si hay más fragmentos, M
- TLS-Start, para saber si es el primer fragmento, S

Por otro lado se incluye un campo longitud de TLS de 4 octetos.
Para más detalles sobre el funcionamiento exacto de la fragmentación, consultar el RFC de EAP-TLS[1].

2.7.4. Verificación de identidades

Como parte de la negociación TLS, habrá un intercambio de certificados entre el servidor y el cliente.
Estos certificados contendrán una identidad, y los mensajes EAP contendrán otra identidad, potencialmente distinta.
No se requiere que estas dos identidades sean idénticas, y será la identidad del certificado la que tenga validez, y la que conseguirá acceso a los servicios[16].

2.7.5. Formato de los mensajes

En este apartado se resume el formato de los mensajes EAP-TLS:

- Code código del mensaje; 1 (Request), 2 (Response)
- Identifier, El identificador de la sesión
- Length, 2 octetos, incluye los campos de Data, Flags, Length, Identifier y Code

[15] Recordemos que EAP no proporciona mecanismos de fragmentación
[16] Durante el desarrollo del proyecto encontraremos este hecho al usar EAP-TLS

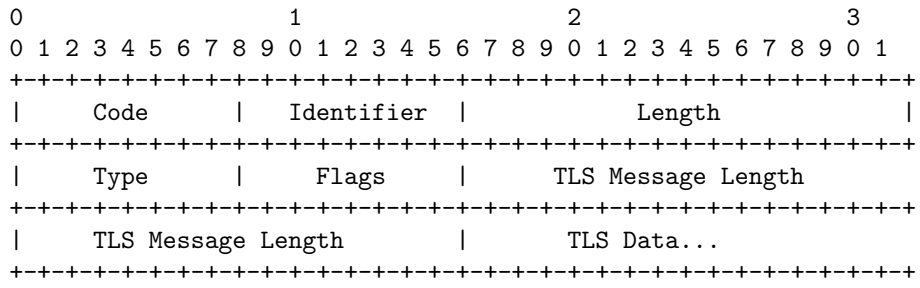
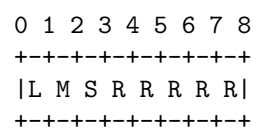


Figura 2.28: Formato de un mensaje EAP-TLS Request

- Type : 13 (EAP-TLS)
- Flags:



  - L Length included
  - M More fragments
  - S TLS Start.
    Estos flags fueron explicados anteriormente.

- TLS Message Lenght, 4 octetos, sólo estará presente si el bit L está a 1. Este campo de 4 octetos contiene la longitud total del mensaje TLS.
- TLS Data Datos de TLS encapsulados.

2.7.6. Conclusiones

- EAP-TLS aporta a EAP capacidades de verificación de identidad.
- EAP-TLS requiere llevar EAP por debajo para funcionar.
- EAP-TLS es una ampliación de EAP, cuyos datos se introducirán en el campo EAP-Payload de EAP.

Parte II

Autenticación con MD5

Capítulo 3

Servidor de Autenticación y NAS

Durante este primer capítulo práctico, se lograrán los siguientes objetivos:

- Explicar las bibliotecas de funciones que van a usarse.
- Explicar, y visualizar de manera práctica, los mensajes de Diameter Base Protocol que se intercambiarán el NAS y el servidor AAA para establecer comunicación.
- Explicar y justificar los motivos por los cuales el cliente PANA que proporcionan las bibliotecas de funciones no es válido.
- Explicar y aplicar los cambios necesarios en el NAS y el servidor AAA para que puedan interactuar con un cliente PANA.

3.1. Figura y comunicación genérica

No podemos continuar con el proyecto sin explicar brevemente el escenario de comunicaciones que pretende construirse.
Este escenario es el siguiente:

La figura contiene estos tres elementos:

1. Cliente PANA: Es el cliente que busca la autenticación. Emplea como protocolo EAP sobre PANA, y un método de autenticación aún sin determinar.
2. NAS (Network Access Server): Es el servidor de Pasarela. Redirige las peticiones del cliente PANA al servidor de autenticación y traduce entre

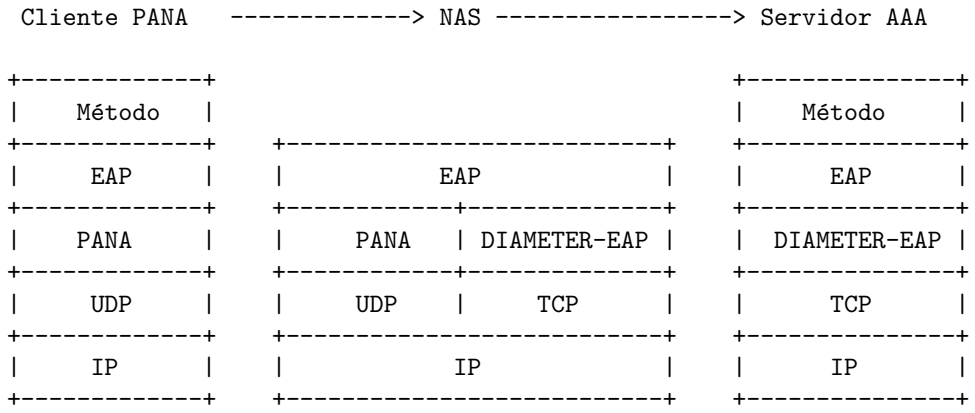


Figura 3.1: Escenario genérico

los protocolos de ambas entidades.
Como ya se explicó en las motivaciones del proyecto, este servidor de pasarela permite que el servidor de AAA pueda soportar multitud del cliente distintos, al no tener que implementar todos los protocolos de dichos clientes. Y , además, permite a los clientes autenticarse frente a servidores con los que de otra manera no podría comunicarse.
Por otro lado, varios NAS permiten reducir la carga en el servidor de autenticación, encaminando las peticiones a otros servidores si el primario está muy saturado.

3. Servidor AAA: Es el servidor de autenticación frente al cual el cliente desea conseguir la autenticación.

Como resumen de la figura: El cliente PANA se comunicará con el NAS por medio de PANA para conseguir autenticación en el servidor de AAA.
El NAS y el servidor de AAA a su vez, usarán Diameter EAP para comunicarse.

3.2. Las bibliotecas de funciones OpenDiameter

En esta sección se pretende explicar la estructura de OpenDiameter y porqué es necesario su uso. La instalación de dichas bibliotecas forman parte de otro apartado de este proyecto (ver anexos).

OpenDiameter es un conjunto de bibliotecas de código abierto, programadas en c++, que contienen implementaciones de los siguientes protocolos:

- Diameter Base Protocol.
- Diameter-EAP.
- Diameter MIP4.
- Diameter NASREQ.
- EAP.
- PANA.

Y adicionalmente y de manera opcional:

- EAP-TLS.

Existen otras implementaciones de estos protocolos, entre las que se puede destacar WireDiameter, sin embargo, ésta última implementación, pese a ser de código abierto lleva sin actualizarse desde el año 2004.
En cuanto al resto de implementaciones encontradas, se trata de versiones que no son de código abierto.

Además de estas bibliotecas en código abierto, OpenDiameter ofrece 3 demonios (programas ejecutables), en teoría, operativos:

- nasd : Es un NAS (Network Access Server) con dos interfaces, una de ellas PANA y la otra Diameter-EAP.
- aaad: Se trata de un servidor de AAA, tiene una interfaz Diameter-EAP.
- pacd: Es un cliente PANA, capaz de soportar 3 métodos de autenticación distintos: MD5, Archie y TLS.

Vemos que los tres demonios son adecuados para el escenario que nos gustaría construir.

3.2.1. Configuración del NAS y del servidor AAA

Para conseguir la comunicación de estas dos entidades es necesario un largo proceso de compilación, descarga y modificación de ficheros. No es tema del presente capítulo explicar dicha configuración, que se encuentra convenientemente explicada en los apéndices.
Desde este punto, se considera que el servidor AAA y el NAS se comunican correctamente y que los ajustes están correctamente establecidos.

3.2.2. intercambio de capacidades y Watchdog

Trazas de pantalla

Con los ajustes explicados en los apéndices, el nasd y el aaad realizarán un intercambio de capacidades y empezarán a enviarse mensajes de tipo watchdog.
Las trazas obtenidas por pantalla para el intercambio de capacidades están indicadas en la siguiente figura (3.2).

```
(16293|3046546320) Peer Capabilities
(16293|3046546320)              Hostname : localaaa.localdomain1.net
(16293|3046546320)                 Realm : localdomain1.net
(16293|3046546320)               Host IP : type=1, 127.0.01
(16293|3046546320)              VendorId : 41
(16293|3046546320)          Product Name : OpenDiameterPeer
(16293|3046546320)            Orig State : 0
(16293|3046546320)  Vendor Specific Id : (16293|3046546320)
(16293|3046546320)  Vendor Specific Id : (16293|3046546320)
(16293|3046546320)  Vendor Specific Id : (16293|3046546320)
(16293|3046546320)  Vendor Specific Id : (16293|3046546320)
(16293|3046546320)  Vendor Specific Id : (16293|3046546320)
(16293|3046546320)            Inband Sec : 0
(16293|3046546320)          Firmware Ver : 0
(16293|3046546320) *** Local capabilities accepted by Peer ***
```

Figura 3.2: Intercambio de capacidades entre el NAS y el servidor AAA en local

Y para el caso del Watchdog tenemos la figura 3.3.

```
Watchdog msg from [localaaa.localdomain1.net.localdomain1.net],
 state=1207434903, time=1207435199
```

Figura 3.3: Watchdog entre el NAS y el servidor AAA en local

Las trazas por pantalla son significativas, pero realizaremos un análisis más profundo del tráfico intercambiado entre ambas entidades por medio de el programa Wireshark.

Trazas de Wireshark y análisis de los mensajes

- Intercambio de capacidades:

  Wireshark nos muestra los siguientes mensajes:

```
Flags = 10000000 (Sólo el flag "Request" a 1)
Command Code = 257
ApplicationId = 0 (Diameter Common Messages)
AVPs:
Origin-Host (264) : localaaa.localdomain1.net
Origin-Realm (296): localdomain1.net
Host-IP-Address (257) : 127.0.0.1
Vendor-Id (266) : 0
Product-Name (269) : Open Diameter
Supported-Vendor-Id (265) : 0
Supported-Vendor-Id (265) : 1
Origin-State-Id : 1240325465
Auth-Application-Id (258) : 1 (NASREQ)
Auth-Application-Id (258) : 2 (Mobile IP v4)
Auth-Application-Id (258) : 5 (Diameter EAP)
Acct-Application-Id (259) : 3 (Base Accounting)
Firm-Ware-Revision (267) : 1
```

Si comparamos con el RFC de Diameter:

```
<CER> ::= < Diameter Header: 257, REQ >
                  { Origin-Host }
                  { Origin-Realm }
               1* { Host-IP-Address }
                  { Vendor-Id }
                  { Product-Name }
                  [ Origin-State-Id ]
                * [ Supported-Vendor-Id ]
                * [ Auth-Application-Id ]
                * [ Inband-Security-Id ]
                * [ Acct-Application-Id ]
                * [ Vendor-Specific-Application-Id ]
                  [ Firmware-Revision ]
                * [ AVP ]
```

Comprobamos que el mensaje está correctamente construido, tanto en flags, como en AVPs.
Para el caso de la respuesta tenemos algo muy similar:

```
Flags = 00000000 (El flag "R" a 0 puesto que es una respuesta)
Command Code = 257
ApplicationId = 0 (Diameter Common Messages)
AVPs:
Result-Code (268) : DIAMETER_SUCCESS (2001)
Origin-Host (264) : localaaa.localdomain1.net
Origin-Realm (296): localdomain1.net
Host-IP-Address (257) : 127.0.0.1
Vendor-Id (266) : 0
```

```
Product-Name (269) : Open Diameter
Supported-Vendor-Id (265) : 0
Supported-Vendor-Id (265) : 1
Origin-State-Id : 1240325465
Auth-Application-Id (258) : 1 (NASREQ)
Auth-Application-Id (258) : 2 (Mobile IP v4)
Auth-Application-Id (258) : 5 (Diameter EAP)
Acct-Application-Id (259) : 3 (Base Accounting)
Firm-Ware-Revision (267) : 1
Inband-Security-Id (299) : 0 (NO_INBAND_SECURITY) -> No estamos usando
                                                    ni TLS ni IPsec
```

Comprobamos con el RFC:

```
<CEA> ::= < Diameter Header: 257 >
                { Result-Code }
                { Origin-Host }
                { Origin-Realm }
             1* { Host-IP-Address }
                { Vendor-Id }
                { Product-Name }
                [ Origin-State-Id ]
                [ Error-Message ]
              * [ Failed-AVP ]
              * [ Supported-Vendor-Id ]
              * [ Auth-Application-Id ]
              * [ Inband-Security-Id ]
              * [ Acct-Application-Id ]
              * [ Vendor-Specific-Application-Id ]
                [ Firmware-Revision ]
              * [ AVP ]
```

Cabe destacar que se añade el AVP 268 (Result Code) para indicar que el intercambio de capacidades es correcto.

- Watchdog

```
Flags = 10000000 (Sólo el R (Request) a 1)
Command-Code = 280 (Device-Wachtdog)
Application-Id = 0 (Diameter Common Messages)
AVPs:
Origin-Host (264) : localnas.localdomain2.net
Origin-Realm (296): localdomain2.net
Origin-State-Id : 1240325465
```

Comparando con el RFC:

```
<DWR>  ::= < Diameter Header: 280, REQ >
           { Origin-Host }
           { Origin-Realm }
           [ Origin-State-Id ]
```

Y en cuanto a la respuesta:

```
Flags = 10000000 (Sólo el R (Request) a 1)
Command-Code = 280 (Device-Wachtdog)
Application-Id = 0 (Diameter Common Messages)
AVPs:
Result-Code (268):  DIAMETER_SUCCESS (2001)
Origin-Host (264) : localaaa.localdomain1.net
Origin-Realm (296): localdomain1.net
Error-Message (281) : Succesfull Device Watchdog
Origin-State-Id : 1240325465
```

En el RFC:

```
<DWA>  ::= < Diameter Header: 280 >
           { Result-Code }
           { Origin-Host }
           { Origin-Realm }
           [ Error-Message ]
         * [ Failed-AVP ]
           [ Original-State-Id ]
```

El mensaje es igualmente correcto.
Cabe destacar que de momento no se ha empleado ningún mensaje perteneciente a Diameter-EAP, sino únicamente mensajes de Diameter Base Protocol.

3.3. El cliente PANA de OpenDiameter

Una vez conectados en servidor NAS y el servidor AAA vamos a intentar conectar el cliente PANA a la red, para terminar de construir el escenario propuesto.

3.3.1. pacd, cliente PANA de prueba

La distribución de OpenDiameter proporcionaba un cliente PANA de prueba, llamado pacd.
El primer pasó fue intentar hacer funcionar dicho cliente.

Configuración del pacd

El primer problema en la configuración del pacd lo encontramos al intentar buscar los ficheros de configuración, ya que estos ni siquiera existen.
Para obtener el directorio /config del cliente PANA podemos usar el svn o subversion, que nos permite acceder al repositorio de versiones alojado en el servidor de OpenDiameter. Para ello basta con teclear:

```
svn co https://Diameter.svn.sourceforge.net/svnroot/Diameter/
                        cplusplus/applications/pana/config
```

Con esta instrucción tendremos los siguientes ficheros:

- pana_setup.xml Fichero de configuración general, indica dónde están los demás ficheros.
- pana_dictionary.xml Fichero de diccionario, para construir y parsear mensajes.
- pana_pac.xml Fichero de configuración del cliente, contiene datos como la dirección del servidor PANA.

Una vez realizada esta instrucción encontraremos un nuevo problema: El fichero de diccionario está completamente desactualizado. Por suerte, este fichero es idéntico al fichero de diccionario PANA del nasd, así que, simplemente tendremos que sustituir el fichero pana_dictionary.xml por el nasd_pana_dictionary.xml y, posteriormente, cambiarle el nombre.
Dentro de dicho fichero haremos un cambio más en la segunda linea, para apuntar al fichero .dtd correcto:

```
<!DOCTYPE dictionary SYSTEM "pana_dictionary.dtd">
```

El cliente ya apunta al puerto 3002 del nasd, que es precisamente dónde el nasd está escuchando la conexiones PANA, si quisiera cambiarse este hecho bastaría con modificar el puerto dentro del fichero pana_pac.xml.

Con estos cambios el cliente ya es capaz de comunicarse con el NAS, sin embargo, ni el NAS ni el servidor AAA están preparados para actuar correctamente sobre las peticiones del cliente.

Cambios en el NAS y el AAA para interactuar con el cliente PANA

para conseguir que el cliente PANA sea correctamente procesado por el NAS y el AAA necesitamos dos cosas:

1. Que el NAS sea capaz de enrutar los mensajes hacia el AAA.

2. Que el AAA pueda darle o no acceso en función de su nombre y usuario y su clave, es decir, que tenga al cliente dentro de su base de datos.

Para conseguir estos dos puntos haremos lo siguiente:

1. Dar soporte a Diameter-EAP en el NAS y el servidor AAA.
   Hasta ahora el NAS y el servidor AAA sólo han intercambiado mensajes del protocolo Diameter básico, Diameter-EAP está fuera de dicho protocolo, así que tendremos que habilitarlo en ambos extremos. Para ello añadimos en los fichero nasd_Diameter_eap.xml y aaad_Diameter_server.xml la siguiente linea:

```
<auth_application_id>5</auth_application_id>
```

2. Crear un cliente de prueba en la base de datos del servidor AAA.
   Para ello dentro del fichero aaad_user_db.xml añadimos las siguientes líneas:

```
+    <user_entry>
+       <name_match>testuser@localdomain1.net</name_match>
+       <eap_method>md5</eap_method>
+       <md5>
+          <password_type>flat</password_type>
+          <secret>12345</secret>
+       </md5>
+    </user_entry>

     <user_entry>


-       <name_match>user</name_match>
+       <name_match>dont_match_anyone</name_match>
```

3. Añadir la ruta al NAS para que pueda enrutar el cliente hacia el AAA.
   Para ello, en el fichero nasd_diameter_eap.xml añadimos la ruta siguiente:

```
<route>
    <Realm>localdomain1.net</Realm>
    <role>1</role>
    <redirect_usage>0</redirect_usage>
       <application>
       <application_id>5</application_id>
          <vendor_id>0</vendor_id>
          <Peer_entry>
             <server>localaaa.localdomain1.net</server>
             <metric>1</metric>
             </Peer_entry>
        </application>
</route>
```

El resto de rutas pueden eliminarse para tener una mayor claridad en el fichero, ya que no serán usadas.

4. Corregir las inconsistencias entre los fichero de diccionario del NAS y el servidor AAA.
   Dentro del AVP User-Name encontramos una inconsistencia que hará que el NAS y el AAA den errores de análisis de los mensajes.
   En uno de ellos ha sido declarado como "Mandatory"(bit M a 1) y en el otro no (bit M = 0).
   Para corregir esta inconsistencia, en el fichero nasd_diameter_eap.xml, modificamos la línea 734 de tal manera que pase de:

```
<avp name="User-Name" code="1" mandatory="must">
  <type type-name="UTF8String"/>
```

   A esto otro:

```
<avp name="User-Name" code="1">
  <type type-name="UTF8String"/>
```

Con estos cambios ya podemos hacer las pruebas del escenario propuesto.

Pruebas y resultados con el pacd

Para probar el sistema completo, desde 3 terminales distintos, ejecutamos:

```
./nasd
./aaad
./pacd -f ./config/pac_setup.xml
```

Los resultados obtenidos no son los esperados, la comunicación siempre se interrumpe tras enviar el pacd un mensaje de tipo Nak[1].

Ya que a estas alturas no comprendemos del todo el código del cliente y además nos gustaría tener un control mayor sobre sus funcionalidades, decidimos implementar un nuevo cliente PANA, basándonos en el pacd, que termine con la autenticación.
A pesar de ello, el trabajo realizado sobre el pacd no es inútil; ya que todos los ajustes de configuración son válidos para futuros clientes.

[1]Este tipo de mensajes se producen al enviar el servidor un método de autenticación al Peer que éste no comprende, revisando la configuración observamos que ambos extremos están configurados para trabajar con MD5.

3.4. Conclusiones

Para concluir, después de este primer capítulo, tenemos un NAS y un servidor AAA correctamente comunicados (en un escenario local, es decir, dentro de la misma máquina), y a la espera de un cliente PANA que desee autenticarse en el servidor de AAA a través del NAS.
Hemos explicado los mensajes que el NAS y el servidor AAA se intercambiarán en ausencia de tráfico y los ajustes que necesitan para enrutar el cliente.
El siguiente paso es desarrollar un cliente PANA que complete la autenticación.

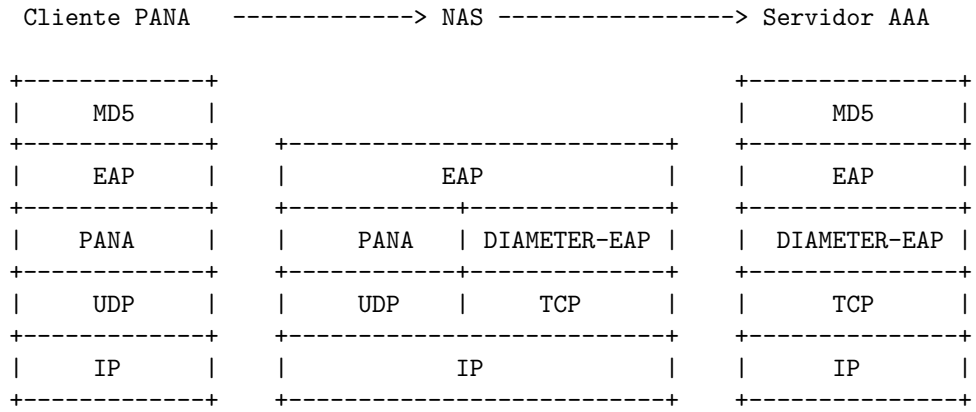
Capítulo 4

Cliente PANA : Creación y autenticación en local

Como hemos visto, hasta ahora, tenemos funcionando las comunicaciones entre el NAS y el servidor AAA, ambos en un escenario local, a través de Diameter-EAP. Nos gustaría introducir un cliente PANA que se comunicara con el NAS, a través de PANA, y se autenticara a la vista del servidor AAA.

4.1. Objetivos. Esquema de comunicación

El primer paso es decidir el método de autenticación que queremos utilizar, en este caso, elegimos MD5[20], por ser muy simple y por estar ya implementado en OpenDiameter.
Con este método, el esquema de la comunicación, resumido, se encuentra en la siguiente figura.



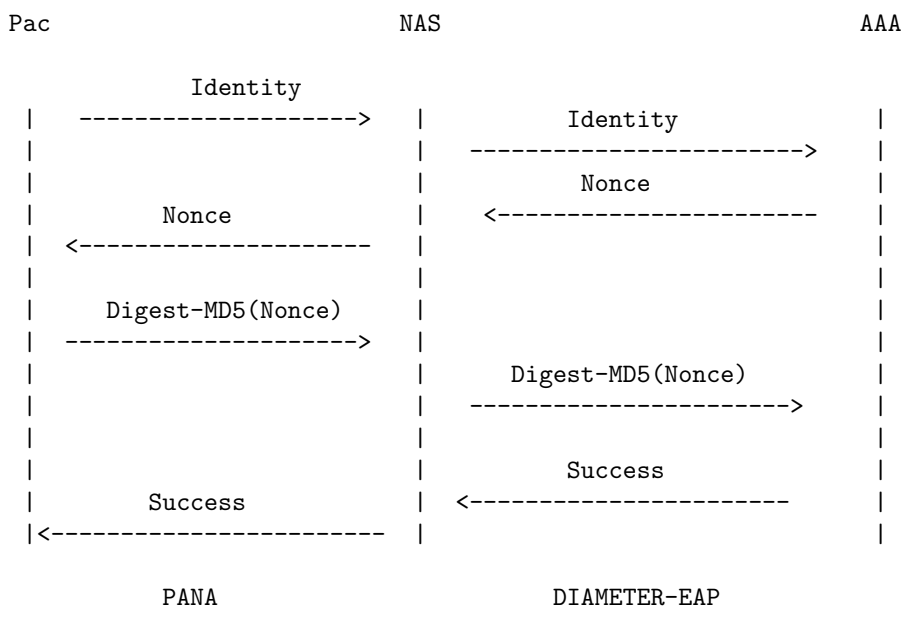


Figura 4.1: Esquema resumido de los mensajes del primer cliente PANA

El funcionamiento es el siguiente: El cliente envía al NAS su identidad, y este a su vez se la reenvía al servidor AAA. Éste (el servidor AAA) busca en su base de datos (en este caso, se trata de un fichero xml) al cliente. Al encontrar que se trata de un cliente que espera autenticarse con MD5[20], le contesta con un nonce, que el cliente deberá codificar con su clave[1] y reenviar de nuevo al servidor (Digest-MD5[20]). Por su parte, el servidor llevará a cabo este mismo procesamiento sobre el número aleatorio que ha enviado al cliente. Esto es posible gracias a que el servidor conoce la clave privada del usuario. Por último, el servidor, al comprobar que el nonce procesado por el cliente es el mismo que el procesado por el mismo, enviará un mensaje de Éxito o Success. En caso contrario enviaría un Failure o Fracaso.
Como se ve en la figura, el NAS sólo actúa de traductor entre PANA y Diameter-EAP, sin hacer ningún procesamiento más allá de funciones de análisis o de seguimiento de la secuencia de mensajes, el NAS no realiza ningún procesamiento de los datos a nivel EAP.

Un dato importante en este esquema es que, en el procesado MD5, aunque se hable de nonce procesado, en realidad lo que se está tomando como entrada para MD5 es una concatenación de: (id de la sesión) + (password del usuario) + nonce.

[1]El cliente codifica (id de la sesión) + password + nonce aplicando el algoritmo MD5.

4.2. Máquina de estados del cliente PANA

Durante su funcionamiento, al igual que sucede en la mayoría de los protocolos de comunicaciones, el cliente PANA seguirá una máquina de estados.
La máquina de estados que implementará el cliente PANA es la máquina de estados básica para PANA[21].

Los posibles estados, por los que el cliente se moverá en función de los mensajes que reciba y los eventos de temporización son:

1. INITIAL: Es el estado inicial de PANA.
2. WAIT_PAA: El cliente espera un mensaje del PAA para continuar, de una forma o de otra, dependiendo del contenido de dicho mensaje.
3. WAIT_EAP_MSJ: El cliente espera un mensaje del nivel superior, es decir, a nivel EAP.
4. WAIT_EAP_RESULT: Se espera un mensaje de resultado a nivel EAP.
5. WAIT_EAP_RESULT_CLOSE: La conexión va a cerrarse a nivel PANA, se espera el resultado a nivel EAP únicamente por motivos de notificación.
6. OPEN: El canal entre el cliente y el servidor se ha establecido con éxito.
7. WAIT_PNA: Se espera un evento de reconexión o de test de conectividad.
8. SESS_TERM: En este último estado se espera que se produzca un fin de la sesión.

La máquina de estados es como sigue:

```
------------------------------
Estado: INITIAL (Estado inicial)
------------------------------
Initialization Action:

Nonce no enviado
Contador de retransmisiones 0
Timer no activado

Condición de cambio             Acción                Estado siguiente
-----------------------+--------------------------+-----------------
```

```
                - -    Iniciado por el PaC   - -
Fallo de                Tx:PCI[]();                    INITIAL
Temporizador            RtxTimerStart();
al inicio               SessionTimerReStart
                        (FAILED_SESS_TIMEOUT);
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

- - - - - - -(Iniciado por el PAA, not optimized) - - - - -
x:PAR[S] &&             EAP_Restart();                 WAIT_PAA
No Existe un campo       (FAILED_SESS_TIMEOUT);
EAP-Payload              if (existía una SA)
                           Tx:PAN[S]("PRF-Algorithm",
                                "Integrity-Algorithm");
                         else
                             Tx:PAN[S]();
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

- - - - - - - -(Iniciado por el PAA, optimized) - - - - - -
Rx:PAR[S] &&            EAP_Restart();                 INITIAL
PAR.exist_avp           TxEAP();
("EAP-Payload") &&      SessionTimerReStart
Vence un temporizador    (FAILED_SESS_TIMEOUT);

Rx:PAR[S] &&            EAP_Restart();                 WAIT_EAP_MSG
PAR.exist_avp           TxEAP();
("EAP-Payload")         SessionTimerReStart
                        (FAILED_SESS_TIMEOUT);
                        if (generate_pana_sa())
                            Tx:PAN[S]("PRF-Algorithm",
                              "Integrity-Algorithm");
                        else
                            Tx:PAN[S]();
EAP_RESPONSE            if (generate_pana_sa())    WAIT_PAA
                             Tx:PAN[S]("EAP-Payload",
                              "PRF-Algorithm",
                              "Integrity-Algorithm");
                        else
                            Tx:PAN[S]("EAP-Payload");
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


---------------
Estado: WAIT_PAA
---------------

Condición de cambio           Acción                    Estado siguiente
----------------------+-------------------------+------------
- - - - - - - - - - - - - -(Intercambio PAR-PAN) - - - - - - - -
Rx:PAR[]                RtxTimerStop();             WAIT_EAP_MSG
```

```
                        TxEAP();
                        EAP_RespTimerStart();
                        if (NO hay nonce fijado) {
                          NONCE_SENT=Set;
                          Tx:PAN[]("Nonce");
                        }
                        else
                          Tx:PAN[]();

Rx:PAR[] &&             RtxTimerStop();            WAIT_EAP_MSG
Temporizador            TxEAP();
                        EAP_RespTimerStart();

Rx:PAN[]                RtxTimerStop();            WAIT_PAA

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - -(Resultado PANA) - - - - - - - - - -
Rx:PAR[C] &&            TxEAP();  Se transmite     WAIT_EAP_RESULT
PAR.RESULT_CODE==        el contenido de EAP
   PANA_SUCCESS
(Hay éxito a nivel
 de PANA)
Rx:PAR[C] &&            if (PAR.exist_avp          WAIT_EAP_RESULT_
PAR.RESULT_CODE!=         ("EAP-Payload"))         CLOSE
  PANA_SUCCESS            TxEAP();
  (no hay éxito a       else
  nivel de PANA )        alt_reject();
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


-------------------
Estado: WAIT_EAP_MSG
-------------------

Condición de cambio          Acción                    Estado siguiente
----------------------+-------------------------+------------
- - - - - - - - - - ( PAN/PAR para  EAP) - - - - - - - - - - - -
EAP_RESPONSE &&         EAP_RespTimerStop()           WAIT_PAA
eap_piggyback()         if (No hay nonce) {
                          Tx:PAN[]("EAP-Payload",
                                      "Nonce");
                            NONCE_SENT=Set;
                        }
                        else
                           Tx:PAN[]("EAP-Payload");

EAP_RESPONSE &&         EAP_RespTimerStop()       WAIT_PAA
temporizador            Tx:PAR[]("EAP-Payload");
                        RtxTimerStart();
```

```
EAP_RESP_TIMEOUT         Tx:PAN[]();                WAIT_PAA

EAP_FAILURE              SessionTimerStop();        CLOSED
                         Disconnect();
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


----------------------
Estado: WAIT_EAP_RESULT
----------------------

Condición de cambio           Acción                    Estado siguiente
----------------------+-------------------------+------------
- - - - - - - - - - - - - (EAP Result) - - - - - - - - - - - - -
EAP_SUCCESS              if (PAR.exist_avp           OPEN
                            ("Key-Id"))
                           Tx:PAN[C]("Key-Id");
                         else
                           Tx:PAN[C]();
                         Authorize();
                         SessionTimerReStart
                         (LIFETIME_SESS_TIMEOUT);

EAP_FAILURE              Tx:PAN[C]();               CLOSED
                         SessionTimerStop();
                         Disconnect();
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

----------------------------
Estado: WAIT_EAP_RESULT_CLOSE
----------------------------

Condición de cambio           Acción                    Estado siguiente
----------------------+-------------------------+------------
- - - - - - - - - - - - - (EAP Result) - - - - - - - - - - - - -
EAP_SUCCESS ||           if (EAP_SUCCESS &&          CLOSED
EAP_FAILURE                PAR.exist_avp("Key-Id"))
                              Tx:PAN[C]("Key-Id");
                         else
                            Tx:PAN[C]();
                          SessionTimerStop();
                            Disconnect();
   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


-----------
Estado: OPEN
-----------
```

```
Condición de cambio           Acción                    Estado siguiente
----------------------+-------------------------+------------
- - - - - - - - - - (test de vida iniciado por el PaC)- - - - - -
PANA_PING                Tx:PNR[P]();               WAIT_PNA
                         RtxTimerStart();
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - (Re-autenticación iniciada  por el PaC)- - - - - -
REAUTH                   NONCE_SENT=Unset;          WAIT_PNA
                         Tx:PNR[A]();
                         RtxTimerStart();
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - (Reautenticación iniciada por el  PAA)- - - - - -
Rx:PAR[]                 EAP_RespTimerStart();      WAIT_EAP_MSG
                         TxEAP();
                         if (no vence el
                              temporizador)
                           Tx:PAN[]("Nonce");
                         else
                           NONCE_SENT=Unset;
                         SessionTimerReStart
                           (FAILED_SESS_TIMEOUT);
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - -(Terminación de la sesión iniciada por el PAA) - - - - - -
Rx:PTR[]                 Tx:PTA[]();                CLOSED
                         SessionTimerStop();
                         Disconnect();
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - -(Terminación de la sesión iniciada por el PaC) - - - - - -
TERMINATE                Tx:PTR[]();                SESS_TERM
                         RtxTimerStart();
                         SessionTimerStop();
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -


---------------
Estado: WAIT_PNA
---------------
Condición de cambio           Acción                    Estado siguiente
----------------------+-------------------------+------------
- - - - - - - - -(re-authentication initiated by PaC) - - - - -
Rx:PNA[A]                Rtx TimerStop();           WAIT_PAA
                         SessionTimerReStart
                         (FAILED_SESS_TIMEOUT);
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - -(liveness test initiated by PaC) - - - - - - -
Rx:PNA[P]                RtxTimerStop();            OPEN
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
---------------
State: SESS_TERM
---------------
Exit Condition          Exit Action               Exit State
----------------------+-------------------------+------------
- - - - - - - -(Session termination initiated by PaC) - - - - -
Rx:PTA[]               Disconnect();             CLOSED
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Para interpretar el diagrama basta con colocarse en uno de los estados y ver que comportamientos hacen cambiar de estado, fijándose en hacia qué nuevo estado se dirige el cliente tras recibir los mensajes.
Se intentó realizar un diagrama visual de la máquina, pero resultó imposible debido a la complejidad de la misma.

4.3. Código del cliente PANA: El método main

El cliente PANA debe implementar dos máquinas de estados: la de PANA, vista en el apartado anterior, y la de EAP[8], por suerte, ambas máquinas de estado están codificadas en OpenDiameter.

Explicar aquí el código del cliente PANA sería inútil y poco ilustrativo, sin embargo, sí consideramos importante comentar el código del método main del mismo, para de esta forma poder explicar el funcionamiento general del cliente.

4.3.1. Código main

```
int main(int argc, char **argv)
{
  std::string cfgfile;
  std::string userdb;

  ACE_Get_Opt opt(argc, argv, "cf:u:U:P:", 1);

  for (int c; (c = opt()) != (-1); ) {
      switch (c) {
          case 'f': cfgfile.assign(opt.optarg); break;
          case 'U': gUserName.assign(opt.optarg); break;
          case 'P': gPasswd.assign(opt.optarg); break;
      }
```

```
  }

  if ((opt.argc() < 1) ||
      (cfgfile.length() == 0)) {
    std::cout << "Usage: clientepana [-c] -f [configuration file]" << std::endl;
    return (0);
  }
```

En esta primera parte simplemente lee los los parámetros de la linea de comandos, como mínimo necesitamos un parámetro: el fichero de configuración del cliente, en este caso es: /config/pac.xml de donde obtendremos el nombre de usuario por defecto, la clave por defecto, la dirección del PAA, etc.

```
#ifdef WIN32
  EapLogMsg_S::instance()->open("EAP", ACE_Log_Msg::STDERR);
#else
  EapLogMsg_S::instance()->open("EAP", ACE_Log_Msg::STDERR);
#endif
  EapLogMsg_S::instance()->enable_debug_messages();

        PANA_CfgManager manager;
        manager.open(cfgfile);
        PANA_CFG_OPEN(cfgfile);
```

Inicializamos el log que escribirá en la ventana de comandos.

```
        PanaTask task;
        task.Start(5);
```

Creamos el hilo que llevará al cliente

```
 EapMethodStateMachineCreator<MyEapAuthIdentityStateMachine>
    myAuthIdentityCreator;

  EapMethodStateMachineCreator<MyEapPeerMD5ChallengeStateMachine>
    myPeerMD5ChallengeCreator;

  EapMethodRegistrar methodRegistrar;

  methodRegistrar.registerMethod
    (std::string("Identity"), EapType(1),
    Authenticator, myAuthIdentityCreator);

  methodRegistrar.registerMethod
    (std::string("MD5-Challenge"), EapType(4),
     Peer, myPeerMD5ChallengeCreator);
```

Creamos los métodos EAP que usaremos, en primer lugar enviaremos un mensaje de tipo 1, Identity y después el tipo 4, o Reto MD5, de esta forma nuestro cliente realizara los métodos EAP en el orden indicado.

```
        ACE_Semaphore semaphore(0);
        PANA_Node node(task, cfgfile);
        PeerApplication Peer(node, semaphore);
        Peer.pac().Start();
        semaphore.acquire();
        task.Stop();
    }
```

Por último utilizamos un semáforo para que el cliente no acabe su ejecución antes de tiempo y por causas ajenas a la comunicación (el semáforo estará siempre bloqueado). Creamos el nodo PANA, que internamente creará un cliente PANA y un canal PANA ligado a nuestro servidor NAS. A su vez, también creará la máquina de estados de PANA, que junto con la máquina de estados de EAP creada al registrar los métodos se encargará de gestionar los mensajes que envía y recibe del servidor.

4.4. Resultados obtenidos

Con este cliente sí obtenemos la autenticación pedida: El resultado obtenido por ventana de comandos es el siguiente:
AAAD:

```
[] Authorization totally success.
[] Sending DEA with a success Result-Code.
```

NASD:

```
AAA_SUCCESS received.
Passthrough: EAP authentication succeeded.
```

CLIENTE:

```
Peer: Success received.
Peer:  Success.
Authentication success detected at Peer
Welcome to the world, testuser@localdomain1.net !!!
```

4.5. Análisis del funcionamiento del cliente. Trazas

En esta sección realizaremos un análisis de los mensajes que se han enviado entre las tres entidades para llegar al resultado expuesto en el apartado anterior.

4.5.1. Mensajes entre el PaC y el NAS

Realizamos un análisis de los mensajes intercambiados entre el PaC y el NAS (PANA):

Se trata de una comunicación iniciada por el cliente.

```
  Cliente                                                    NAS

TxPCI(id=0)  ------------------------------->     RxPCI (id = 0)
```

Mandamos el mensaje de inicio de la conexión, el identificador de sesión es 0 en este punto, ya que debe ser elegido por el servidor.

```
RxPSR(id, seq)    <---------------------------     TxPSR(id, seq)
```

El servidor envía al cliente su identificador de sesión[2] (desde este punto id) y el número de secuencia[3](desde este punto seq) que el cliente deberá incluir en todos los mensajes subsiguientes.

```
TxPSA(id , seq+1)    ------------------------->     RxPSA(id, seq+1)
```

El cliente envía su petición al servidor (quiere autenticarse con MD5)

```
RxPAR(id, seq+2)    <--------------------------     TxPAR(id, seq+2 )
```

El servidor pide la identidad al cliente.

```
TxPAN(id, seq+3)    ------------------------->     RxPAN(id,seq + 3)
```

El cliente construye su mensaje de identidad y se lo envía al servidor. A partir de este punto el NAS se comunicará y con el AAA usando Diameter-EAP, para conseguir un resultado para la autenticación de este cliente. El NAS enviará la identidad del PaC al servidor AAA y este le pedirá que procese un determinado nonce.

```
RxPAR(id, seq + 4)   <-----------------------     TxPAR(id, seq + 4)
```

El servidor reenvía la petición del AAA sobre el nonce, para que sea el cliente quién calcule dicho nonce. (El NAS no conoce las claves de los clientes)

[2]EL identificador de sesión PANA, como vimos en el capítulo 2 es elegido de manera aleatoria.

[3]El número de secuencia en PANA es también elegido de manera aleatoria y monótonamente creciente.

```
TxPAN(id, seq + 5)   ------------------------>     RxPAN(id, seq + 5)
```

El cliente una vez calculado el procesamiento del nonce se lo envía al NAS que a su vez se lo envía al AAA.

```
RxPBR(id, seq + 6)   <------------------------      TxPBR(id, seq + 6)
```

El NAS envía el resultado obtenido desde el AAA al cliente, siendo este resultado un éxito.

```
TxPRA(id, seq + 7)   ------------------------>      RxPRA(id, seq + 7)
```

Finalmente el cliente asiente este último resultado.

4.5.2. Mensajes entre el NAS y el servidor AAA

Realizando un análisis con el analizador de red wireshark nos encontramos con que los mensajes intercambiados (entre NAS y servidor AAA) son los siguientes:
Nota importante: No se encontró ningún parche para wireshark que soportara Diameter-EAP, por lo que todo el análisis referente a dicho protocolo (por ejemplo, el AVP EAP-Payload, está hecho de manera manual).

1. Primer mensaje: Identity (NAS —- AAA)
   Contiene los siguientes AVPs:
   - Session-Id(263)
   - Auth-Application-Id (258) indicando el número "5"(Diameter-Eap)
   - Origin-Host (264) localnas.localdomain2.net
   - Origin-Realm (296) localdomain2.net
   - Destination-Realm (284) localdomain1.net
   - Auth-Request-Type(274) con el valor "3"(Authorize and Authenticate)
   - EAP-Payload (462) (ver después)
   - User-Name (1) testuser
   - Authorization-Lifetime (291) 30
   - Session-Time Out (27) 30

2. Segundo mensaje: Petición MD5 (AAA —- NAS)

   Contiene los siguientes AVPs:

   - Session-Id(263)
   - Auth-Application-Id (258) indicando el número "5"(Diameter-Eap)
   - Origin-Host (264) localaaa.localdomain1.net
   - Origin-Realm (296) localdomain1.net
   - Destination-Real (284) localdomain2.net
   - Auth-Request-Type(274) con el valor "3"(Authorize and Authenticate)
   - EAP-Payload (462) (ver después)
   - User-Name (1) testuser
   - Authorization-Lifetime (291) 30
   - Auth-Grace-Period (276) 30
   - Auth-Session-State (277) NO_STATE_MAINTEINED (1)
   - Session-Time Out (27) 30

3. Tercer mensaje: Respuesta MD5 (NAS —- AAA)

   Contiene los siguientes AVPs:

   - Session-Id(263)
   - Auth-Application-Id (258) indicando el número "5"(Diameter-Eap)
   - Origin-Host (264) localnas.localdomain2.net
   - Origin-Realm (296) localdomain2.net
   - Destination-Real (284) localdomain1.net
   - Auth-Request-Type(274) con el valor "3"(Authorize and Authenticate)
   - EAP-Payload (462) (ver después)
   - User-Name (1) testuser

4. Cuarto mensaje: Resultado (AAA —- NAS)

   Contiene los siguientes AVPs:

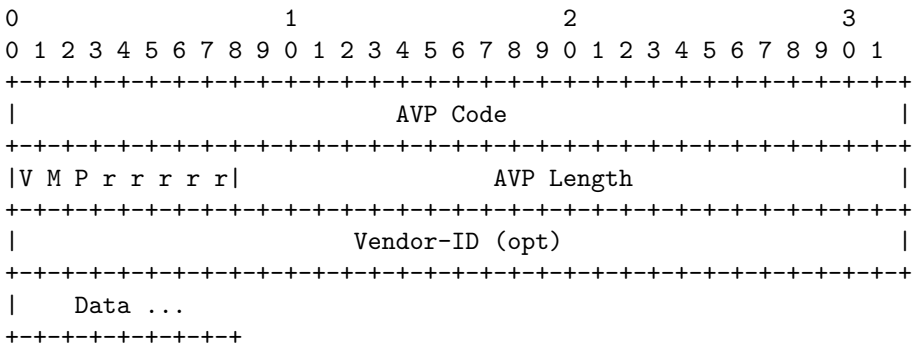
- Session-Id(263)
- Auth-Application-Id (258) indicando el número "5"(Diameter-Eap)
- Origin-Host (264) localaaa.localdomain1.net
- Origin-Realm (296) localdomain1.net
- Result-Code (268) llevando un 2001 Diameter Success
- Destination-Real (284) localdomain2.net
- Auth-Request-Type(274) con el valor "3"(Authorize and Authenticate)
- EAP-Payload (462) (ver después)
- User-Name (1) testuser

El AVP EAP-Payload (462) es el más importante en estos mensajes, ya que es el que transporta la información de EAP que viaja entre los enlaces y que es la que se encarga Realmente de llevar a cabo la autenticación.

En el primer mensaje, este AVP contiende la identidad del Peer, en el segundo contiene el nonce que el Peer debe encriptar con MD5, en el tercero contiene dicho nonce encriptado, para que el servidor pueda compararlo con el que el espera recibir y en el cuarto contiene el resultado de la autenticación: Éxito o fracaso.

La estructura de este mensaje no se especifica en el RFC de Diameter-EAP, sin embargo, observando las trazas de wireshark, podemos inducir que la estructura del mensaje es la siguiente:

Primero tenemos la estructura de cualquier AVP de EAP:



Y en la parte etiquetada como data tenemos un EAP-Payload:

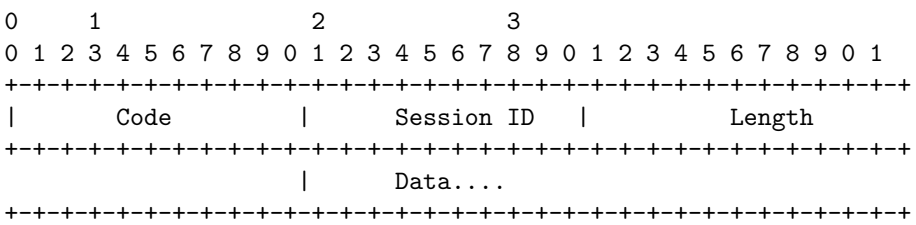
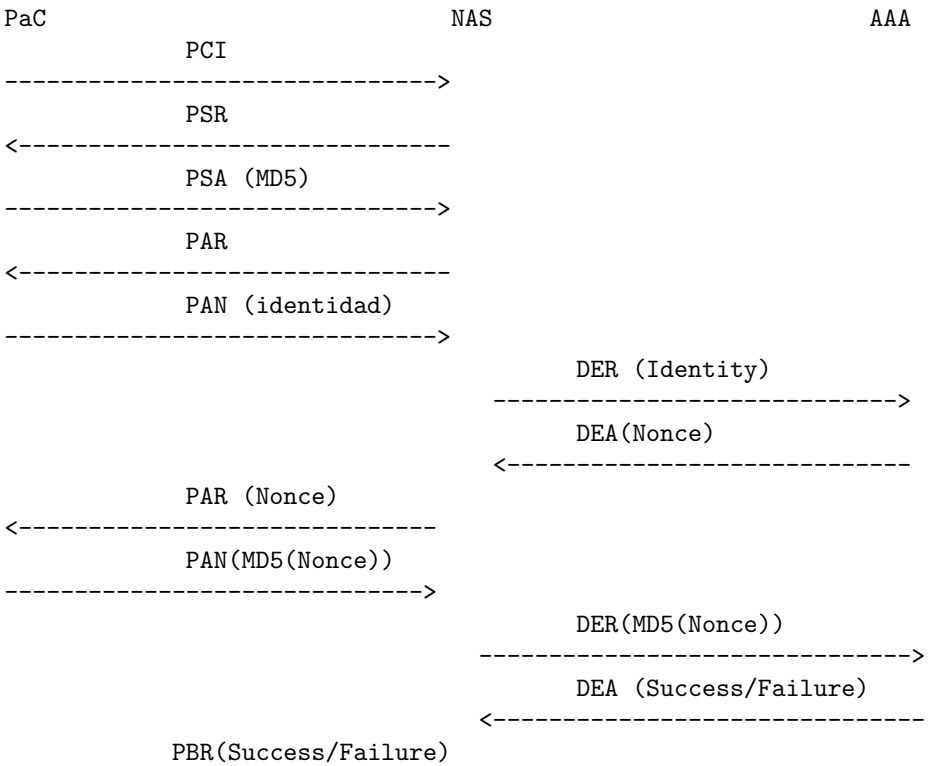


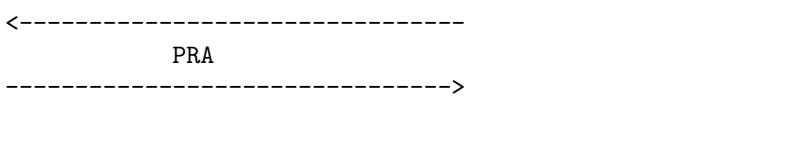
Figura 4.2: Estructura del AVP EAP-Payload (462)

Tenemos un bit de código de mensaje, pudiendo ser 1 (Request), 2 (Response),3 (Éxito) o 4 (Fracaso). Un bit de identificador de sesión. 2 bits de longitud del campo Data del AVP, es decir, incluyendo todos los campos aquí descritos. Y, por último, un campo de datos, que llevará la información propiamente dicha (nombre de usuario, nonce, etc) dependiendo del caso.

4.6. Esquema de la comunicación completa

El esquema de la comunicación completa, sin omitir ningún mensaje, e integrando en el diagrama a los 3 elementos que participan, es el siguiente:





4.7. Conclusiones

En esta sección, hemos desarrollado un cliente PANA, capaz de transportar EAP y de autenticarse con un servidor de AAA, a través de un NAS (Network Access Server) o servidor de pasarela.

Este cliente se ha autenticado frente a un servidor de AAA en local, sin embargo, se espera que el código sea válido también para el servidor alojado en el core de IMS.

Hay que tener en cuenta, que la comunicación desarrollada se limita a la fase de autenticación y autorización de PANA.
Una vez terminada esta fase implementada, comenzaría la fase de acceso, en la que el cliente podría enviar tráfico IP a la red.

Desde este punto, el objetivo del proyecto es reproducir esta autenticación con el servidor integrado en el core de IMS.

Capítulo 5

Comunicación con el servidor de IMS

Terminada la fase de autenticación en local, el objetivo ahora es conseguir el mismo resultado contra un servidor remoto de IMS. La estructura de la red IMS puede consultarse en la bibliografía del presente documento[7].

Nuestro objetivo es autenticarnos con el servidor HSS, que forma parte del core de IMS y que utiliza Diameter para comunicarse. La comunicación con dicho servidor se efectuará directamente a través del NAS.
El escenario propuesto es el indicado en la figura [5.1]

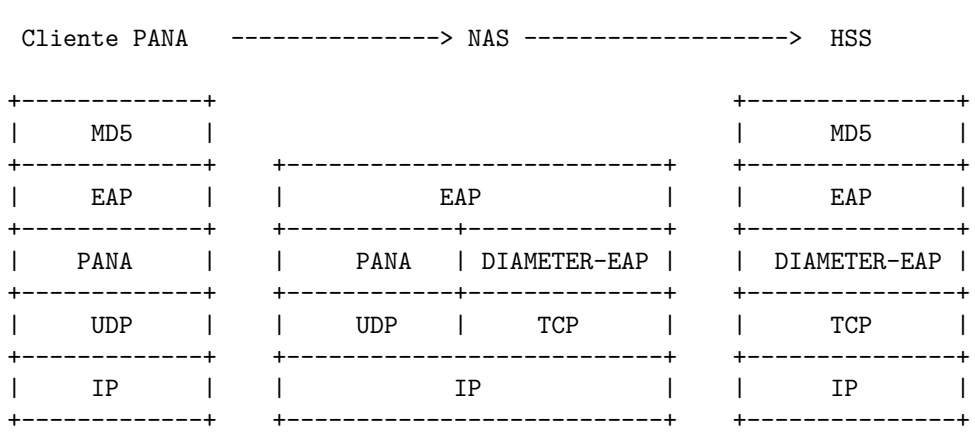


Figura 5.1: Escenario MD5 remoto

Para conseguir la comunicación, en principio, bastaría con cambiar en el fichero de configuración del NAS (nasd_diameter_eap.xml) los siguientes parámetros (estos valores de configuración se incluyen aquí y no en un anexo

debido a que son cortos, y pensamos que incluirlos en una anexo sería perjudicial):

- El Peer. Actualmente el servidor local AAA es el Peer, habría que cambiarlo por el servidor HSS:

```
<Peer>
   - <hostname>locaaa.localdomain1.net</hostname>
   - <port>1812</port>
   + <hostname>hss.carissimi.gast.it.uc3m.es</hostname>
   + <port>3868</port>
     <use_sctp>0</use_sctp>
     <tls_enabled>0</tls_enabled>
</Peer>
```

- La tabla de rutas: Añadimos la siguiente ruta:

```
<route>
    <Realm>carissimi.gast.it.uc3m.es</Realm>
    <role>1</role>
    <redirect_usage>0</redirect_usage>
    <application>
       <application_id>5</application_id>
       <vendor_id>0</vendor_id>
       <Peer_entry>
           <server>hss.carissimi.gast.it.uc3m.es</server>
           <metric>1</metric>
       </Peer_entry>
    </application>
 <\route>
```

Por último, debido a que el HSS funciona con Vendor-Specific-Ids, necesitamos declarar una que el HSS tenga registrada. Añadimos:

```
<vendor_specific_application_id>
   <vendor_id>10415</vendor_id>
   <auth_application_id>16777216</auth_application_id>
</vendor_specific_application_id>
```

En la parte de declaraciones.
El motivo de hacer esto es, aunque como veremos a continuación los interfaces declarados en el HSS no nos sirven para usar Diameter-EAP, sí nos servirán para usar Diameter Base Protocol, ahorrándonos redefinir y duplicar código.

Con estos cambios realizados lanzamos el nasd y el cliente PANA, el resultado obtenido es:

- El intercambio de capacidades y el watchdog son correctos (se analizarán más adelante).

- Una vez el cliente PANA entra en funcionamiento, las peticiones del NAS al HSS acerca de dicho cliente nunca son contestadas.

Es decir, el HSS contesta a los mensajes de Diameter Base Protocol, pero no a los mensajes de Diameter-EAP. Ésto hace pensar que el HSS no tiene implementada la aplicación Diameter-EAP y que, por lo tanto, estamos intentando construir el siguiente escenario, que nunca podrá funcionar (figura 5.2):

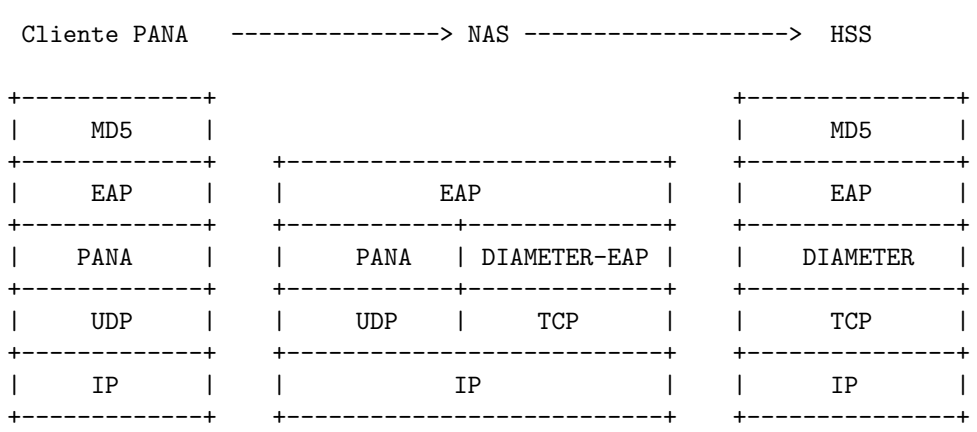


Figura 5.2: Escenario MD5 remoto sin modificaciones en el HSS

5.1. Interfaces del HSS

Consultando la bibliografía del HSS[7] comprobamos que las interfaces soportadas son las siguientes:

- Cx = 16777216
- Sh = 16777217
- Zh = 16777221

Como puede verse, el interfaz 5 de Diameter-EAP no está soportado. Por otro lado, los comandos soportados por el Core de IMS no incluyen los comandos DER y DEA (268).

Como consecuencia de todo esto, habrá que implementar el interfaz de Diameter-EAP y los comandos DER y DEA desde 0.

5.2. Implementación del Interfaz Diameter-EAP. Comandos DER y DEA

El código del core de IMS se encontraba alojado en una de las máquinas del laboratorio. El objetivo de este apartado es explicar como se implementó el interfaz y los comandos de Diameter-EAP sobre el servidor HSS. Este módulo IMS no usa las bibliotecas OpenDiameter. Además, no está implementado en c++, sino en Java.

5.2.1. Desarrollo del interfaz Diameter-EAP

El código implementado para este interfaz se encuentra en el tercer anexo de este documento, por lo que no será comentado aquí.

Se trata de un prototipo básico de servidor implementado para comprobar la funcionalidad del cliente. La máquina de estados de dicho servidor no está completa y sólo recoge algunos supuestos:

```
Estado inicial              Acción              Estado siguiente
-------------------------------------------------------------------

  INICIO                Se recibe un mensaje
                            Identity,
                        si es correcto ---------> ENVÍO DE NONCE
                        si no es correcto-------> INICIO
                        *( En este caso el mensaje
                        es correcto o no si la
                        indentidad está o no
                        registrada)

  ENVÍO DE NONCE        Se recibe un nonce
                        procesado con MD5
                        si es correcto  --------> ÉXITO
                        si no ------------------> FRACASO

  ÉXITO

  FRACASO
```

Como puede apreciarse, el diagrama de estados es muy simple, y no se han tenido en cuenta eventos de temporización.

5.2.2. Conclusiones al desarrollo del servidor

1. Tuvimos que dar un salto de c++ a java para implementar esta interfaz, con las consecuentes dificultades que esto conlleva.

2. Puede observarse que nuestra interfaz Diameter-Eap es bastante limitada, esto se debe a que la implementación se ciñó estrictamente a lo que íbamos a utilizar, puesto que el objetivo del PFC no era el desarrollo de dicha interfaz para el core de IMS.

3. Aunque procuramos programar la interfaz de la manera más cuidadosamente posible, no podemos asegurar que la implementación esté libre de errores. Ésto complica el proceso de pruebas del cliente, ya que si algo falla tendremos primero que localizar qué está fallando. Este hecho unido a los múltiples problemas que presentan las bibliotecas de OpenDiameter hacen que el determinar el elemento que no funciona bien en una comunicación sea realmente difícil.

4. Puede observarse (en el código), que el servidor no busca en ninguna base de datos o fichero el método de autenticación que usará con el cliente, sino que utiliza directamente MD5. De nuevo, no consideramos necesario corregir este fallo, ya que sólo usaremos esta interfaz para MD5, pero si en un futuro se planteara usarla de forma más extensa sería una de las primeras cosas a corregir.

5.3. Comunicación con el HSS, análisis de trazas

En esta sección realizaremos un análisis, como el que ya se realizó para la comunicación en local, de los mensajes intercambiados entre las 3 entidades. Para ello usaremos la herramienta wireshark y las trazas que el NAS y el cliente PANA escriben por pantalla.
El servidor HSS no escribirá ningún mensaje por pantalla. Ésto es una decisión de implementación de sus creadores, que nosotros respetamos al crear el interfaz para Diameter-EAP.

5.3.1. Watchdog e intercambio de capacidades

Intercambio de capacidades

Al lanzar el NAS con las configuraciones correctas para que se comunique con el HSS, puede observarse por pantalla el siguiente mensaje:

```
(16293|3046546320) Peer Capabilities
(16293|3046546320)             Hostname : hss.carissimi.gast.it.uc3m.es
(16293|3046546320)                Realm : carissimi.gast.it.uc3m.es
(16293|3046546320)              Host IP : type=1, 163.117.141.114
(16293|3046546320)             VendorId : 10415
(16293|3046546320)         Product Name : JavaDiameterPeer
(16293|3046546320)           Orig State : 0
(16293|3046546320)  Vendor Specific Id : (16293|3046546320)
                                         Vendor=10415,  Auth=16777216
(16293|3046546320)  Vendor Specific Id : (16293|3046546320)
                                         Vendor=4491,  Auth=16777216
(16293|3046546320)  Vendor Specific Id : (16293|3046546320)
                                         Vendor=13019,  Auth=16777216
(16293|3046546320)  Vendor Specific Id : (16293|3046546320)
                                         Vendor=10415,  Auth=16777217
(16293|3046546320)  Vendor Specific Id : (16293|3046546320)
                                         Vendor=10415,  Auth=16777221
(16293|3046546320)           Inband Sec : 0
(16293|3046546320)          Firmware Ver : 0
(16293|3046546320) *** Local capabilities accepted by Peer ***
```

Usando wireshark para analizar las trazas nos encontramos lo siguiente:

- Flags = 10000000 (Sólo el flag Request a 1)
- Command Code = 257
- ApplicationId = 0 (Diameter Common Messages)
- AVPs:
  - Origin-Host (264) : localaaa.localdomain1.net
  - Origin-Realm (296): localdomain1.net
  - Host-IP-Address (257) : 127.0.0.1
  - Vendor-Id (266) : 0
  - Product-Name (269) : Open Diameter
  - Supported-Vendor-Id (265) : 0
  - Supported-Vendor-Id (265) : 1
  - Origin-State-Id : 1240325465
  - Auth-Application-Id (258) : 1 (NASREQ)
  - Auth-Application-Id (258) : 2 (Mobile IP v4)
  - Auth-Application-Id (258) : 5 (Diameter EAP)
  - Acct-Application-Id (259) : 3 (Base Accounting)

- Firm-Ware-Revision (267) : 1
- Inband-Security-Id (299) : 0 (NO_INBAND_SECURITY)

Si comparamos con el RFC de Diameter:

```
<CER> ::= < Diameter Header: 257, REQ >
                { Origin-Host }
                { Origin-Realm }
              1* { Host-IP-Address }
                { Vendor-Id }
                { Product-Name }
                [ Origin-State-Id ]
               * [ Supported-Vendor-Id ]
               * [ Auth-Application-Id ]
               * [ Inband-Security-Id ]
               * [ Acct-Application-Id ]
               * [ Vendor-Specific-Application-Id ]
                 [ Firmware-Revision ]
               * [ AVP ]
```

Comprobamos que el mensaje está correctamente construido, tanto en flags, como en AVPs. Para el caso de la respuesta tenemos algo muy similar:

- Flags = 00000000 (El flag R.ª 0 puesto que es una respuesta)
- Command Code = 257
- ApplicationId = 0 (Diameter Common Messages)
- AVPs:
  - Result-Code (268) : DIAMETER_SUCCESS (2001)
  - Origin-Host (264) : hss.carissimi.gast.it.uc3m.es
  - Origin-Realm (296): carissimi.gast.it.uc3m.es
  - Host-IP-Address (257) : 163.117.141.114
  - Vendor-Id (266) : 10415
  - Product-Name (269) : JavaDiameter
  - AVP-Vendor-Speccific-Id (260) ; Valor no legible
  - AVP-Vendor-Speccific-Id (260) ; Valor no legible
  - AVP-Vendor-Speccific-Id (260) ; Valor no legible
  - AVP-Vendor-Speccific-Id (260) ; Valor no legible

Comprobamos con el RFC:

```
<CEA> ::= < Diameter Header: 257 >
                { Result-Code }
                { Origin-Host }
                { Origin-Realm }
             1* { Host-IP-Address }
                { Vendor-Id }
                { Product-Name }
                [ Origin-State-Id ]
                [ Error-Message ]
              * [ Failed-AVP ]
              * [ Supported-Vendor-Id ]
              * [ Auth-Application-Id ]
              * [ Inband-Security-Id ]
              * [ Acct-Application-Id ]
              * [ Vendor-Specific-Application-Id ]
                [ Firmware-Revision ]
              * [ AVP ]
```

Es importante destacar que los mensajes de respuesta del HSS son distintos a los que obteníamos en local, son coherentes con el RFC, pero podemos ver que en lugar de hacer un intercambio de aplicaciones de manera normal[1], intercambia las capacidades específicas para el vendor-ID.

Watchdog

Para el WacthDog:

- Flags = 10000000 (Sólo el R (Request) a 1)
- Command-Code = 280 (Device-Wachtdog)
- Application-Id = 0 (Diameter Common Messages)
- AVPs:
  - Origin-Host (264) : localnas.localdomain2.net
  - Origin-Realm (296): localdomain2.net
  - Origin-State-Id : 1240325465

[1]Esto no es del todo correcto: para los mensajes tipo Watchdog y de intercambio de capacidades estamos usando el interfaz Cx, en lugar del interfaz Diameter-EAP, que sería lo correcto.
Esta decisión fue tomada, porque para usar estos mensajes por el interfaz de Diameter-EAP habríamos tenido que reimplementarlos y cambiar aún más el código del HSS, lo que conllevaría mayor probabilidad de tener errores. Ya que los mensajes pertenecen al Diameter Base Protocol, consideramos que no era un error grave el hacerlo así.

Comparando con el RFC:

```
<DWR>  ::= < Diameter Header: 280, REQ >
           { Origin-Host }
           { Origin-Realm }
           [ Origin-State-Id ]
```

Y en cuanto a la respuesta:

- Flags = 10000000 (Sólo el R (Request) a 1)
- Command-Code = 280 (Device-Wachtdog)
- Application-Id = 0 (Diameter Common Messages)
- AVPs:
  - Result-Code (268): DIAMETER_SUCCESS (2001)
  - Origin-Host (264) : hss.carissimi.gast.it.uc3m.es
  - Origin-Realm (296): carissimi.gast.it.uc3m.es

RFC:

```
<DWA>  ::= < Diameter Header: 280 >
           { Result-Code }
           { Origin-Host }
           { Origin-Realm }
           [ Error-Message ]
         * [ Failed-AVP ]
           [ Original-State-Id ]
```

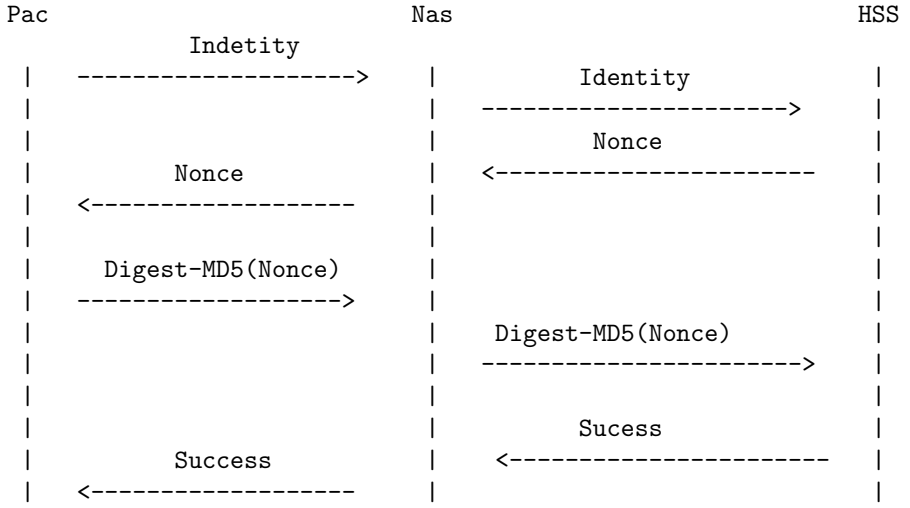
El mensaje es igualmente correcto según el RFC, pero al igual que sucedía en el caso del intercambio de capacidades es distinto al ofrecido por el AAA local.

5.3.2. Cambios en el cliente PANA

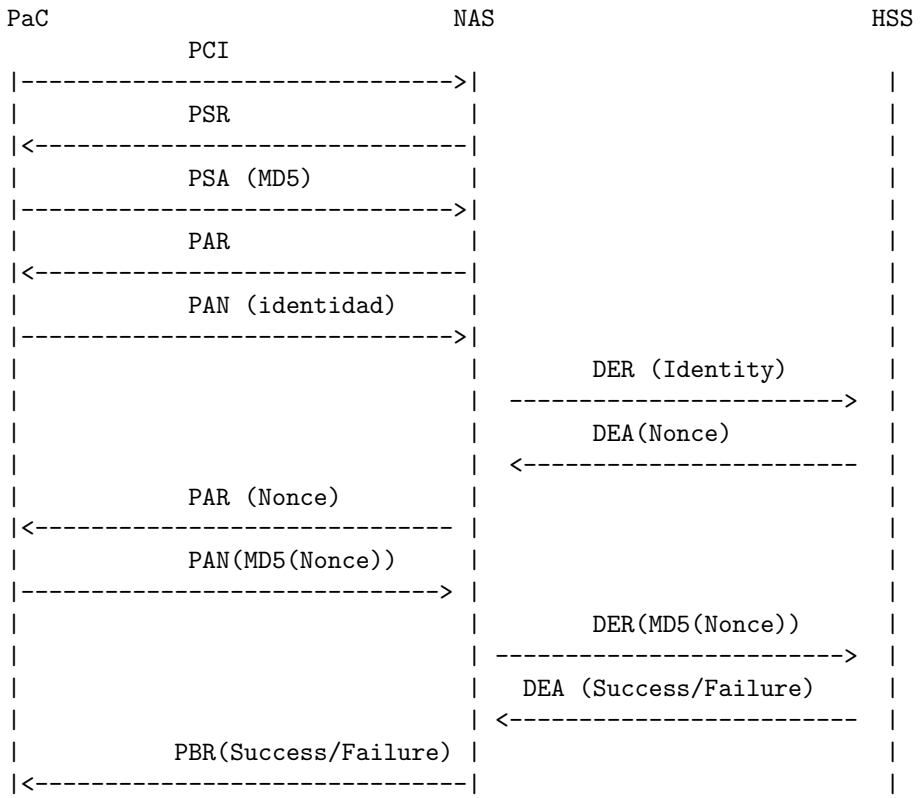
El cliente PANA no requiere ningún cambio, ni en código ni en configuración.
El cliente PANA se comunicará directamente con el NAS, y éste sigue siendo el mismo que en el escenario en local.
El único cambio aplicable al cliente PANA es que ahora los clientes deben pertenecer al dominio carissimi.gast.it.uc3m.es para que el NAS sea capaz de enrutarlos hacia el HSS.

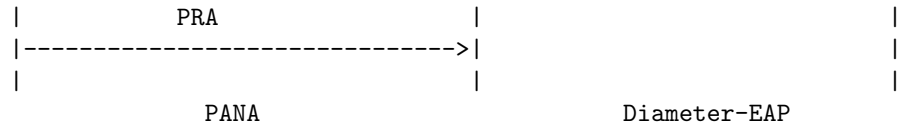
5.3.3. Autenticación con MD5. Intercambio de mensajes

El escenario que tenemos ahora es muy similar al escenario local, al igual que los mensajes que deseamos enviar y recibir:



o, de una manera más detallada, el objetivo es el siguiente:





Detalle de los mensajes intercambiados

Dado que para implementar el servidor de Diameter-EAP en el HSS nos basamos en los mensajes intercambiados por el NAS y el servidor de AAA en local, las trazas de los mensajes son idénticas en ambos supuestos. Para un estudio más detallado consultar a sección xxxxx.
Por completitud, se transcriben aquí las trazas de wireshark obtenidas.

1. Primer mensaje: Identity (NAS — HSS)

   Contiene los siguientes AVPs:

   - Session-Id(263)
   - Auth-Application-Id (258) indicando el número "5"(Diameter-Eap)
   - Origin-Host (264)
   - Origin-Realm (296)
   - Destination-Real (284)
   - Auth-Request-Type(274) con el valor "3"(Authorize and Authenticate)
   - EAP-Payload (462) (ver después)
   - User-Name (1)
   - Authorization-Lifetime (291)
   - Session-Time Out (27)

2. Segundo mensaje: Petición MD5 (HSS — NAS)

   Contiene los siguiente AVPs:

   - Session-Id(263)
   - Auth-Application-Id (258) indicando el número "5"(Diameter-Eap)
   - Origin-Host (264)
   - Origin-Realm (296)
   - Destination-Real (284)

- Auth-Request-Type(274) con el valor "3"(Authorize and Authenticate)
- EAP-Payload (462) (ver después)
- User-Name (1)
- Authorization-Lifetime (291)
- Auth-Grace-Period (276)
- Auth-Session-State (277)
- Session-Time Out (27)

3. Tercer mensaje: Respuesta MD5 (NAS — HSS) Contiene los siguiente AVPs:

   - Session-Id(263)
   - Auth-Application-Id (258) indicando el número "5"(Diameter-Eap)
   - Origin-Host (264)
   - Origin-Realm (296)
   - Destination-Real (284)
   - Auth-Request-Type(274) con el valor "3"(Authorize and Authenticate)
   - EAP-Payload (462) (ver después)
   - User-Name (1)

4. Cuarto mensaje: Resultado (HSS — NAS)

   Contiene los siguiente AVPs:

   - Session-Id(263)
   - Auth-Application-Id (258) indicando el número "5"(Diameter-Eap)
   - Origin-Host (264)
   - Origin-Realm (296)
   - Result-Code (268) llevando un 2001 Diameter Success
   - Destination-Real (284)
   - Auth-Request-Type(274) con el valor "3"(Authorize and Authenticate)

- EAP-Payload (462) (ver después)
- User-Name (1)

El EAP-Payload lleva exactamente el mismo contenido que el explicado para la autenticación en local[2], y por eso no se considera necesario incluirlo de nuevo aquí.

5.3.4. Resultados obtenidos

Los resultados son completamente análogos a los obtenidos con la autenticación en local:
NASD:

```
AAA_SUCCESS received.
Passthrough: EAP authentication succeeded.
```

CLIENTE:

```
Peer: Success received.
Peer:  Success.
Authentication success detected at Peer
Welcome to the world, bob@carissimi.gast.it.uc3m.es !!!
```

En otro capítulo de este mismo documento hay una serie de pruebas que se pasaron a este escenario remoto.

5.4. Resumen de la autenticación con MD5

Durante estos dos primeros capítulos se ha conseguido:

1. Crear un cliente PANA, capaz de comunicarse con el NAS local y de autenticarse con distintos servidores de AAA, tanto locales como remotos.

2. Comprender el funcionamiento de las bibliotecas OpenDiameter.

3. Comprender el funcionamiento de los ficheros de configuración xml.

4. Ampliar el servidor HSS alojado en la Universidad con el módulo de Diameter-EAP.

[2]Ésto se debe a que el HSS fue programando siguiendo como modelo un intercambio de trazas entre el NAS y el servidor AAA en local.

Parte III

Autenticación con TLS

Capítulo 6

TLS. Objetivos y panorámica

En esta tercera parte del PFC intentaremos y conseguiremos realizar una autenticación con TLS en local.
La autenticación sólo se llevará a cabo en local porque programar una interfaz de Diameter-EAP con TLS para el HSS es algo muy complejo y que llevaría mucho tiempo; pudiendo tratarse de un PFC entero por sí mismo.
Para llevar a cabo esta autenticación usaremos el NAS que hemos usado para MD5, ya que esta entidad actúa sólo como intérprete entre PANA y Diameter[1]. Usaremos un nuevo cliente PANA programado y como servidor AAA usaremos el demonio server_test_tls proporcionado por las bibliotecas OpenDiameter.
TLS requiere el uso de certificados que obtendremos de la universidad.

Por último, realizaremos un análisis de las trazas intercambiadas entre los 3 participantes en la autenticación tal como hicimos en MD5.

En la siguiente figura, resumimos gráficamente el escenario que pretende construirse.

6.1. OpenDiameter y TLS

OpenDiameter proporciona soporte para TLS. En uno de los anexos de este documento puede encontrarse la guía de instalación con TLS.
La instalación de TLS es bastante complicada y está plagada de errores, además algunos de los ficheros que deben usarse son del año 2004 (siendo el resto de la distribución del 2007).

[1]Consultar la figura 3.1 ó 5.1

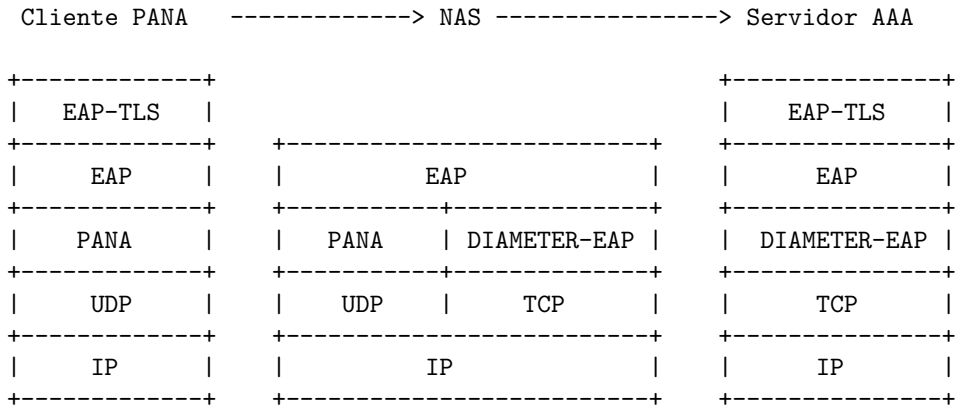
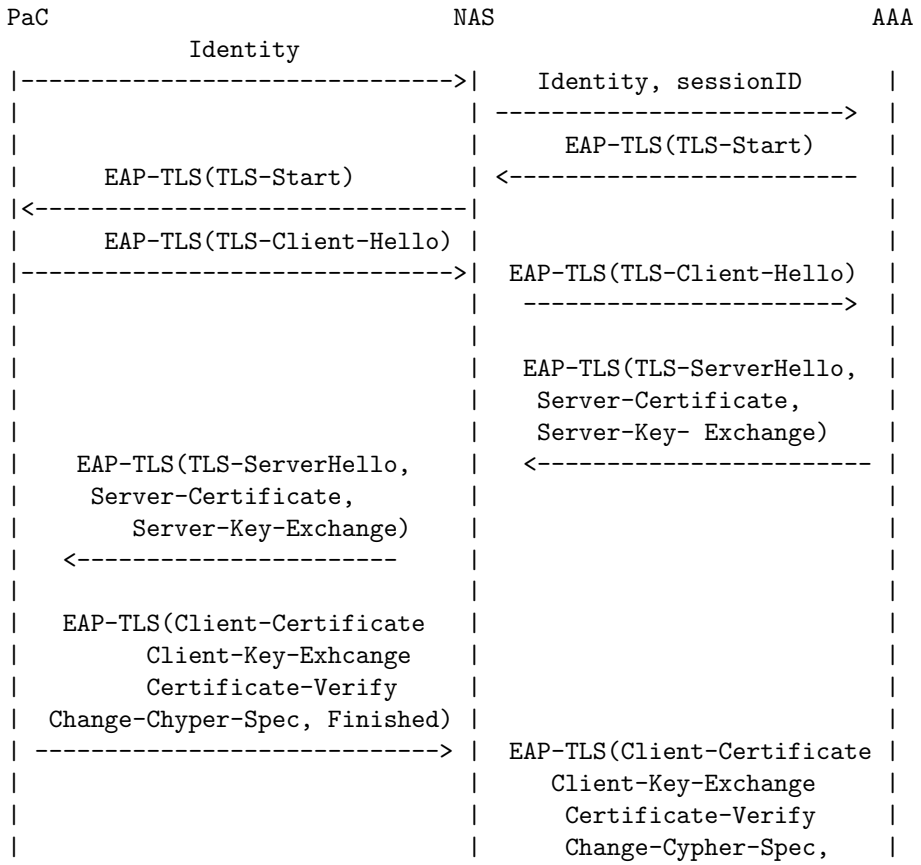
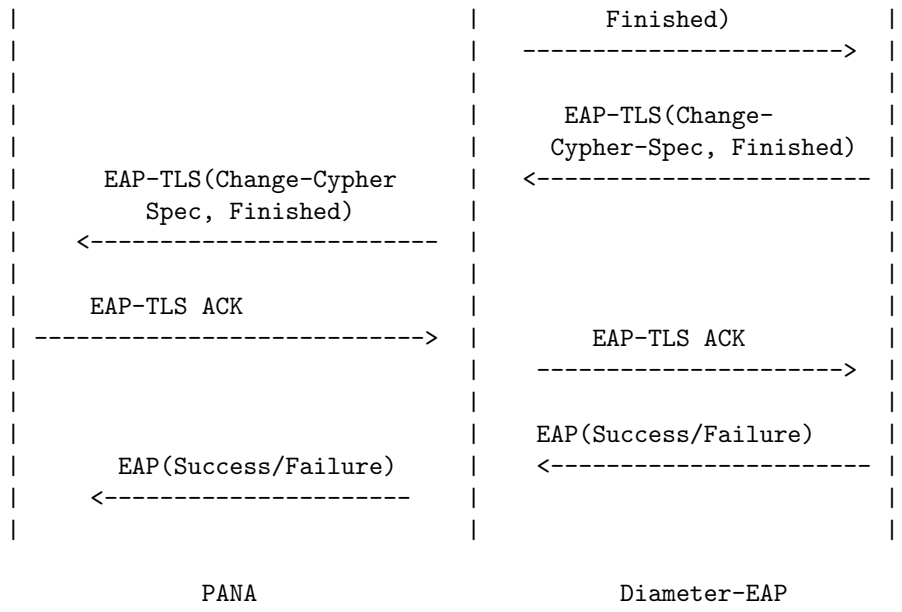


Figura 6.1: Escenario TLS

6.2. Esquema de la comunicación

En el siguiente esquema, tenemos un resumen gráfico de los mensajes que se intercambiarán las 3 entidades para llevar a cabo la autenticación:





Cabe remarcar, de nuevo, que este mecanismo produce autenticación mutua y un intercambio de claves, además de crear una conexión tunelada.
Los mensajes de la figura son bastante gráficos y autoexplicativos, pero podemos destacar varias cosas:

- Se está enviando TLS sobre EAP-TLS sobre EAP.
- Los mensajes se fragmentan.
- pese a que la comunicación se produce en EAP-TLS, el éxito o fracaso se produce a nivel EAP.

6.3. Cliente PANA con TLS. Creación y funcionamiento

Para programar el cliente PANA con TLS se aprovechó la estructura ya programada para el cliente PANA con MD5.

Hay que tener en cuenta que, ahora, el cliente tendrá que implementar hasta 4 máquinas de estados distintas (PANA, EAP, EAP-TLS y TLS). Por suerte, todas ellas están ya implementadas en OpenDiameter.

Fijándonos en las bibliotecas de OpenDiameter, para conseguir un cliente que soporte TLS, basta con implementar los métodos declarados como

abstractos, de forma que actúen de acuerdo a TLS, y registrar TLS como método para dicho cliente.

El código del cliente está resumido en los anexos, aquí sólo se indicarán las partes más importante del mismo:
El método main:

```
int main(int argc, char **argv)
{
  std::string cfgfile;
  std::string userdb;

  ACE_Get_Opt opt(argc, argv, "cf:u:U:P:A:", 1);

  for (int c; (c = opt()) != (-1); ) {
      switch (c) {
          case 'f': cfgfile.assign(opt.optarg); break;
          case 'U': gUserName.assign(opt.optarg); break;
          case 'P': gPasswd.assign(opt.optarg); break;
      }
  }

  if ((opt.argc() < 1) ||
      (cfgfile.length() == 0)) {
    std::cout << "Usage: clientepanatls [-c] -f [configuration file]" << std::endl;
    return (0);
  }
```

En esta primera parte simplemente tomamos los parámetros de la línea de comandos, pudiendo ser dichos parámetros el fichero de configuración (obligatorio), el nombre de usuario y la clave del usuario. Es obligatorio que al menos aparezca el parámetro referente al fichero de configuración.

```
#ifdef WIN32
  EapLogMsg_S::instance()->open("EAP", ACE_Log_Msg::STDERR);
#else
  EapLogMsg_S::instance()->open("EAP", ACE_Log_Msg::STDERR);
#endif
  EapLogMsg_S::instance()->enable_debug_messages();
```

Aquí se inicializan los distintos logs en los que escribirá el programa.

```
PanaTask task;
task.Start(5);

 EapMethodStateMachineCreator<MyEapAuthIdentityStateMachine>
    myAuthIdentityCreator;

  EapMethodStateMachineCreator<MyEapPeerTlsStateMachine>
```

```
    myPeerTlsCreator;

  EapMethodRegistrar methodRegistrar;
```

En esta parte declaramos y registramos los métodos que se usarán, creando la máquina de estados correspondiente a cada uno de ellos (Identity y TLS).

```
  methodRegistrar.registerMethod
    (std::string("Identity"), EapType(1),
     Authenticator, myAuthIdentityCreator);


  methodRegistrar.registerMethod
    (std::string("TLS"), EapType(TLS_METHOD_TYPE),
     Peer, myPeerTlsCreator);
```

registramos los métodos, comenzamos por -Identity- y seguimos por -TLS- que tiene asignado el número "13".

```
ACE_Semaphore semaphore(0);
PANA_Node node(task, cfgfile);
PeerApplication Peer(node, semaphore);
Peer.pac().Start();
semaphore.acquire();
task.Stop();
}
```

Iniciamos el cliente en sí mismo, con el semáforo evitamos que la ejecución termine por causas ajenas a la propia comunicación, ya que el semáforo permanecerá siempre bloqueado.

Como puede apreciarse el código del main es prácticamente igual al que usábamos en el caso de MD5, sólo cambian las declaraciones de los métodos y que en lugar de registrar MD5 se registra TLS.

Hay otra serie de diferencias en el código que marcan realmente las diferencias con el cliente de MD5, pueden consultarse comparando los ficheros -cliente.cxx- (MD5) y -cliente2.cxx- (TLS), pero que no se comentarán aquí por tener una extensión elevada y no ser necesario.

Se incluye el código del método main a modo de representación de la modularidad del cliente (con pocos cambios podemos implementar distintos métodos de autenticación).

6.4. NAS y servidor AAA para TLS

Para el NAS y el servidor AAA usaremos ficheros incluidos en la distribución de OpenDiameter.

El NAS es el mismo que se usó en MD5; en cuanto al servidor AAA, se usará el fichero server_test_tls.cxx, que implementa un servidor AAA con capacidad de entender EAP-TLS.

Las comunicaciones entre el NAS y el AAA sin cliente son idénticas a las especificadas para el caso de MD5, es decir, los mensajes de intercambio de capacidades y de Watchdog son los mismos[2]. Por lo tanto, y dado que en la etapa anterior se hizo un análisis detallado de los ellos, omitiremos el comentarlos en este apartado de la memoria.

6.4.1. Complicaciones en el funcionamiento del servidor de AAA.

Se recomienda ver los anexos para poder hacer funcionar el servidor AAA con TLS. No es objeto de esta sección del proyecto explicarlo ya que se tratan de errores de compilación y de ficheros de configuración. Se indica aquí este apunte debido a que es realmente complicado hacerlo funcionar, y no se trata, en ningún aspecto, de una instalación normal.

6.5. Resultados obtenidos

Finalmente, tras las complicaciones para hacer funcionar el servidor AAA con TLS, los resultados obtenidos fueron de éxito. A continuación, se justificará esta afirmación con trazas por pantalla y trazas de la herramienta de análisis de redes wireshark.

6.5.1. Trazas por pantalla

Las trazas obtenidas por pantalla son muy extensas, y por lo tanto no se incluirán enteras en este documento.
Consideramos que el análisis de wireshark es suficientemente completo como para no incluir la totalidad de las trazas. Sólo se incluirán fragmentos de las mismas, a modo de referencia.

Trazas en el lado del servidor AAA

```
rlm_eap_tls: <<< TLS 1.0 Handshake [length 0052], ClientHello ...
rlm_eap_tls: >>> TLS 1.0 Handshake [length 004a], ServerHello ...
```

[2]Excepto porque se declara TLS como método de autenticación mutua.

```
rlm_eap_tls: >>> TLS 1.0 Handshake [length 04e9], Certificate ...
rlm_eap_tls: >>> TLS 1.0 Handshake [length 0059], CertificateRequest ...
rlm_eap_tls: <<< TLS 1.0 Handshake [length 04e7], Certificate ...
rlm_eap_tls: <<< TLS 1.0 Handshake [length 0086], ClientKeyExchange ...
rlm_eap_tls: <<< TLS 1.0 Handshake [length 0086], CertificateVerify ...
rlm_eap_tls: <<< TLS 1.0 ChangeCipherSpec [length 0001] ...
rlm_eap_tls: <<< TLS 1.0 Handshake [length 0010], Finished ...
rlm_eap_tls: >>> TLS 1.0 ChangeCipherSpec [length 0001] ...
rlm_eap_tls: >>> TLS 1.0 Handshake [length 0010], Finished ...
rlm_eap_tls: Readed bytes 59
EapRequestTlsParser::parseAppToRaw LENGTH PACKET 59
 Backend: Request sent and timer started.
EAP Request sent from backend authenticator
[] EAP Request received from backend.
[] Sending DEA with continue result-code.
[] forwarding EAP Response to authenticator.
EAP Response forwarded to backend authenticator
Backend: Integrity Check.
------------>AuthTls: Process response second way message.
EapRequestTlsParser::parseRawToApp LENGTH PACKET 6
AuthTls: AcNotifySuccess
rlm_eap_tls: >>> TLS 1.0 Alert [length 0002], warning ...  close_notify
TLS Alert write:warning:close notify
Backend: Composing Success message.
EAP Success sent from backend authenticator
[] EAP Success received from backend.
Auth TLS session key
LENGTH cad 64
5f d6 e9 dc e4 77 b3 17 79 5e c5 8f 4b 78 0f d7 84 3f 71 2e ae
6a 40 b2 af 44 91 ae 3b 7e 0b b1 21 24 4e 16 43 05 58 ae 42 9f b8
 bc cf 3f ea 1e 3d 47 d5 c9 45 3b fc 3e 02 7c 8f 5c 8e 32 21 31
[] Authorizing DER.
Setting master session key
[] Authorization totally success.
[] Sending DEA with a success Result-Code.
```

Trazas en el lado del NAS

```
EAP-Response received from passthrough.
sending DER.
AAA_SUCCESS received.
```

Trazas en el lado del cliente PANA

```
PeerTls: Processing EAP-TLS Start.<-----------------
PeerTls: Building a Client Hello Message.--------------->
-------------->PeerTls: Verify Auth More Fragments.
<----------- PeerTls: Send ACK.
```

```
rlm_eap_tls: <<< TLS 1.0 ChangeCipherSpec [length 0001] ...
rlm_eap_tls: <<< TLS 1.0 Handshake [length 0010], Finished ...
PeerTls: Send Ack.
EapRequestTlsParser::parseAppToRaw LENGTH PACKET 0
 rlm_eap_tls: >>> TLS 1.0 Alert [length 0002], warning ...  close_notify
Peer: Success received.
Peer:  Success.
Authentication success detected at Peer
Welcome to the world, testusertls@localdomain1.net !!!
Peer TLS session key
LENGTH cad 64
 5f  d6  e9  dc  e4  77  b3  17  79  5e  c5  8f  4b  78  0f  d7  84  3f
71  2e  ae  6a  40  b2  af  44  91  ae  3b  7e  0b  b1  21  24  4e  16
43  05  58  ae  42  9f  b8  bc  cf  3f  ea  1e  3d  47  d5  c9  45  3b
fc  3e  02  7c  8f  5c  8e  32  21  31
```

6.5.2. Trazas obtenidas con Wireshark

Se analizaran ahora las trazas obtenidas con Wireshark en la comunicación entre el NAS y el servidor de AAA. Las trazas entre el cliente PANA y el NAS son muy similares a las mostradas en el caso de MD5 (aunque con mayor número de mensajes) por lo que no se considera necesario incluirlas en este documento. Nota: Del mismo modo que no hay parche para Diameter-EAP tampoco lo hay para EAP-TLS. El análisis se hará manualmente.

1. Primer mensaje: Identity (NAS — HSS)

   Contiene los siguientes AVPs:

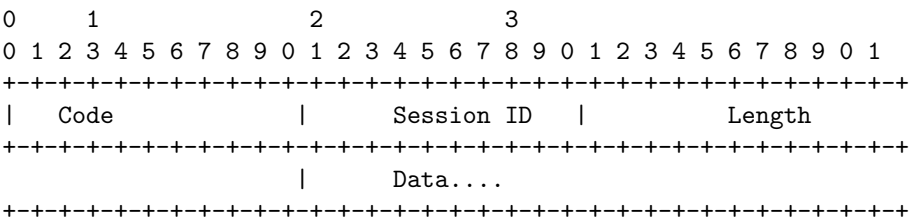
   - Session-Id(263)
   - Auth-Application-Id (258) indicando el número "5"(Diameter-Eap)
   - Origin-Host (264) localnas.localdomain2.net
   - Origin-Realm (296) localdomain2.net
   - Destination-Realm (284) localdomain1.net
   - Auth-Request-Type(274) con el valor "3"(Authorize and Authenticate)
   - EAP-Payload (462): Contiene la identidad del Peer. Es igual que en el caso de MD5

- User-Name (1) testuser
- Authorization-Lifetime (291) 30
- Session-Time Out (27) 30

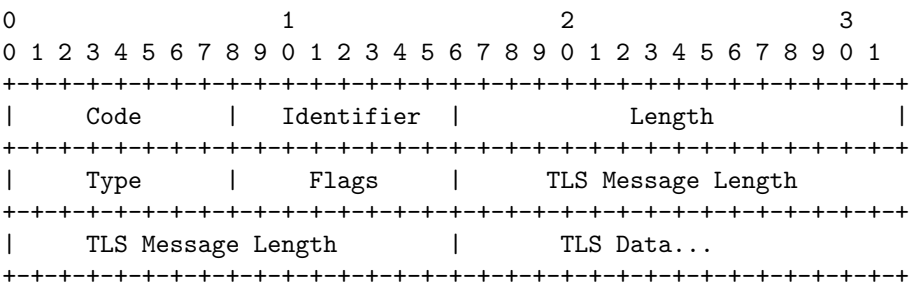
2. Segundo mensaje: EAP-TLS Start (HSS — NAS)

- Session-Id(263)
- Auth-Application-Id (258) indicando el número "5"(Diameter-Eap)
- Origin-Host (264) localnas.localdomain2.net
- Origin-Realm (296) localdomain2.net
- Destination-Realm (284) localdomain1.net
- Auth-Request-Type(274) con el valor "3"(Authorize and Authenticate)
- User-Name (1) testuser
- Authorization-Lifetime (291) 30
- Session-Time Out (27) 30
- Result-Code (268): DIAMETER_MULTI_ROUND_AUTH (1001)
- EAP-Payload (462) (ver después)

El EAP-Payload tiene la siguiente estructura genérica:



Dentro del campo marcado como Data, hay datos de EAP-TLS, lo que hace que la estructura de este campo sea en realidad:



y la estructura de los flags de TLS es:

```
0 1 2 3 4 5 6 7 8
+-+-+-+-+-+-+-+-+
|L M S R R R R R|
+-+-+-+-+-+-+-+-+
```

Analizando el EAP-Payload manualmente obtenemos:

a) Code: 01

b) Identifier:02

c) Length: 0006

d) Type: 0D, que en decimal es 13 (TLS)

e) Flags: 20 = 00100000, es decir, flag Start a 1. Se trata de un mensaje EAP-TLS-Start.

f) TLS Data: vacío

A partir de este momento, dado que hay un gran número de mensajes y que la información sobre TLS está contenida en el AVP EAP-Payload; realizaremos el análisis únicamente y de manera manual de este AVP.

3. Tercer Mensaje: Client-Hello (NAS — HSS) análisis del EAP-Payload AVP:

   a) Code: 02

   b) Identifier:02

   c) Length: 005D

   d) Type: 0D, que en decimal es 13 (TLS)

   e) Flags: 00 = 00000000, es decir, ningún flag activo, el mensaje no tiene más fragmentos

   f) TLS Data: Client-Hello.

4. Cuarto mensaje: Server-Hello + Datos de certificado y conexión (1/3) (HSS — NAS)

   análisis del EAP-Payload AVP:

   a) Code: 01

   b) Identifier:03

   c) Length: 01F4

d) Type: 0D, que en decimal es 13 (TLS)

e) Flags: C0 = 11000000, es decir, el mensaje incluye campo de longitud de TLS y está fragmentado.

f) TLS-length: 0000059B

g) TLS Data: Server-Hello + datos de certificado y conexión[3]

5. Quinto mensaje: Client-ACK (NAS — HSS)

   análisis del EAP-Payload AVP:

   a) Code: 02

   b) Identifier:03

   c) Length: 0006

   d) Type: 0D, que en decimal es 13 (TLS)

   e) Flags: 00 = 00000000, es decir, ningún flag activo, el mensaje no tiene más fragmentos

   f) TLS Data: Vacío. Se trata de un ACK[4]

6. Sexto mensaje: Datos de certificado y conexión del servidor (continuación 2/3) (HSS — NAS)

   análisis del EAP-Payload AVP:

   a) Code: 01

   b) Identifier:04

   c) Length: 01F4

   d) Type: 0D, que en decimal es 13 (TLS)

   e) Flags: 40 = 01000000, es decir,el mensaje tiene más fragmentos

   f) TLS Data: Datos de certificado y conexión.

7. Séptimo mensaje: Client-ACK (NAS — HSS)

   análisis del EAP-Payload AVP:

   a) Code: 02

[3]Puede verse como este mensaje contiene datos de certificado y conexión viendo las trazas legibles en wireshark; en el anterior mensaje no podía leerse nada, sin embargo, en este, pueden leer palabras tales como Certificate, CA o UC3M.

[4]Como pudimos comprobar en el RFC de EAP-TLS[1], los ACKs se producen a nivel EAP-TLS.

b) Identifier:04

c) Length: 0006

d) Type: 0D, que en decimal es 13 (TLS)

e) Flags: 00 = 00000000, es decir, ningún flag activo, el mensaje no tiene más fragmentos

f) TLS Data: Vacío. Se trata de un ACK

8. Octavo mensaje: Datos de conexión y certificado del servidor (continuación 3/3) (HSS — NAS)

   análisis del EAP-Payload AVP:

   a) Code: 01

   b) Identifier:05

   c) Length: 01C9

   d) Type: 0D, que en decimal es 13 (TLS)

   e) Flags: 00 = 00000000, es decir, ningún flag activo, el mensaje no tiene más fragmentos

   f) TLS Data: Datos de certificado del servidor y conexión.

9. Noveno mensaje: Datos del certificado del cliente y de conexión (1/4) (NAS — HSS)

   análisis del EAP-Payload AVP:

   a) Code: 02

   b) Identifier:05

   c) Length: 01F4

   d) Type: 0D, que en decimal es 13 (TLS)

   e) Flags: C0 = 11000000, es decir, el mensaje tiene más fragmentos e incluye el campo TLS-Length

   f) TLS-Length = 0000063D

   g) TLS Data: Datos de certificado del cliente y de conexión.

10. Décimo mensaje: Server-ACK (HSS — NAS)

    análisis del EAP-Payload AVP:

    a) Code: 01

b) Identifier:06

c) Length: 0006

d) Type: 0D, que en decimal es 13 (TLS)

e) Flags: 00 = 00000000, es decir, ningún flag activo, el mensaje no tiene más fragmentos

f) TLS Data: Vacío. Se trata de un ACK

11. Décimo-primer mensaje: Datos del certificado del cliente y conexión (continuación 2/4) (NAS—-¿HSS)

    análisis del EAP-Payload AVP:

    a) Code: 02

    b) Identifier:06

    c) Length: 01F4

    d) Type: 0D, que en decimal es 13 (TLS)

    e) Flags: 00 = 01000000, es decir, el mensaje tiene más fragmentos

    f) TLS Data: Datos del certificado del cliente y la conexión.

12. Décimo-segundo mensaje: Server-ACK (HSS — NAS)

    análisis del EAP-Payload AVP:

    a) Code: 01

    b) Identifier:07

    c) Length: 0006

    d) Type: 0D, que en decimal es 13 (TLS)

    e) Flags: 00 = 00000000, es decir, ningún flag activo, el mensaje no tiene más fragmentos

    f) TLS Data: Vacío. Se trata de un ACK

13. Décimo-tercer mensaje: Datos del certificado del cliente y conexión (3/4) (NAS — HSS)

    análisis del EAP-Payload AVP:

    a) Code: 02

    b) Identifier:07

    c) Length: 01F4

d) Type: 0D, que en decimal es 13 (TLS)

e) Flags: 40 = 01000000, es decir, el mensaje tiene más fragmentos

f) TLS Data: Datos de certificado del cliente y conexión.

14. Décimo-cuarto mensaje: Server-ACK (HSS — NAS)

    análisis del EAP-Payload AVP:

    a) Code: 01

    b) Identifier:08

    c) Length: 0006

    d) Type: 0D, que en decimal es 13 (TLS)

    e) Flags: 00 = 00000000, es decir, ningún flag activo, el mensaje no tiene más fragmentos

    f) TLS Data: Vacío. Se trata de un ACK

15. Décimo-quinto mensaje: Datos del certificado del cliente y conexión (4/4) (NAS — HSS):

    análisis del EAP-Payload AVP:

    a) Code: 02

    b) Identifier:08

    c) Length: 007D

    d) Type: 0D, que en decimal es 13 (TLS)

    e) Flags: 00 = 00000000, es decir, ningún flag activo, el mensaje no tiene más fragmentos

    f) TLS Data: Datos del certificado del cliente y la conexión.

16. Décimo-sexto mensaje: Datos del cypher-Spec y terminación de la fase de autenticación (HSS — NAS)

    análisis del EAP-Payload AVP:

    a) Code: 01

    b) Identifier:09

    c) Length: 0041

    d) Type: 0D, que en decimal es 13 (TLS)

    e) Flags: 00 = 00000000, es decir, ningún flag activo, el mensaje no tiene más fragmentos

f) TLS Data: Datos.

17. Décimo-séptimo mensaje: Client-ACK (NAS — HSS)

    análisis del EAP-Payload AVP:

    a) Code: 02

    b) Identifier:03

    c) Length: 0006

    d) Type: 0D, que en decimal es 13 (TLS)

    e) Flags: 00 = 00000000, es decir, ningún flag activo, el mensaje no tiene más fragmentos

    f) TLS Data: Vacío. Se trata de un ACK

18. Décimo-octavo mensaje: Mensaje de fin de conexión: Éxito (HSS — NAS)[5]

    análisis del EAP-Payload AVP:

    a) Code: 03 (Éxito)

    b) Identifier:09

    c) Length: 0004

6.5.3. Conclusiones de las trazas

1. El número del identificador de la conexión (campo Identifier) es monótonamente creciente y es dictado por el HSS incluso cuando envía ACKs.

2. El campo TLS-Length sólo aparece en el primer fragmento del mensaje enviado (por ejemplo, si el mensaje tiene 3 fragmentos, sólo aparece en el primero de los 3)

3. Los mensaje son asentidos por el otro extremo.

4. El éxito o fracaso se produce a nivel EAP y no a nivel EAP-TLS. EAP-TLS va sobre EAP.

[5]Es importante destacar como el mensaje de Éxito se produce a nivel EAP (y no a nivel EAP-TLS) y no recibe Ack, ésta era una de las complejidades de seguridad que mencionaba el RFC de EAP[2] en este tipo de mensajes

5. Aún pudiendo realizar el análisis con un parche de wireshark, los datos específicos de conexión o certificados no habrían podido ser analizados por el mismo. Las pérdidas por el análisis manual son prácticamente nulas.

6. Los mensajes se ajustan a la perfección al esquema de trazas marcado en la figura del principio de la sección.

7. De todas estas conclusiones podemos deducir que, sin tener en cuenta las trazas por pantalla de las aplicaciones y teniendo sólo en cuenta los mensajes realmente enviados por la red, la autenticación con TLS ha sido correcta.

6.5.4. Comparativa de MD5 y TLS

Para realizar una comparativa adecuada entre la autenticación con MD5[20] y la autenticación con EAP-TLS encontramos un problema importante: No podremos realizar una comparativa real de tiempos. Esto de debe a que no tenemos una implementación de EAP-TLS en remoto, ni en un escenario real, como es el core IMS.
Se realizará la comparativa de los aspectos a los que puede accederse:

- Número y longitud de los mensajes: Éste es el aspecto más evidente. En el caso de MD5 teníamos 4 mensajes y en el caso de EAP-TLS tenemos 18, de los cuales, algunos van fragmentados. El aumento de complejidad es evidente.

- Seguridad: MD5 no es un método de cifrado, sino una función de hash, es, por tanto, más sencillo de romper que TLS y mucho menos seguro.

- Flexibilidad: EAP-TLS es mucho más flexible. Permite realizar una negociación de la suite de cifradores que se usará en la conexión

- Eficiencia: A falta de información temporal, suponemos que MD5 es más eficiente, por el número de mensajes mucho menor que emplea.

Parte IV

Pruebas, historia del proyecto y Conclusiones

Capítulo 7

Pruebas

En esta sección se explicará el proceso de pruebas llevado a cabo para cada uno de los apartados del proyecto (MD5 en local, MD5 en remoto y TLS en local).

Es importante destacar que, debido a problemas de espacio, sólo se tratarán en esta sección la descripción de las pruebas pasadas y las conclusiones. Para más detalles sobre las pruebas, en el último anexo hay un dossier detallado, con trazas y demás información relevante referente a las mismas.

7.1. Pruebas del Escenario con MD5 en local

7.1.1. Pruebas del NAS y el servidor AAA sin cliente

Para probar estos dos elementos, se efectuarán las siguientes pruebas, en total 9:

- Prueba 0.1 : NAS y servidor AAA sin parámetros: En esta prueba arrancaremos el NAS y el servidor AAA sin pasarles ningún parámetros por la linea de comandos.

- Prueba 0.2 : NAS y servidor AAA con parámetros: En esta prueba arrancaremos el NAS y el servidor AAA pasándoles el fichero de configuración por la línea de comandos, e incluyendo otra serie de parámetros detrás de este.

- Prueba 1 : Arranque de NAS y servidor AAA en distinto orden.

- Prueba 2 : Incongruencias en los ficheros de diccionario del NAS y del servidor AAA: Esta prueba consiste en tener ficheros de diccionario de Diameter-EAP distintos en el NAS y el servidor AAA, por ejemplo, tener en uno de ellos un flag "mandatory"que no aparece en el otro (debe modificarse algún AVP que sepamos que será usado).

- Prueba 3 : Conexión de varios NAS con el mismo servidor AAA: Esta prueba consiste en conectar varios NAS al mismo servidor AAA. Hay que tener en cuenta que si se realiza en local habrá que asignar puertos distintos a cada uno de ellos y modificar el fichero de configuración del AAA para que todos ellos figuren como Peers.

- Prueba 4: Conexión del NAS a varios servidores AAA.

- Prueba 5: Intento de comunicación con ninguna aplicación común: Esta prueba consiste en intentar comunicar un NAS y un servidor AAA con ninguna aplicación en común (dentro de los ficheros de configuración, la parte marcada como auth_application_id).

- Prueba 6: Intento de comunicación sin métodos de seguridad comunes.

- Prueba 7: Comprobación de temporizadores.

7.1.2. Pruebas del cliente PANA en local

En esta sección se tratarán las pruebas sobre el cliente PANA programado para el proyecto. Las pruebas siempre deben llevarse a cabo en un escenario en el que las comunicaciones entre el NAS y el servidor AAA estén libres de errores.

En total, se llevarán a cabo 12 pruebas, que son las siguientes:

- Prueba 0: Intento de conexión con un NAS fuera de linea.

- Prueba 1: Conexión de un cliente no perteneciente al dominio (Realm).

- Prueba 2: Nombre de usuario correcto pero método de autenticación distinto en el cliente y el servidor AAA.

- Prueba 3: Password Incorrecta.

- Prueba 4: Todos los parámetros correctos.

- Prueba 5: Batería de clientes.

- Prueba 6: Servidor AAA fuera de línea.
- Prueba 7: Conexión directa del cliente PANA con el servidor AAA (Sin pasar por el NAS).
- Prueba 8: Llamada al cliente PANA sin parámetros.
- Prueba 9: Llamada al cliente PANA con un fichero de configuración erróneo o inexistente.
- Prueba 10: Gestión de temporizadores.
- Prueba 11: Distinto MD5-Digest: Esta prueba consiste en calcular de manera distinta el MD5-Digest en el cliente PANA y en el servidor de AAA.

7.1.3. Errores de comunicación entre el NAS y el cliente PANA

Estos errores suceden de manera aleatoria independientemente de la prueba que se intente pasar.
Consisten en que, a veces, con la comunicación entre el NAS y el servidor AAA correctamente establecida, al intentar introducir el cliente en la comunicación el NAS interrumpe su funcionamiento.
No se pudo determinar cuando y porqué sucedían estos errores, pero se pudieron capturar tres mensajes de error distintos:

1. Primer error:

```
terminate called without an active exception
PassThrough: Trying a legacy method.
Cancelado
```

2. Segundo error:

```
nasd: /usr/include/boost/shared_array.hpp:87: T&
               boost::shared_array<T>::operator[]
                (ptrdiff_t) const [with T = std::
                 list<int, std::allocator<int> >]:
                        Assertion ‘px != 0’ failed.
(14574|3045059472) From state: OFFLINE to WAIT_SUCC_PBA
Cancelado
```

3. Tercer error:

```
Virtual pure method called
Fallo de segmentación
```

7.1.4. Conclusiones

En general, las conclusiones a estas pruebas son positivas. El cliente PANA funciona bien ante las eventualidades y se comunica correctamente con el NAS. Tenemos bastante autonomía sobre él sin tocar el código gracias a los ficheros de configuración y a los parámetros de la línea de comandos y la autenticación es muy rápida.
Sin embargo, las comunicaciones entre el NAS y el servidor AAA no son todo lo buenas que desearíamos, especialmente por el inconveniente de que, a pesar de intercambiar capacidades, a veces no se intercambien el Watchdog. Cuando se cumple este hecho, se puede comprobar como los mensajes del cliente PANA no llegan hasta el servidor AAA haciendo imposible la comunicación.
También hay que tener en cuenta el problema de comunicación aleatorio entre el NAS y el cliente PANA, del cual no pudo determinarse la causa. A pesar de ésto, dado que el código del NAS y del servidor de AAA son propios de OpenDiameter y que no han sido modificados en este proyecto, podemos dar por superada satisfactoriamente esta fase.

7.2. Pruebas del escenario con MD5 en remoto

7.2.1. Pruebas del NAS y el servidor HSS sin cliente

Estas pruebas comprobarán la comunicación entre el NAS y el HSS sin interacción con el cliente. En total son 7 pruebas:

- Prueba 1 : Arranque de NAS y HSS en distinto orden.
- Prueba 2 : Incongruencias en los ficheros de diccionario del NAS y el HSS.
- Prueba 3 : Conexión de varios NAS con el mismo HSS.
- Prueba 4 : Conexión del NAS a varios HSS.
- Prueba 5: Intento de comunicación con ninguna aplicación común.
- Prueba 6: Intento de comunicación sin métodos de seguridad comunes.
- Prueba 7: Comprobación de temporizadores.

7.2.2. Pruebas del cliente PANA en el escenario remoto

Se trata de las mismas pruebas que se efectuaron en local, es decir:

- Prueba 0: Intento de conexión con un NAS fuera de linea.
- Prueba 1: Conexión de un cliente no perteneciente al dominio (Realm).
- Prueba 2: Nombre de usuario correcto pero método de autenticación distinto en el cliente y el servidor AAA.
- Prueba 3: Password Incorrecta.
- Prueba 4: Todos los parámetros correctos.
- Prueba 5: Batería de clientes.
- Prueba 6: Servidor AAA fuera de línea.
- Prueba 7: Conexión directa del cliente PANA con el servidor AAA (Sin pasar por el NAS).
- Prueba 8: Llamada al cliente PANA sin parámetros.
- Prueba 9: Llamada al cliente PANA con un fichero de configuración erróneo o inexistente.
- Prueba 10: Gestión de temporizadores.
- Prueba 11: Distinto MD5-Digest: Esta prueba consiste en calcular de manera distinta el MD5-Digest en el cliente PANA y en el servidor de AAA.

7.2.3. Conclusiones

El resultado de las pruebas es bastante satisfactorio. A pesar de haber programado un interfaz Diameter-EAP tan limitado en el HSS, el hecho de utilizar otro interfaz para los mensajes de Diameter Base Protocol hace que, salvo en la prueba número 1, no haya ningún error grave en las pruebas.
Podría dedicarse más tiempo a mejorar la implementación del HSS, pero dado que no era un objetivo original de este proyecto programar dicha interfaz, prefirió dejarse así y centrarse con la autenticación con TLS.

El error de comunicación entre el NAS y el cliente PANA comentado en el apartado referido a las pruebas en local sigue produciéndose de manera aleatoria.

El hecho de que se produzcan estos errores, unido a las dificultades para instalar las bibliotecas con TLS y a que las bibliotecas dejaron de mantenerse hace 2 años, reafirman la idea de que las bibliotecas OpenDiameter no son una opción muy acertada para trabajar.

7.3. Pruebas con EAP-TLS

Una vez terminado el escenario con EAP-TLS y comprobada que al autenticación básica funciona es necesario probar el escenario conseguido.
Dado que el servidor de AAA con TLS es distinto, volveremos a pasar las pruebas al NAS y al servidor AAA por separado. Posteriormente se añadirán pruebas de funcionalidad con el cliente PANA conectado.

Nota importante: Es recomendable consultar el anexo de manual de usuario para iniciar ambas partes.

7.3.1. Pruebas del NAS y el servidor AAA sin cliente

Dado que los programas son muy similares a los probados en el caso de MD5, se pasará un conjunto más reducido de test, ya que pasar muchas de las pruebas sería redundante.
En total son 3 pruebas :

- Prueba 1 : Arranque de NAS y el servidor AAA en distinto orden.
- Prueba 2: Intento de comunicación con ninguna aplicación común.
- Prueba 3: Intento de comunicación sin métodos de seguridad comunes.

7.3.2. Pruebas del cliente TLS

Para la realización de estas pruebas se asume que el servidor de AAA y el NAS están correctamente conectados.

En total son 8 pruebas:

- Prueba 1: Método de autenticación incorrecto: En esta prueba intentaremos autenticar un cliente de PANA con MD5 en un servidor que sólo soporta EAP-TLS.
- Prueba 2: Autenticación correcta.
- Prueba 3: certificado incorrecto: En esta prueba se intentará la autenticación TLS con un certificado mal construido (en el lado del cliente o en el del servidor).
- Prueba 4: Clave privada de usuario incorrecta.
- Prueba 5: Ficheros inexistentes: En esta prueba se intentará la autenticación TLS con alguno de los ficheros necesarios no presente.
- Prueba 6: Rutas de los ficheros mal especificadas.
- Prueba 7: CA desconocida. Uno de los extremos no conoce a al autoridad de certificación que se usará.
- Prueba 8: Certificados caducados.

7.3.3. Conclusiones a las pruebas de EAP-TLS

En general, el resultado de las pruebas de EAP-TLS es satisfactorio. Podrían haberse realizado más pruebas sobre situaciones erróneas, pero no se considera necesario por estar recogidos en el documento los casos más importantes.
Por otro lado, los errores más importantes en este apartado han sido los problemas de comunicación entre el NAS y el servidor AAA, que ya se reproducían en el caso de MD5.

Capítulo 8

Historia del proyecto

El proyecto estuvo marcado por los problemas con las bibliotecas OpenDiameter.
La primera fase del proyecto fue la lectura de los RFCs correspondientes y la instalación de OpenDiameter. Esta fase consumió aproximadamente un mes de trabajo.
La segunda fase consistió en la realización de las pruebas de las bibliotecas de OpenDiameter y la inmersión en el código, sobre todo desde el punto de vista de los objetivos originales del proyecto, que cambiaron con posterioridad:

1. En primer lugar, analizar el funcionamiento de distintos protocolos usados para la autenticación de clientes en servidores.

2. En segundo lugar, realizar un análisis de las bibliotecas de OpenDiameter, que se usarán para la realización del trabajo.

3. La creación de un cliente PANA capaz de autenticarse con el servidor de IMS usando el protocolo EAP y el sistema de autenticación MD5.

4. Exploración de otros posibles protocolos de autenticación.

5. Utilización de EAP-TLS sobre EAP.

6. Utilización de otros protocolos.

7. Realización de pruebas sobre los distintos protocolos usados.

8. Análisis de los resultados de las pruebas y obtención de conclusiones.

Esta fase consumió aproximadamente 20 días de trabajo.
En una tercera fase se pretendía programar el primer cliente PANA con MD5 para lograr una autenticación en local. Debido a las complicaciones

para configurar el NAS y el servidor AAA para que se comunicarán en local, y, pensando que el HSS sí tenía implementada la interfaz Diameter-EAP, se intentó pasar directamente a la autenticación en remoto.
El hecho de que tanto el intercambio de capacidades como los Wacthdog fueran correctos con el HSS, hizo pensar que no habría problemas con Diameter-EAP, sin embargo, dado que el HSS no contestaba a ninguna petición del cliente, se decidió volver de nuevo a la autenticación en local, para poder realizar un análisis más exhaustivo de los mensajes.
En esta vuelta al principio de la fase, se logró la autenticación en local y se descubrió que el interfaz Diameter-EAP era inexistente en el HSS.
Al final, esta fase duró aproximadamente un mes y medio.
La cuarta fase consistió en la programación del módulo Diameter-EAP en el HSS para lograr la autenticación en remoto. Esta fase duró aproximadamente 15 días.
Como quinta fase, se pasaron las pruebas, tanto en local como en remoto a la fase de autenticación MD5. Se programaron los scripts de pruebas. La duración fue de 5 días aproximadamente.
Una vez conseguidas las autenticaciones con MD5, se pasó a la parte de TLS. La instalación y funcionamiento de las bibliotecas y los clientes y servidores con TLS fue muy complicada, durando aproximadamente un mes. Conseguida la instalación de TLS y la configuración correcta de todos los elementos, la migración del cliente MD5 al cliente TLS fue casi automática, y las pruebas con TLS duraron del orden de dos semanas.

Debido al tiempo invertido con las bibliotecas OpenDiameter y a la programación del interfaz Diameter-EAP en el HSS, el desarrollo final del proyecto quedó como el indicado en objetivos del proyecto:

1. Analizar el funcionamiento de distintos protocolos usados para la autenticación de clientes en servidores:
   Se analizarán protocolos de distintos niveles y su integración mutua. Este análisis pondrá un especial interés en Diameter[16], ya que es el protocolo base usado en las autenticaciones en IMS[7].

2. Realizar un análisis de las bibliotecas de OpenDiameter, que se usarán para la realización del trabajo: Se trata de bibliotecas de código abierto que, tal y como se explica en posteriores capítulos, proporcionan una implementación en el lenguaje C++ de los protocolos que se usarán.

3. Creación de un cliente PANA capaz de autenticarse con el servidor de IMS[7] (HSS): Usando el protocolo EAP[2] sobre PANA[2] y el sistema de autenticación MD5[20], se pretende crear un cliente que pueda ser

autenticado en el entorno de IMS, a través de un servidor intermedio. Este es el cliente más básico que podemos crear con las especificaciones dadas, como puede comprobarse en la memoria, la autenticación se consigue en un número reducido de mensajes, y sin la intervención de ninguna entidad externa.

4. Utilización de EAP-TLS, en lugar de MD5: Conseguida la autenticación básica, se procederá a la integración en el cliente de un protocolo más complejo, y que ofrece un nivel se seguridad mucho mayor, al basarse en criptografía asimétrica y certificados de seguridad.

5. Realización de pruebas sobre los distintos escenarios: Terminadas las implementaciones, se probará el código creado y las contingencias específicas de cada escenario.

6. Análisis de los resultados de las pruebas y obtención de conclusiones: Se obtendrán los resultados de las pruebas y se comentarán los errores conseguidos, es importante en este punto ser críticos; tanto con el código programado, como con las librerías de OpenDiameter.

Capítulo 9

Conclusiones y trabajos futuros

9.1. Consecución de objetivos

La consecución de objetivos del trabajo ha sido satisfactoria: Se ha conseguido la autenticación con TLS que tantos problemas dio al principio, y la sensación conseguida es que, el introducir en el cliente otro método de autenticación de los recogidos en OpenDiameter sería automático.

Considerando los objetivos propuestos:

- Analizar el funcionamiento de distintos protocolos usados para la autenticación de clientes en servidores: Este objetivo puede darse por cumplido. Tanto el trabajo teórico previo, como el trabajo práctico en los diversos protocolos; que incluye la comprensión del código, la implementación de EAP en el lado del HSS y la comprobación con wireshark de que los mensajes coinciden con los esperados, favorecieron una buena inmersión en los protocolos usados y su funcionamiento.
- Realizar un análisis de las bibliotecas de OpenDiameter, que se usarán para la realización del trabajo: Este objetivo también se considera cubierto. Todo el trabajo se basó en estas bibliotecas, y, debido a sus errores, fue fundamental comprender bien su funcionamiento para poder llevar adelante el proyecto.
- Creación de un cliente PANA capaz de autenticarse con el servidor de IMS[7] (HSS): Este objetivo se considera cubierto de manera parcial.

  El cliente PANA, con EAP y MD5, funciona correctamente e incluye todas sus funcionalidades, tal y como se observó en las pruebas en local. Sin embargo, la implementación del HSS no es completa.

El prototipo de HSS programado es muy simple y no incluye la funcionalidad completa del protocolo, obviando la gestión de temporizadores, entre otras cosas. Dado que se creía que el HSS funcionaría correctamente con Diameter-EAP y que no era un objetivo del proyecto; consideramos que la funcionalidad conseguida es suficiente: Se consiguen autenticaciones correctas para parámetros correctos, y el sistema rechaza a los clientes cuyo nombre de usuario o contraseña son incorrectos.

- Utilización de EAP-TLS, en lugar de EAP: Este objetivo se considera cubierto totalmente, con el inconveniente de que no pudo ser probado con el HSS, al no programarse una interfaz EAP-TLS para el mismo. En local la autenticación con EAP-TLS es completa y acorde con el RFC[1]

- Análisis de los resultados de las pruebas y obtención de conclusiones: En general, los resultados de las pruebas fueron satisfactorios y las conclusiones obtenidas son válidas y acordes con los RFCs. La parte referida a OpenDiameter se explica en la siguiente sección.

Con estas conclusiones, consideramos que no es necesario seguir con el desarrollo por tres motivos:

- OpenDiameter sólo incluye un método más de autenticación, siendo TLS el más complejo de los incluidos.

- Las bibliotecas OpenDiameter no están siendo mantenidas correctamente. Contactar con los creadores ha sido imposible y la última modificación se llevó a cabo en 2007, hace dos años.

- El HSS, con el que se pretendía que el cliente se comunicara, no tiene implementado Diameter-EAP, y antes de seguir adelante con este proyecto sería necesario conseguir una buena interfaz de Diameter-EAP en dicho servidor.

En resumen, los nuevos objetivos han sido alcanzados con un alto grado de satisfacción.

9.2. Impresiones tras la realización del proyecto

Las impresiones tras la realización del proyecto son:

1. Las bibliotecas de OpenDiameter no son todo lo buenas que se desearía. Sin embargo, no parece haber ninguna otra alternativa viable, ya que las otras versiones que se encontraron de Diameter o son de pago o tienen un mantenimiento peor que el de OpenDiameter.

   Las bibliotecas OpenDiameter Tienen una serie de virtudes y defectos que se analizaran a continuación.

   Virtudes:

   - La programación está estructurada y es bastante limpia. No aparecen métodos demasiado enrevesados ni largos. Las máquinas de estados, que son la parte más complicada, están comentadas.
   - Son amplias. Dan soporte a multitud de protocolos.
   - Dan mucha facilidad a la programación. Una vez comprendidas, crear clientes o servidores es sencillo, ya que casi la totalidad de los protocolos está implementada y suele bastar con implementar una serie de métodos definidos como abstractos y arrancar las máquinas de estados.
   - Son transparentes a niveles inferiores. Para establecer las conexiones, es suficiente con especificar puertos y direcciones en un fichero de configuración, en ningún momento es necesario trabajar con sockets o similares.
   - Los ficheros de configuración xml son muy prácticos, y es raro que sea necesario tocar el código para modificar parámetros.

   Defectos:

   - A pesar de que el código está bien estructurado, los ficheros suelen ser demasiado largos, incluyendo cada uno de ellos varias clases distintas. Ésto dificulta mucho seguir las implementaciones de los métodos y hace que sea fácil perderse en el código sin encontrar lo que se busca.
   - Es muy difícil bajar de nivel. La transparencia, que también se considera una virtud, hace complicado entender y encontrar el código referente a los niveles inferiores.
   - El mantenimiento por parte de los creadores es muy malo. Es imposible contactar con ellos y los foros están completamente parados. Por otro lado, existe muy poca información en la red al respecto.

- La instalación es demasiado compleja. Especialmente si se pretende usar TLS; tal y como queda patente en los anexos sobre instalación. Ésto, unido a la falta de información en la red, hace muy desalentador intentar trabajar con OpenDiameter.
- A pesar de que el código es correcto, los programas de ejemplo de OpenDiameter, en su mayoría, no funcionan o funcionan mal. Por ejemplo, el cliente PANA de pruebas funciona mal, el cliente PANA con TLS no compila, y el NAS y el servidor AAA funcionan, pero no siempre, y además, tal y como se explico en otra sección, a veces, dan errores de tipo Segmentation fault, o la comunicación entre ellos no llega a producirse.

2. Se ha conseguido una visión global de protocolos de seguridad bastante importante, y esperamos que este proyecto sirva de guía a quién en el futuro trabaje con dichos protocolos.

3. El cliente programado es válido, tal y como demuestran las pruebas. Además es funcional, ya que es sencillo cambiar el método de autenticación dentro del mismo.

En definitiva, el hecho de que los protocolos tratados sean más o menos recientes ha complicado el proyecto, pero los resultados obtenidos pueden ser útiles para seguir con el desarrollo en el tema de seguridad en IMS y la comprensión de los propios protocolos implicados.

9.3. Trabajos futuros

Hay varias lineas por las que este proyecto podría ser continuado.

1. Unificar los servidores y clientes de MD5 y TLS, dentro del marco de OpenDiameter, para tener una implementación común para ambos. Esto simplificaría el proceso de pruebas y, sobre todo, proporcionaría una implementación unificada, que siempre es deseable.
   El coste de este cambio dentro del cliente no sería muy elevado; bastaría con habilitar ambas máquinas de estados y a la llegada de un mensaje, multiplexarlo entre estas dos máquinas, en función del tipo de mensaje recibido. Habría que tener en cuenta que ambas máquinas de estados utilizan métodos comunes implementados de distinta manera, lo que podría complicar el desarrollo.
   En el caso del servidor sí sería más complejo: MD5 y EAP-TLS utilizan

en el lado del servidor los mismos métodos, implementados de distinta manera para cada caso. Sería necesario hacer comprobaciones dentro de cada método,o bien activar una u otra máquina de estados según el cliente que llegue al sistema. Ésto es común tanto en cliente como en servidor, sin embargo, en el servidor se produce en un número de métodos mucho mayor. Además, en el servidor habría que tener la capacidad de cambiar entre ficheros de configuración.

2. Completar y mejorar la implementación de Diameter-EAP en el servidor HSS.
   Sería otra ampliación útil. Consistiría en añadir las funcionalidades que faltan en MD5 y crear desde cero EAP-TLS.

   La primera parte no sería en exceso complicada, bastaría con seguir el RFC, pero sí sería trabajosa; añadir funcionalidades de temporizador implicaría, posiblemente, tener un nuevo hilo de ejecución que se encargara de regular los tiempos y enviar señales si estos se sobrepasan. Otra opción viable, sería hacer comprobaciones de tiempos dentro de los métodos (la primera es más correcta). En cualquier caso, no sería un cambio automático.
   Por otro lado, implementar EAP-TLS sí sería muy complejo. Como hemos podido comprobar, EAP-TLS añade mucha complejidad al sistema: Se pasa de 4 a 18 mensajes, aparece una entidad de certificación, aparece la posibilidad de fragmentar mensajes, se pasa de 2 máquinas de estados a 4... La complejidad aumenta en exceso, y con ello también aumentan las pruebas necesarias y la posibilidad de encontrar errores. No parece una salida lógica, y en caso de querer usar EAP-TLS en el HSS parece más lógico buscar otra implementación, ya probada y funcional.

3. Investigar más en protocolos de seguridad distintos para el cliente PANA.
   Hasta este momento tenemos dos protocolos de seguridad implementados, sin embargo, uno de ellos es demasiado simple y el otro podría ser demasiado complejo para determinadas aplicaciones. Podrían probarse otros protocolos de un nivel más intermedio.
   Otra opción viable es, ya que Diameter obliga a la implementación de IPSec en sus servidores, probar dicho protocolo.

4. Realizar un análisis más exhaustivo de tiempos y rendimientos.: Debido a las complicaciones con el escenario, los clientes sólo han podido ser

probados en PCs (que tiene una gran capacidad de computación) y, en algunos casos, en ámbito local, lo que no deja mucho margen para la extracción de características temporales.
Sería deseable probar el cliente dentro de dispositivos móviles (con una capacidad de computación mucho menor), ya que IMS está pensado para interactuar con móviles, y realizar un análisis temporal de los resultados, para comprobar si realmente merece la pena EAP-TLS, dada su complejidad.

En caso de haber tenido más tiempo, se podría elegir cualquier opción, excepto la segunda. Considero demasiado complejo integrar EAP-TLS en el HSS, como para poder abordarse de manera satisfactoria en un tiempo razonable. Probablemente, la última opción sería la más útil buscando resultados prácticos.

Apéndice A

Manual de instalación

En esta parte de la memoria se pretende realizar una guía de como instalar las bibliotecas OpenDiameter.

A.1. Manual de instalación sin TLS

Lo primero es descargar la release 1.0.7-i de la página www.opendiameter.org, una vez descargado descomprimir el archivador. Para construir las bibliotecas hay que cumplir una serie de requisitos:

1. Dependencias:

   - bibliotecas BOOST: Pueden obtenerse de www.boost.org o bien con el gestor de paquetes para distribuciones debian: apt-get install libboost-dev
   - bibliotecas ACE: Pueden descargarse desde el link de la página de OpenDiameter o desde el gestor de paquetes : apt-get install libace-dev

   Si hemos usado la opción del gestor de paquetes, ambas bibliotecas estarán colocada en /usr/include/

   Estableceremos las dependencias que requiere OpenDiameter (suponemos bash):

   ```
   export ACE_ROOT=/usr/include/
   export BOOST_ROOT= /usr/include/
   ```

2. Instalación:

   Una vez fijadas las variables de entorno, nos colocamos en el directorio raíz de OpenDiameter y hacemos:

```
./configure
```

Si todas las dependencias están bien fijadas, obtendremos al final una lista de los directorios preparados para ser compilados. Como root hacemos:

```
make
make install
```

Y con esto la instalación estaría terminada. Se recomienda, si algo falla, antes de empezar de nuevo, hacer:

```
make clean
```

Para asegurar que no quedan ficheros residuales.

A.2. Manual de instalación con TLS

La instalación de las bibliotecas con TLS es mucho más compleja y llevo mucho más tiempo de completar que la instalación sin TLS.
Esta guía pretende ser una guía útil para instalar OpenDiameter con TLS en el futuro.

El principal problema con TLS es que al parecer la versión 1.0.7-i de OpenDiameter está desactualizada con respecto a esta parte del código, así que obtendremos muchos errores de dependencias que habrá que solucionar a mano.

- Dependencias:

  Además de ACE y BOOST, para compilar las bibliotecas de TLS necesitamos usar XERCESC, así que habrá que instalarlo con el gestor de paquetes de la máquina y posteriormente teclear:

  ```
  export ACE_ROOT=/usr/include/
  export BOOST_ROOT= /usr/include/
  export XERCESCROOT=/usr/include/
  ```

- Instalación:

  Teclearemos desde el directorio raíz de OpenDiameter:

  ```
  ./configure --with-eap-tls
  ```

y posteriormente, como root:

```
make
```

Aquí encontraremos el primer error de dependencias: Necesitamos los ficheros xml.h y xml_errorreporter.h, no incluidos en OpenDiameter. Podemos descargarlos desde aquí:

```
xml_errorreporter.h: http://enable.tssg.org/cgi-bin/trac.cgi/browser
     /coderepos/Bootstrapping%20and%20load%20sharing/HA/
      OpenDiameter%20(MIP6%20application)
     /libdiamparser/xml_errorreporter.h?rev=33
xml. h:  http://enable.tssg.org/cgi-bin/trac.cgi/browser
     /coderepos/Bootstrapping%20and%20load%20sharing/HA
     /OpenDiameter%20(MIP6%20application)/libdiamparser/xml.h
```

Depositaremos los ficheros en /libeap/include.

Una vez realizado este paso necesitamos fijar algunas otras dependencias perdidas en los ficheros .cxx pertenecientes a la carpeta /libeap/include/

En cada fichero referido a tls añadiremos en las dependencias:

```
#include "../libdiamparser/include/diameter_parser.h"
```

Y modificaremos las dependencias a las bibliotecas de xercesc para que apunten a rutas globales, por ejemplo cambiaremos:

```
#include <xercesc/dom/DOM.hpp>
```

Por:

```
#include "/usr/include/xercesc/dom/DOM.hpp"
```

Llegados a este punto, encontraremos un error bastante oscuro y ambiguo al compilar. Nos dirá que el compilador no puede crear referencias a vtable para ciertos objetos, para solucionar este error entraremos en xml_errorreporter.h y modificaremos las lineas desde la 129 a la 132, cambiando:

```
    void warning(const SAXParseException& toCatch);
    void error(const SAXParseException& toCatch);
    void fatalError(const SAXParseException& toCatch);
    void resetErrors();
```

Por :

```
void warning(const SAXParseException& toCatch){};
void error(const SAXParseException& toCatch){};
void fatalError(const SAXParseException& toCatch){};
void resetErrors(){};
```

Llegados a este punto solo obtendremos errores de compilación en el cliente de prueba incluido en /applications/pana/. Para corregirlo habrá que realizar una serie de cambios muy numerosa, dado que el fichero client_pana_eap_tls.cxx utiliza funciones obsoletas incluidas en la versión anterior de OpenDiameter, tras realizar estos cambios nos encontraremos la misma situación que tenía este cliente de prueba cuando no usábamos TLS, el código no funciona. Además será necesario descargar con el svn de nuevo los ficheros de configuración, ya que dichos ficheros no han sido incluidos en la distribución de OpenDiameter.
Debido a que este fichero no funciona correctamente, no consideramos necesario incluir aquí como puede hacerse que compile. El hecho de que use funciones obsoletas hace pensar que está desactualizado y fuera de mantenimiento.

Ahora será necesario corregir los errores de linkado, ya que los ficheros makefile generados por el comando configure son erróneos y no incluyen ninguna referencia a las bibliotecas xerces-c. Para corregir estos errores tendremos que modificar los ficheros makefile de los siguientes directorios:

```
libeap
libDiametereap
applications
applications/pana
```

(aunque aquí no es necesario ya que el código no funciona)

```
libpana o libpana2
```

(si hemos creado el directorio y colocado ahí nuestro cliente).
Los cambios consisten en cambiar las lineas:

```
L_LIBS = -lssl -lcrypto -lACE_SSL -lACE -lACEXML -lACEXML_Parser
```

Por:

```
L_LIBS = -lssl -lcrypto -lACE_SSL -lACE -lACEXML
                              -lACEXML_Parser -lxerces-c
```

Y las lineas del tipo:

```
client_pana_eaptls_SOURCES = client_pana_eap-tls.cxx
client_pana_eaptls_LDADD = -lssl -lcrypto -lACE_SSL
                       -lACE -lACEXML -lACEXML_Parser \
```

Por:

```
client_pana_eaptls_SOURCES = client_pana_eap-tls.cxx
client_pana_eaptls_LDADD = -lssl -lcrypto -lACE_SSL -lACE
                     -lACEXML -lACEXML_Parser -lxerces-c \
```

Nota: Este es un ejemplo, en general deberán modificarse de esta forma todas las lineas que hagan referencia a un ejecutable que usará funciones de tls.

A.2.1. Ajustes adicionales para hacer funcionar el fichero server_test_tls.cxx

Con los ajustes explicados, el servidor de AAA compilará y será capaz de intercambiar capacidades y Wacthdog con el NAS, sin embargo, no será capaz de procesar ninguna petición TLS.
Para conseguir que las peticiones TLS sean procesadas necesitamos:

1. Obtener los ficheros de configuración:
   El código utiliza una serie de ficheros que deben ser descargados, según la estructura de directorios que se explicará a continuación, sin embargo, el código no alerta ni de la ausencia de estos ficheros ni de errores en los mismos.
   El único mensaje de error que se escribe es:

   ```
   TlsStateMachine [Auth] is not running
   ```

   Que como puede apreciarse no es nada clarificador.
   La estructura de directorios y ficheros que debe descargarse debe colocarse en:

   ```
   /opendiameter-1.0.7-i/libdiametereap/test/tls_config/
   ```

   y su estructura es la siguiente:

```
                        |-demoCA    |- private  |cakey.pem
                        |ca-cert.pem
                        |dh
                        |john.key
                |- auth  |john.pem
                |        |random
                |        |svr-cert.pem
                |        |svr-key.pem
tls_config      |        |svr-key.pem.p1
                |
                |
                |        |clt-cert.pem
                |- Peer  |clt-key.pem
                |        |clt-key.pem.p1
                |        |ca-cert.pem
```

Los ficheros pueden obtenerse de la página de OpenDiameter.

Además, será necesario obtener de la página de OpenDiameter también el fichero eap-tls.configuration.xsd, que habrá que colocar en:

`/opendiameter-1.0.7-i/libdiametereap/config`

2. Obtener unos certificados no caducados.

   Los certificados descargados de la página de OpenDiameter caducaron en 2005, por lo que es necesario obtener unos que estén al día. En los ficheros proporcionados junto con este proyecto, estos certificados en fecha se encuentran contenidos en: certs.tgz, pero cualquier set será válido. Habrá que sustituir los viejos por los nuevos conociendo la siguiente leyenda:

   - ca-cert.pem : Certificado de la Certification Authority.
   - svr-cert.pem: Certificado del servidor.
   - svr-key.pem: Clave del servidor
   - svr-key-pem-p1: Certificado del servidor cifrado.
   - clt-cert.pem: Certificado del cliente.
   - clt-key.pem: Clave del cliente.
   - clt-key.pem.p1: Certificado del cliente cifrado.
   - ca-key.pem: Clave pública de la Certification Authority

Con esto queda instalado y configurado OpenDiameter para ser usado con tls.

IMPORTANTE: Durante la realización del proyecto traté varias veces de contactar con los creadores de las bibliotecas en referencia a la desactualización y problemas de instalación de las bibliotecas de tls, pero no obtuve ninguna respuesta.

Apéndice B

Configuración del NAS y el servidor de AAA

En esta sección se trata la configuración de estos dos elementos del escenario propuesto para lograr la comunicación entre ellos en un entorno local. La estructura de directorios de OpenDiameter se encuentra en un anexo de este documento. No obstante, es importante resumir aquí un fragmento de la misma:
Dentro del directorio OpenDiameter encontramos el directorio /applications y dentro de este directorio, los directorios /nas y /aaa. Dentro de cada uno de ellos encontraremos el nasd y el aaad respectivamente.
Cada uno de estos dos programas necesita una serie de ficheros de configuración para funcionar, estos fichero se encuentran contenidos en las carpetas /config de cada uno de ellos. Son ficheros en formato .xml
En el caso del nasd tenemos:

- nasd.xml Resume la configuración general del nas; donde buscará el fichero de diccionario, el fichero del interfaz PANA, etc.

- nasd_diameter_eap_dictionary.xml Este fichero actúa como diccionario de mensajes Diameter-EAP para el NAS. De este fichero obtendrá la información para construir los mensajes y parsear los que reciba.

- nasd_diameter_eap.xml A partir de este fichero, el NAS obtiene información de enrutamiento y de los Peers a los que se conecta mediante Diameter-EAP.

- nasd_pana_dictionary.xml Este fichero actúa como diccionario de mensajes PANA para el nas. De este fichero obtendrá la información para construir los mensajes y parsear los que reciba por la interfaz PANA.

- nasd_pana_paa.xml Este fichero contiene la configuración del nas para actuar como servidor PANA, el puerto de escucha, los temporizadores, etc.

En el caso del aaad tenemos algo parecido:

- aaad.xml Fichero general, principalmente contiene información sobre dónde buscar el resto de ficheros.
- aaad_diameter_dictionary.xml Diccionario de Diameter-EAP
- aaad_diameter_server.xml Parámetros de configuración de Diameter-EAP: Enrutamiento, Peers, etc.
- aaad_user_db.xml Este fichero contiene la información de los usuarios registrados en el sistema; sistema de autenticación, nombre de usuario, clave, etc.

Con estas descripciones, para conseguir la comunicación entre el nasd y el aaad deberemos actuar sobre los ficheros:
nasd_Diameter_eap.xml y nasd_Diameter_dictinary.xml en el caso del NAS y aaad_Diameter_server.xml y aaad_Diameter_dictionary.xml en el caso del servidor AAA.

B.1. Ajustes para el NAS y el AAA

Nota importante: La estructura de todos los ficheros tratados en esta sección se encuentra explicada y detallada en un anexo de este documento.

Un primer intento de ejecución del nasd y el aaad pone de manifiesto que los ficheros de configuración son erróneos.
El primer paso será corregir dichos ficheros y algunos errores de programación.

B.1.1. Recompilación del NAS

Dentro del fichero nasd_main.cxx, por algún motivo no determinado, las líneas referentes a Diameter-EAP están comentadas, con lo cual el NAS no dará soporte a Diameter-EAP.
Para solucionar ésto, dentro de dicho fichero, haremos los siguientes cambios:

```
 NASD_PanaInitializer apPanaInit;
-//NASD_DiameterEapInitializer aaaDiameterEapInit;
+NASD_DiameterEapInitializer aaaDiameterEapInit;
 NASD_EapBackendInitializer aaaEapBackendInit;
 NASD_PolicyScriptInitializer plcyScriptInit;

 std::string strApPanaName("pana");
-//std::string strAaaDiameterEapName("Diameter_eap");
+std::string strAaaDiameterEapName("Diameter_eap");
 std::string strAaaEapBackendName("local_eap_auth");
 std::string strPlcyScriptName("script");

 NASD_CnInitializer_I->Register(strApPanaName, apPanaInit);
-//NASD_CnInitializer_I->Register(strAaaDiameterEapName, aaaDiameterEapInit);
+NASD_CnInitializer_I->Register(strAaaDiameterEapName, aaaDiameterEapInit);
 NASD_CnInitializer_I->Register(strAaaEapBackendName, aaaEapBackendInit);
```

Los signos - significan líneas que deben ser eliminadas, y los signos + son las líneas por las que se sustituyen las eliminadas.
Con estos cambios, recompilamos el directorio /applications.

B.1.2. Ajustes generales

Dado que el fichero nasd.xml es el que controla al resto de ficheros de configuración, parece lógico empezar los ajustes en este punto.
Si abrimos el código del nasd nos encontramos la siguiente línea:

```
#define NASD_DEFAULT_CFG_FILE  "/etc/opendiameter/nas/config/nasd.xml"
```

En el fichero de código principal del aaad también encontramos:

```
#define AAAD_DEFAULT_CFG_FILE  "/etc/opendiameter/aaa/config/aaad.xml"
```

Y del mismo modo, dentro de los ficheros nasd.xml y aaad.xml observamos que las rutas a los demás ficheros son parecidas (/etc/opendiameter/ ...).
Dichos directorios no existen. Podríamos modificar las rutas, sin embargo, será más útil crear dichos directorios y copiar allí los ficheros. De tal manera que podamos tener dos sets de ficheros de configuración: uno en /etc/opendiameter/nas/config y el otro en el propio directorio del nasd[1]
Con esta idea en mente hacemos:

```
mkdir -p /etc/opendiameter/nas/

cp -r ./applications/nas/config/ /etc/opendiameter/nas/
cp -r ./applications/nas/scripts /etc/opendiameter/nas/
```

[1] Ésto será útil cuando intentemos realizar la conexión con el servidor remoto de IMS[7], ya que podremos tener la configuración en local y la configuración remota simultáneamente sin tocar ningún fichero.

```
mkdir -p  /etc/opendiameter/aaa/

cp -r ./applications/aaa/config/ /etc/opendiameter/aaa/
```

B.1.3. Fichero de diccionario

El fichero de diccionario del NAS está complemente desactualizado. Por suerte, este fichero puede obtenerse dentro de la carpeta de /libdiameter de OpenDiameter. Cambiamos el contenido del fichero nasd_diameter_eap_dictionary.xml por el del fichero diameter_dictionary.xml contenido en libdiameter/config (parte de diameter del OpenDiameter).
Una vez realizado este cambio, dentro del documento, sustituimos la línea:

```
<!DOCTYPE dictionary SYSTEM "dictionary.dtd">
```

Por:

```
<!DOCTYPE dictionary SYSTEM "nasd\_diameter\_eap\_dictionary.dtd">
```

Para tomar el fichero .dtd correcto.

B.1.4. Creación de hosts ficticios

Observando los fichero nasd_diameter_eap.xml y aaad_diameter_server.xml y con el estudio del protocolo Diameter realizado, concluimos que es necesario enrutar nuestros servidores por medio de Realms.
Para que la máquina sea capaz de hacer esto necesitamos crear unos hosts ficticios en el fichero de hosts del ordenador (/etc/host/). Para ello modificamos dicho fichero de tal manera que incluya:

```
127.0.0.1 localaaa localaaa.localdomain1.net
127.0.0.1 localnas localnas.localdomain2.net
```

De esta forma tenemos creadas las dos entidades:

- NAS = localnas.localdomain2.net
- AAA = localaaa.localdomain1.net

B.1.5. Enrutamiento

Necesitamos que el NAS y el servidor AAA puedan encontrarse el uno al otro para establecer la conexión. Hasta este momento tenemos los ficheros de diccionario y los hosts ficticios creados. En este apartado haremos que cada uno tenga al otro como Peer, de tal manera que puedan conectarse entre sí. Para conseguir esto tenemos que modificar el fichero nasd_diameter_eap.xml de la siguiente manera:

```
      </vendor_specific_application_id>
    </general>
    <parser>
-       <dictionary>config/nasd_Diameter_eap_dictionary.xml</dictionary>
+       <dictionary>/etc/OpenDiameter/nas/config/nasd_Diameter_eap_dictionary.xml</dictionary>
    </parser>
    <transport_mngt>
-       <identity>nas.access1.net</identity>
-       <Realm>access1.net</Realm>
+       <identity>localnas.localdomain2.net</identity>
+       <Realm>localdomain2.net</Realm>
        <tcp_listen_port>1811</tcp_listen_port>
        <sctp_listen_port>1810</sctp_listen_port>
        <use_ipv6>0</use_ipv6>
@@ -41,7 +42,7 @@
        <Peer_table>
            <expiration_time>1</expiration_time>
            <Peer>
-               <hostname>server.isp.net</hostname>
+               <hostname>localaaa.localdomain1.net</hostname>
                <port>3868</port>
                <use_sctp>0</use_sctp>
                <tls_enabled>0</tls_enabled>
```

Y el fichero aaad_Diameter_server.xml de esta manera:

```
      </vendor_specific_application_id>
    </general>
    <parser>
-       <dictionary>config/aaad_Diameter_dictionary.xml</dictionary>
+       <dictionary>/etc/OpenDiameter/aaa/config/aaad_Diameter_dictionary.xml</dictionary>
    </parser>
    <transport_mngt>
-       <identity>server.isp.net</identity>
-       <Realm>isp.net</Realm>
+       <identity>localaaa.localdomain1.net</identity>
+       <Realm>localdomain1.net</Realm>
        <tcp_listen_port>3868</tcp_listen_port>
        <sctp_listen_port>1813</sctp_listen_port>
        <use_ipv6>0</use_ipv6>
@@ -38,15 +38,16 @@
        <request_retransmission_interval>10</request_retransmission_interval>
        <max_request_retransmission_count>3</max_request_retransmission_count>
        <receive_buffer_size>2048</receive_buffer_size>
-       <advertised_hostname>server1.isp.net</advertised_hostname>
+       <advertised_hostname>localaaa.localdomain1.net</advertised_hostname>
        <Peer_table>
            <expiration_time>1</expiration_time>
            <Peer>
-               <hostname>nas.access1.net</hostname>
```

```
+               <hostname>localnas.localdomain2.net</hostname>
                <port>1811</port>
                <use_sctp>0</use_sctp>
                <tls_enabled>0</tls_enabled>
            </Peer>
+
        </Peer_table>
        <route_table>
            <expire_time>0</expire_time>
```

Desde este momento, el NAS y el servidor AAA están conectados. Como prueba de ello debería producirse el correspondiente intercambio de capacidades y el Wacthdog entre ambos.

Apéndice C

Manual de usuario

En este capítulo de los Anexos se explicará como arrancar cada uno de los ficheros ejecutables de los que se habla en la memoria.

- NAS. Hay tres opciones para arrancar el NAS: comunicándose con el servidor AAA local con MD5 o con TLS, o comunicándose con el HSS. En ambos casos habrá que colocarse en el directorio opendiameter-1.0.7-i/applications/nas y teclear:

  ```
  ./nasd (fichero de configuración).
  ```

  Para el fichero de configuración, si se han seguido los pasos de esta guía, hay tres opciones:

  1. Para el servidor AAA local sin TLS: Dejarlo en blanco, es decir, ./nasd sin parámetros. El NAS cogerá el fichero de configuración por defecto.
  2. Para el servidor AAA local con TLS: ./nasd ./config/nasdtls.xml
  3. Para el HSS: ./nasd ./config/nasd.xml.

- Servidor AAA. Hay dos opciones, sin TLS o con TLS.

  1. Sin TLS. Para arrancarlo nos colocaremos en /opendiameter-1.0.7-i/applications/aaa/ y teclearemos:

     ```
     ./aaad
     ```

  2. Con TLS. Para arrancarlo nos colocaremos en /opendiameter-1.0.7-i/libdiametereap/ y teclearemos:

     ```
     ./server_test_tls
     ```

- Cliente PANA. Para arrancarlos nos colocaremos en /opendiameter-1.0.7-i/libpana2/. Una vez aquí hay dos opciones:

1. Sin TLS: Teclearemos:

   ```
   ./clientepana parámetros.
   ```

2. Con TLS: Teclearemos:

   ```
   ./clientepanatls parámetros.
   ```

Los parámetros son iguales en los dos casos y son:

1. -f Fichero de configuración (lo normal será poner -f ./config/pac.xml)
2. -U Nombre de usuario (con la estructura nombre@dominio)
3. -P Password

El parámetro de fichero de configuración es obligatorio.

Apéndice D

Implementación del módulo Diameter-EAP en el HSS

En el presente anexo explicamos las modificaciones que tuvieron que realizarse sobre el HSS y se explica el porqué de las mismas.

D.1. Declaración de constantes

En el fichero DiameterConstants.java, se añadieron las siguientes líneas:
En el fichero DiameterConstants.java se añadió lo siguiente:

```
public static final int EAP = 5;
```

como atributo de la clase Application.

```
public static final int DER = 268;
```

como atributo de la clase Command.

```
public static final int EAP_PAYLOAD = 462;
public static final int AUTH_REQUEST_TYPE = 274;
public static final int AUTH_LIFE_TIME = 291;
public static final int AUTH_GRACE_PERIOD = 276;
public static final int SESSION_TIMEOUT = 27;
```

como atributos de la clase AVPCode.

Con estos cambios, tenemos las constantes necesarias declaradas.

D.2. Mensajes DER y DEA. El fichero DER.java

El código implementado para los mensajes DER y DEA es el siguiente:

```
package de.fhg.fokus.hss.EAP;
import java.util.Iterator;
import java.util.List;
import org.apache.log4j.Logger;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import java.security.NoSuchAlgorithmException;
import de.fhg.fokus.Diameter.DiameterPeer.DiameterPeer;
import de.fhg.fokus.Diameter.DiameterPeer.data.DiameterMessage;
import de.fhg.fokus.hss.cx.CxConstants;
import de.fhg.fokus.hss.cx.CxExperimentalResultException;
import de.fhg.fokus.hss.cx.CxFinalResultException;
import de.fhg.fokus.hss.db.model.IMPI;
import de.fhg.fokus.hss.db.model.IMPU;
import de.fhg.fokus.hss.db.op.IMPI_IMPU_DAO;
import de.fhg.fokus.hss.db.op.IMPU_DAO;
import de.fhg.fokus.hss.db.op.IMSU_DAO;
import de.fhg.fokus.hss.db.op.SP_IFC_DAO;
import de.fhg.fokus.hss.diam.DiameterConstants;
import de.fhg.fokus.hss.diam.UtilAVP;
import de.fhg.fokus.hss.db.hibernate.*;

/**
 * @author Guillermo Rodea Palomares
 * @author Davide Proserpio
 */
public class DER {
private static Logger logger = Logger.getLogger(DER.class);
public static DiameterMessage processRequest(DiameterPeer DiameterPeer,
                                             DiameterMessage request){
DiameterMessage response = DiameterPeer.newResponse(request);
response.flagProxiable = false;
```

Hasta aquí, creamos un mensaje nuevo y fijamos los flags.

```
  Session session = HibernateUtil.getCurrentSession();
  HibernateUtil.beginTransaction();
//Getting EAP_Payload Ientifier...
byte Identifier = UtilAVP.getEapPayloadField(request,1);
//Getting EAP_Payload type
  byte type = UtilAVP.getEapPayloadField(request,4);
```

Obtenemos el tipo del mensaje EAP contenido y el identificador de la sesión

```
    if (type == 1) {
    // identity received...
     try{
       UtilAVP.addAuthSessionState(response, DiameterConstants.
                              AVPValue.ASS_No_State_Maintained);
       UtilAVP.addAuthApplicationID(response, DiameterConstants.
```

```
                                                    Application.EAP);
        UtilAVP.addUserName(response,UtilAVP.getUserName(request));
        UtilAVP.addResultCode(response, DiameterConstants.
                  ResultCode.DIAMETER_MULTI_ROUND_AUTH.getCode());
        UtilAVP.addAuthLifeTime(response,UtilAVP.getAuthLifeTime
                                                          (request));
        UtilAVP.addAuthRequestType(response, UtilAVP.getAuthRequestType
                                                            (request));
        UtilAVP.addSessionTimeOut(response, UtilAVP.getSessionTimeOut
                                                            (request));
        UtilAVP.addEapPayload(response,request,HibernateUtil.
                                                   getCurrentSession());
       }catch (CxExperimentalResultException e){
         e.printStackTrace();
       }
     }
```

Si el mensaje es de tipo Identity, añadimos los AVPs necesarios, incluyendo el EAP-Payload, que contendrá el nonce creado por el servidor.

```
      else if (type == 4){
          UtilAVP.addAuthSessionState(response, DiameterConstants.
                                AVPValue.ASS_No_State_Maintained);
          UtilAVP.addAuthApplicationID(response,
                                DiameterConstants.Application.EAP);
          UtilAVP.addUserName(response,UtilAVP.getUserName(request));
          boolean success=false;
          try{
               success = UtilAVP.addEapPayload(response,request,
                                HibernateUtil.getCurrentSession());
           }catch(Exception e){
               System.out.println("Failed to add EAP-Payload");
               e.printStackTrace();
           }

       if(success){
          UtilAVP.addResultCode(response, DiameterConstants.ResultCode.
                                DIAMETER_SUCCESS.getCode());
       }else{
         UtilAVP.addResultCode(response, DiameterConstants.ResultCode.
                           DIAMETER_AUTHENTICATION_REJECTED.getCode());
       }
        UtilAVP.addAuthRequestType(response,
                                UtilAVP.getAuthRequestType(request));

       }
}
```

Si el mensaje tiene el tipo 4, se trata de un nonce procesado de MD5, así que comprobamos si el nonce es correcto, y en ese caso añadimos un AVP Result-

Code con el valor Success, y en caso contrario, añadimos el AVP con el valor de fallo.

D.3. La clase UtilAVP.java. Modificaciones necesarias

Para que el código propuesto funcionara hubo que modificar y añadir funciones en la clase UtilAVP.java. No se incluye aquí todo el código modificado, por ser demasiado extenso, pero sí se incluye la forma de generar el AVP 462, EAP-Payload que es el más importante para los mensajes.

```
public static boolean addEapPayload(DiameterMessage response,
     DiameterMessage request,Session session)
                  throws CxExperimentalResultException {

   byte[] rere = new byte[1];
   byte[] authtype = new byte[1];
   String reres; //request 1; response 2
   String authtypes; //authorization type
   byte [] identifiera= new byte[1];
   String identifiers;
   //Longitud EAP-Payload
   short longitud;
   byte[] long_byte;
   String longituds;
   //Longitud nonce
   short long_nonce;
   byte[] long_nonce_byte;
   byte[] long_nonce_onebyte =new byte[1];
   String long_nonce_str;
   String nonces;
   String contentString;
   byte[] nonce;
```

En estas primeras lineas de código declaramos las variables que serán necesarias para crear el EAP-Payload:
Un byte para saber si será reponse o request (rere), un byte que nos dirá el tipo de autenticación y autorización requerido (authType), un byte como identificador de la sesión (identifiera), un short que como longitud del Payload(longitud) y por último un array de bytes (nonce) para almacenar el nonce. El resto de variables son auxiliares.

```
   boolean iguales = false;
```

Esta es la variable que nos dirá si el nonce que esperamos es el mismo que el que recibimos del Peer.

```
AVP avp = new AVP(DiameterConstants.AVPCode.EAP_PAYLOAD, true,0);
byte identifier = UtilAVP.getEapPayloadField(request,1);
String datos = UtilAVP.getEapPayloadData(request,5);
byte code = UtilAVP.getEapPayloadField(request,4);
```

Creamos el AVP que enviaremos y copiamos en él los campos que cogemos directamente del AVP que recibimos del Peer.

```
if (code == 1){
    IMPI impi = IMPI_DAO.get_by_Identity(session, datos);
    if (impi == null) {
        throw new CxExperimentalResultException(DiameterConstants.
                    ResultCode.RC_IMS_DIAMETER_ERROR_USER_UNKNOWN);
     }
     byte[] secretKey = impi.getK();
     String Realm = impi.getIdentity().substring(impi.
                     getIdentity().indexOf("@")+1);
```

Comprobamos el código del Mensaje que hemos recibido, un "1"significa que es del tipo "identity", es decir, el Peer nos está diciendo cual es su identidad, en este caso buscamos en la base de datos si dicho cliente existe y está registrado, en caso contrario arrojamos una excepción e interrumpimos la comunicación, en caso de que sí esté registrado, buscamos cual es su clave y a partir de su identidad, que es de la forma -user@dominio.xx- extraemos el dominio de origen del mismo.

```
nonce = new byte[16];
SecureRandom randomAccess;
try {
    randomAccess = SecureRandom.getInstance("SHA1PRNG");
}
catch (NoSuchAlgorithmException e) {
     e.printStackTrace();
     return false;
}

randomAccess.setSeed(System.currentTimeMillis());
randomAccess.nextBytes(nonce);
```

Creamos el nonce que enviaremos al cliente, lo creamos de manera aleatoria, para que los posibles atacantes no tengan ninguna información.

```
String u_name= UtilAVP.getUserName(request);
byte[] ha8 = MD5Util.calculateResponse(identifiera,
                                secretKey, nonce );
String ha8_str = HexCodec.encode(ha8);
try{
    FileOutputStream fichero = new FileOutputStream
             ("/opt/OpenIMSCore/FHoSS/src/de
                 /fhg/fokus/hss/nonces.bin");
```

```
            BufferedOutputStream pw =new BufferedOutputStream(fichero);
            pw.write(ha8);
            pw.close();
        }catch (IOException ioe){
            System.out.println("Error creando el fichero");
        }
```

En esta parte del código calculamos, a partir del nonce aleatorio la respuesta que esperamos recibir del cliente, para ello procesamos en una función hash MD5 el nombre del usuario (sin el dominio), su clave y dicho número aleatorio. Codificamos dicho número en hexadecimal y por último lo almacenamos en un fichero de texto. Como dato a destacar, el servidor no tenía mecanismos para guardar dicho nonce por si mismo, ya que los cliente habituales que entran por otros interfaces reenvían el nonce que han recibido. En nuestro caso, ya que el estándar no especifica que esto deba hacerse decidimos guardarlo en un fichero tipo binario.

```
        rere[0]=1;
        authtype[0]=4;
        reres=HexCodec.encode(rere); //request 1; response 2
        authtypes=HexCodec.encode(authtype); //authorization type
        identifiera[0]=(byte)(identifier+ (byte)(1));
        identifiers=HexCodec.encode(identifiera);
        longitud= (short)(6+nonce.length);
        long_byte = HexCodec.shortToByteArray(longitud);
        longituds=HexCodec.encode(long_byte);
        //Longitud nonce
        long_nonce = (short)(nonce.length);
        long_nonce_byte = HexCodec.shortToByteArray(long_nonce);
        long_nonce_onebyte[0]= long_nonce_byte[1];
        long_nonce_str=HexCodec.encode(long_nonce_onebyte);
        nonces = HexCodec.encode(nonce);
        contentString=reres+identifiers+longituds+
                    authtypes+long_nonce_str+nonces;
        byte[] content=HexCodec.decode(contentString);
        avp.setData(content);
    }
```

Por último construimos el mensaje que queremos enviar, cabe destacar de nuevo que el api de java no contiende métodos eficientes para el tratamiento de arrays de bytes, así que trabajamos con Strings, para las cuales java si proporciona un API muy eficiente.

```
        else if (code == 4){
             byte[] n_response2 = UtilAVP.getEapPayloadNonce(request);
             String n_response2_str = HexCodec.encode(n_response2);
             //Getting server nonce to compare...
             byte[] nonceServer=new byte[16]; //nonce sended
             try{
                 BufferedInputStream rd = new BufferedInputStream(new
```

```
            FileInputStream(new File(
              "/opt/OpenIMSCore/FHoSS/src/
                  de/fhg/fokus/hss/nonces.bin")));
        lrd.read(nonceServer);
        rd.close();
    }catch (IOException ioe){
        System.out.println("Error abriendo el fichero");
    }
    String nonce_mio = HexCodec.encode(nonceServer);
```

En esta parte del código, si el código del mensaje que nos llega del cliente es un "4", significa que está queriendo autenticarse por medio de MD5, y que nos está enviando un nonce para llevar a cabo dicha autenticación. Extraemos el resultado del hash MD5 que almacenamos anteriormente en el fichero.

```
String Realm=UtilAVP.getDestinationRealm(request);
String uname= UtilAVP.getUserName(request);
String name=uname+"@"+Realm;
String uri = Realm;
IMPI impi = IMPI_DAO.get_by_Identity(session, name);
byte[] secretKey = impi.getK();
byte[] ha1 = MD5Util.calculateResponse(uname.getBytes(),
                            secretKey,nonceServer );
byte[] codea ={code};
iguales=true;
for (int i=0;i<16;i++){
    if(nonceServer[i] != n_response2[i]){
            iguales=false;
    }
 }
 if(iguales){
     rere[0]=3; //success
 }else{
     rere[0]=4; //Fail
 }
```

En esta parte del código extraemos los datos del cliente que debemos reenviarle en el mensaje y calculamos si el hash que él nos envía es el mismo que nosotros esperamos recibir por su parte. Si se cumple esta condición le enviamos un "success", y en caso contrario le enviamos un FAIL.

```
reres=HexCodec.encode(rere);
identifiera[0]=identifier;
identifiers=HexCodec.encode(identifiera);
longitud=4;
long_byte = HexCodec.shortToByteArray(longitud);
longituds=HexCodec.encode(long_byte);
contentString=reres+identifiers+longituds;
byte[] content=HexCodec.decode(contentString);
avp.setData(content);
```

```
        }
          //payload=

           response.addAVP(avp);
           return iguales;
    }
```

Por último componemos el AVP y lo añadimos a la trama de respuesta que enviaremos al cliente.

Apéndice E

Ficheros de configuración

En este anexo se trata como están construidos los ficheros de configuración y su estructura dentro del servidor de AAA (aaad), el NAS (nasd) y el cliente PANA.

E.1. Ficheros de configuración del servidor AAA (AAAD)

El aaad usa los siguientes ficheros de configuración: aaad.xml, aaad_Diameter_dictionary.xml, aaad_Diameter_server.xml y aaad_user_db.xml.

E.1.1. aaad.xml

Es el fichero general del servidor AAA.
El fichero ya viene comentado en la distribución de OpenDiameter, así que se omitirá el comentarlo en este documento.

E.1.2. aaad_Diameter_dictionary.xml

Contiene el diccionario de Diameter y Diameter-EAP del servidor AAA.

El fichero está dividido en las siguientes secciones: Vendors: tiene la siguiente estructura:

```
<vendor id="61" name="Merit Networks"/>
    <command name="PROXYABLE-ERROR" code="0" pbit="1">
```

Aquí se define el nombre del comando, el código y el valor del bit p. Se incluyen comandos tanto de Diameter Base Protocol como de Diameter-EAP.

```
    <answerrules>
        <fixed>
          <avprule name="Session-Id" maximum="1"/>
        </fixed>
        <required>
          <avprule name="Origin-Host" maximum="1" minimum="1"/>
          <avprule name="Origin-Realm" maximum="1" minimum="1"/>
          <avprule name="Result-Code" maximum="1" minimum="1"/>
        </required>
        <optional>
          <avprule name="Origin-State-Id" maximum="1" minimum="1"/>
          <avprule name="Error-Reporting-Host" maximum="1"/>
          <avprule name="Redirect-Host"/>
          <avprule name="Redirect-Host-Usage" maximum="1"/>
          <avprule name="Proxy-Info" maximum="1"/>
          <avprule name="AVP"/>
</optional>
```

Se definen los AVPs que llevará el comando, tanto los obligatorios como los opcionales.

```
      </answerrules>
    </command>
```

Tipos definidos. Deben incluirse todos los obligados por el Diameter Base Protocol y además los que el usuario quiera crear. Su estructura es la siguiente:

```
    <typedefn type-name="QOSFilterRule" type-parent="OctetString"/>
```

El campo type-parent es opcional.

```
   <typedefn type-name="Integer32"/>
```

AVPs. Deben incluirse todos los definidos por los estándares. Los campos enum son opcionales. El campo -mandatory- debe ser coherente en emisor y receptor para que no de errores de análisis.

```
    <avp name="Auth-Type" code="274" mandatory="must"
 protected="mustnot" may-encrypt="no">
      <type type-name="Unsigned32"/>
      <enum name="Authenticate Only" code="1"/>
      <enum name="Authorize Only" code="2"/>
      <enum name="Authorize Authenticate" code="3"/>
    </avp>
```

Aplicaciones. En este apartado se definen las aplicaciones soportadas. Como ejemplo tomamos Diameter-EAP:

```
<application id="5" name="EAP"
    uri="ftp://ftp.ietf.org/internet-drafts/draft-ietf-aaa-eap-03.txt">
   <command name="Diameter-EAP" code="268" pbit="1">
```

Hasta aquí se define la aplicación y el comando asignado, de acuerdo a los RFCs

```
    <requestrules>
        <fixed>
  <avprule name="Session-Id" maximum="1" minimum="1"/>
</fixed>
       <required>
  <avprule name="Auth-Application-Id" maximum="1" minimum="1"/>
  <avprule name="Origin-Host" maximum="1" minimum="1"/>
  <avprule name="Origin-Realm" maximum="1" minimum="1"/>
  <avprule name="Destination-Realm" maximum="1" minimum="1"/>
  <avprule name="Auth-Request-Type" maximum="1" minimum="1"/>
  <avprule name="EAP-Payload" maximum="1" minimum="1"/>
</required>
        <optional>
  <avprule name="NAS-Port" maximum="1"/>
  <avprule name="NAS-Port-Id" maximum="1" />
.
.
.
        </optional>
      </requestrules>
```

Hasta aquí se definen los AVPs que son obligatorios en el mensaje de tipo Request. De nuevo, los marcados como required son obligatorios y los marcados como optional no lo son.

```
      <answerrules>
        <fixed>
  <avprule name="Session-Id" maximum="1" minimum="1"/>
</fixed>
        <required>
  <avprule name="Auth-Application-Id" maximum="1" minimum="1"/>
  <avprule name="Auth-Request-Type" maximum="1" minimum="1"/>
  <avprule name="Result-Code" maximum="1" minimum="1"/>
  <avprule name="Origin-Host" maximum="1" minimum="1"/>
  <avprule name="Origin-Realm" maximum="1" minimum="1"/>
</required>
        <optional>
  <avprule name="User-Name" maximum="1" />
 .
.
.
        </optional>
```

Hasta aquí los AVPs en un mensaje del tipo Answer.De nuevo, los marcados como required son obligatorios y los marcados como optional no lo son.

```
      </answerrules>
```

```
    </command>
<avp name="EAP-Payload" code="462" mandatory="must"
 may-encrypt="yes">
      <type type-name="OctetString"/>
 </avp>
```

Desde aquí se definen los AVPs específicos de la aplicación, como, por ejemplo y en este caso, EAP-Payload.

E.1.3. aaad_Diameter_server.xml

Este fichero de configura los parámetros de Diameter-EAP del aaad. Es el más importante de los ficheros y por ello el análisis será más exhaustivo:

```
<?xml version="1.0" encoding="UTF-8"?>

<configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xsi:noNamespaceSchemaLocation='configuration.xsd'>
   <general>
      <product>Open Diameter</product>
```

Especificamos el nombre del producto que usamos, está información viajará en los mensajes de tipo CER y CEA para establecer el intercambio de capacidades

```
      <version>1</version>
```

Versión de Diameter, debe ser la "1"

```
      <vendor_id>0</vendor_id>
Vendor-ID de nuestro Peer

      <supported_vendor_id>0</supported_vendor_id>
      <supported_vendor_id>1</supported_vendor_id>
```

Lista de Vendor Ids soportados, este parámetros se usa para enrutar mensajes que lleven un campo "Vendor Specific Id"

```
      <auth_application_id>1</auth_application_id>
      <auth_application_id>2</auth_application_id>
      <auth_application_id>5</auth_application_id>
```

Lista de las aplicaciones soportadas, en este ejemplo se soportan MIP4[18], NASREQ y Diameter-EAP

```
      <acct_application_id>3</acct_application_id>
      <acct_application_id>4</acct_application_id>
```

Lista de aplicaciones de actuación, no son demasiado relevantes para el funcionamiento de los programas descritos en este documento

```
<vendor_specific_application_id>
```

Lista de las aplicaciones de Vendors específicos aceptadas y soportadas

```
<vendor_id>31</vendor_id>
<vendor_id>32</vendor_id>
<vendor_id>33</vendor_id>
```

Lista de los Vendors soportados

```
<auth_application_id>1</auth_application_id>
<acct_application_id>4</acct_application_id>
```

Instancias de las aplicaciones específicas de ese Vendor que se soportan

```
</general>
```

Parte general del documento

```
<parser>
   <dictionary>/config/aaad_Diameter_dictionary.xml</dictionary>
</parser>
```

Parte de parser del documento, se especifica la ruta relativa o completa del fichero de diccionario

```
<transport_mngt>

   <identity>localaaa.localdomain1.net</identity>
```

Identidad del Peer, debe ser un nombre que la máquina sea capaz de resolver (incluido en /etc/hosts)

```
<Realm>localdomain1.net</Realm>
```

El Realm al que pertenece el Peer, como ya hemos visto, el enrutamiento en Diameter se basa en Realms

```
<tcp_listen_port>3868</tcp_listen_port>
<sctp_listen_port>1912</sctp_listen_port>
```

Puertos donde el servidor escucha TCP y SCTP respectivamente.

```
<Peer_table>
    <expiration_time>1</expiration_time>
```

Tiempo de expiración para Peers descubiertos dinámicamente

```
<Peer>
    <hostname>localnas.localdomain2.net</hostname>
    <port>1812</port>
    <tls_enabled>0</tls_enabled>
```

Nombre del Peer, puerto y flag de si se usará tls o no con dicho Peer

```
    </Peer>
</Peer_table>
```

La tabla de Peers contiene la información de las entidades a las que este Peer intentará conectarse al iniciarse, es decir intentará llevar a cabo un intercambio de capacidades con ellas.

```
<route_table>
    <expire_time>0</expire_time>
```

Tiempo de expiración para rutas aprendidas dinámicamente, un 0 indica que no hay expiración

```
    <route>
```

Definición de las rutas, el enrutado se hará en base al AVP "Destination-Realm".

```
        <Realm>localdomain2.net</Realm>
```

Realm, entrada usada para el enrutamiento

```
        <role>0</role>
```

La función que este Peer realiza siendo:

- 0: Servidor Local
- 1: Servidor Relay
- 2: Servidor Proxy
- 3: Servidor Redirect

```
        <application>
```

La aplicación se usa como segunda entra de la tabla de enrutamiento, los mensajes que entren en este apartado serán de nuevo multiplexados según el application_id, puede haber varias para una misma entrada de Realm.

```
            <application_id>1</application_id>
<application_id>5</application_id>

            <vendor_id>0</vendor_id>
```

Vendor Id de la aplicación, según el RFC de Diameter Base Protocol,debe ser 0 para aplicaciones estándar.

```
            <Peer_entry>
```

Servidor encargado de este Realm y esta aplicación.

```
                    <server>localnas.localdomain2.net</server>

                    <metric>1</metric>
```

El valor de preferencia de este servidor.

```
                </Peer_entry>
          <default_route>
```

La ruta por defecto se define como cualquier otra ruta y es por donde se encaminarán los paquetes para los cuales no hay ruta específica. Puede apuntar, por ejemplo, a un Redirect o a un proxy.

```
            <Realm>research.telcordia.com</Realm>
            <role>full</role>
            <application>
                 <application_id>
                      <id>1</id>
                      <type>1</type>
                 </application_id>
                 <Peer_entry>
                     <server>server1.research.org</server>
                     <metric>1</metric>
                 </Peer_entry>
                 <Peer_entry>
                     <server>server2.research.org</server>
                     <metric>5</metric>
                 </Peer_entry>
            </application>
        </default_route>
    </route_table>
</transport_mngt>

<session_mngt>
```

Sección de parámetros de la sesión.

```
    <max_sessions>10000</max_sessions>
```

Número máximo de sesiones que pueden ser soportadas a la vez. El hardware y el sistema operativo serán quienes fijen este parámetro realmente.

```
    <stateful>1</stateful>
```

Este parámetro indica si los clientes accederán a sesiones de tipo -full access- o no.

```
    <timers>
```

Temporizadores de la sesión

```
        <session>30</session>
```

Tiempo de la sesión antes de necesitar reautenticaciones

```
<lifetime>360</lifetime>
```

Tiempo máximo que durará la sesión independientemente de las reautenticaciones.

```
<grace>30</grace>
```

Periodo de gracia que recibirán los Peers para poder reautenticarse.

```
<abort_retry>20</abort_retry>
```

Temporizador para dar por fallidos los intentos de retransmisión.

```
    </timers>
  </session_mngt>

  <log>
```

Logs que pueden habilitarse o deshabilitarse.

```
    <flags>
```

Niveles de log

```
      <debug>disabled</debug>
      <trace>disabled</trace>
      <info>disabled</info>
    </flags>
    <target>
```

Salidas de log

```
      <console>enabled</console>
      <syslog>disabled</syslog>
    </target>
  </log>
</configuration>
```

E.1.4. aaad:user_db.xml

Este fichero está comentado en el propio código por lo que no se considera necesario explicarlo aquí.

E.2. Ficheros de configuración del NAS (nasd)

El nasd, al tener una interfaz PANA y otra Diameter-EAP tiene un mayor número de ficheros de configuración, sin embargo muchos de ellos tienen estructura común con los comentados para el aaad.

E.2.1. nasd.xml

Este fichero se encuentra comentado sobre el propio código y no es necesario comentarlo aquí.

E.2.2. nasd_Diameter_eap_dictionary.xml

El fichero de diccionario Diameter-EAP del nas, tiene la misma estructura que el fichero aaad_Diameter_dictionary.xml ya comentado en este anexo.

E.2.3. nasd_Diameter_eap.xml

Fichero de configuración del interfaz Diameter-EAP del nas. Tiene la misma estructura que el fichero aaad_Diameter_server.xml comentado en este mismo anexo.

E.2.4. nasd_pana_dictionary.xml

Fichero de diccionario de PANA del nasd. Su estructura es la siguiente:

```
<dictionary>
```

Linea de comienzo del diccionario

```
    <command name="PANA-Start" code="2">
```

Definición del nombre y número del comando.

```
       <requestrules>
```

Reglas para los mensajes de tipo Request.

```
         <optional>
```

AVPs opcionales

```
                <avprule name="EAP-Payload" maximum="1" minimum="0" />
                <avprule name="Algorithm" maximum="1" minimum="0" />
         </optional>
        </requestrules>
        <answerrules>
```

Reglas para los mensajes de tipo respuesta

```
          <optional>

                      <avprule name="EAP-Payload" maximum="1" minimum="0" />
         </optional>
        </answerrules>
    </command>
```

Posteriormente aparecen una serie de tipos definidos:

```
<typedefn type-name="UTF8String" type-parent="OctetString"/>
```

Y por último la definición de los AVPs:

```
<avp name="AVP" code="0"> <!-- Added by Ohba -->
    <type type-name="Any"/>
  </avp>
```

Como vemos la estructura es muy similar a la de los diccionarios de Diameter-EAP.

E.2.5. nasd_pana_paa.xml

Fichero de configuración del interfaz PANA del nasd. Su estructura está explicada en el propio fichero, por lo que no es necesario explicarla aquí.

En cuanto a los ficheros de configuración del cliente PANA, su estructura está explicada en los propios ficheros, y, por este motivo, no se incluye en el presente documento.

Apéndice F

Estructura de directorios

En este anexo del trabajo se trata de como hay que organizar los directorios y donde hay que colocar cada fichero para poder reproducir el trabajo realizado.

F.1. Estructura básica de OpenDiameter

Una vez instaladas las bibliotecas de OpenDiameter tendremos la siguiente estructura de directorios:

```
                        |- applications
                        |- docs
                        |- home
                        |- include
                        |- libdiameter
OpenDiameter-1.0.7-i    |- libdiametereap
                        |- libdiametermip4
                        |- libdiameternasreq
                        |- libdiamparser
                        |- libeap
                        |- liboudtl
                        |- libpana
                        |- tools
                        |- libpana2*
```

*Dentro de esta estructura de directorios se añadirá el directorio libpana2 (incluido en el código adjunto al PFC) y que no se incluye en la distribución de OpenDiameter-1.0.7-i. Este directorio contiene los clientes que se han programado y se creó para posibles modificaciones en las bibliotecas sin alterar la estructura original de OpenDiameter.

F.2. Análisis de los directorios utilizados y ficheros añadidos

Ahora haremos un análisis de los directorios utilizados, especificando los ficheros que es necesario añadir:

```
                | aaa
applications    | nas
                | pana
```

Y dentro de cada uno de ellos:

- AAA

```
                |              |- aaad.xml
                |  config      |- aaad_diameter_dictionary.xml
                |              |- aaad_diameter_server.xml
                |              |- aaad_user_db.xml
                |
                |
                |
      aaa       |  include     |- ficheros .h
                |
                |
                |              |  aaad_diameter_eap.cxx
                |  src         |  aaad_main.cxx
                |
                |
                | ./aaad (Ejecutable del servidor AAA sin TLS)
```

- NAS

```
                |              |- nasd.xml
                |              |- nasd_diameter_eap_dictionary.xml
                |   config     |- nasd_diameter_eap.xml
                |              |- nasd_pana_dictionary.xml
                |              |- nasd_pana_paa.xml
                |              |- nasdtls.xml
                |              |- nasd_diameter_eap_tls.xml
                |
                |
                |
      nas       |  include     |- ficheros .h
                |
                |
                |              |  nasd_diameter_eap.cxx
                |   src        |  nasd_main.cxx
                |              |  nasd_pana.cxx
```

```
|
|   scripts*
|
|   configpruebas*
|
| ./nasd (Ejecutable del NAS, tanto con como sin TLS)
```

*scripts es un directorio incluido en OpenDiameter, dentro hay un sólo script sin ningún código, sólo comentarios.
configpruebas es un directorio creado para incluir la configuración del NAS en las distintas pruebas del plan de pruebas.

- Cliente PANA (NO funciona)

```
                | pacd.cxx
                | pacd_config.h
                | pacd_config.cxx
                | client_pana_eap_tls.cxx
                |
                |
                |          |  *tls  | *auth
                |  config  |        | *peer
    pana        |          | - *client.eap-tls.xml
                |          | - *eap-tls.configuration.xml
                |          | - pana_dictionary.xml
                |          | - pana_pac.xml
                |          | - pana_setup.xml
                |          | - pana_test_auth_script
                |
                | ./pacd (Ejecutable del cliente PANA
                |              sin TLS (no funciona))
                | ./client_pana_eap_tls (Ejecutable
                |   del cliente PANA con TLS (no funciona))
```

Y dentro de los directorios -auth- y -peer- tenemos los ficheros necesarios para hacer funcionar tls.
Los directorios tls, auth y peer no se incluyen en la distribución de OpenDiameter y deben ser descargados desde la página www.opendiameter.org o desde el svn.
Del mismo modo, los ficheros client.eap-tls.xml y eap-tls.configuration.xm deben ser descargados o bien cogidos del directorio /libdiametereap.
Los .xml deben ir acompañados de sus correspondientes .dtd

```
- docs, home, include, libdiameter
```

Aquí no es necesario realizar ningún cambio. Los ficheros instalados por OpenDiameter son correctos.

```
                  |            | - client.eap-tls.xml
                  |            | - client.local.xml
                  |            | - configuration.xml
                  |   config   | - dictionary.xml
                  |            | - server.eap-tls.xml
                  |            | - server.local.xml
                  |            | - eap-tls.configuration.xsd
  libdiametereap  |
                  |
                  |   include  | - Ficheros.h
                  |
                  |   src      | - Ficheros .cxx
                  |
                  |   test     | *tls_config  | *auth
                  |                           | *Peer
                  |
                  |   ./server_test_tls (Servidor AAA con TLS)
```

* Los directorios tls_config, auth y Peer junto con el fichero eap-tls.configuration.xsd deben ser descargados de www.OpenDiameter.org o mediante el svn. Los xml deben ir acompañados de sus correspondientes .dtd

`libDiametermip4, libDiameternasreq y libdiamparser`

Sin cambios

```
                  | libeaparchie
                  | libeapcore
                  | libeap-tls
                  | include      | *xml.h
     libeap       |              | *xml_errorreporter.h
                  |
                  | test
                  | config
```

Los ficheros marcados con * tuvieron que ser descargados de internet para poder compilar las bibliotecas con TLS, son ficheros que parecen a una distribución anterior de OpenDiameter y cuya fecha data de 2004 (en lugar de 2007 como el resto de los ficheros).

`libodutl, libpana`

Sin cambios.

```
                 |            | - client.eap-tls.xml
                 |            | - dictionary.xml
                 |  config    | - pac.xml
                 |            | - users.xml
```

```
                  |            | - eap-tls.configuration.xsd
                  |
                  |  docs
                  |  home
  libpana2        |  include   | - Ficheros .h y .hxx
                  |
                  | *scripts   | - *pruebalocal.sh
                  |            | - *pruebaremota.sh
                  |            | - *pruebatls.sh
                  |
                  |  src       | - Ficheros.cxx
                  |
                  | test       |- *cliente.cxx
                  |            |(Código del cliente PANA sin TLS)
                  |            |- *cliente2.cxx
                  |            |(Código del cliente PANA con TLS)
                  |
                  | ./clientepana
                  |  (Ejecutable del cliente PANA sin TLS)
                  | ./clientepanatls
                  |  (Ejecutable del cliente PANA con TLS)
                  |
```

El directorio scripts se creó enteramente para el desarrollo del PFC, en el directorio test deben añadirse los ficheros cliente.cxx, y cliente2.cxx que son los clientes desarrollados. El primero es el cliente MD5 y el segundo es el cliente TLS.

Por último, para que el aaad y el nasd funcionen correctamente en local debemos tener los ficheros de configuración en local en los siguientes directorios:

```
aaad --> /etc/OpenDiameter/aaa/config/ aaad.xml
                                       aaad_Diameter_dictionary.xml
                                       aaad_Diameter_server.xml
                                       aaad_user_db.xml
```

y los correspondientes .dtd

```
nasd ---> /etc/OpenDiameter/nas/config/  nasd.xml
                                         nasd_Diameter_dictionary.xml
                                         nasd_Diameter_eap.xml
                                         nasd_pana_dictionary.xml
                                         nasd_pana_paa.xml
```

Apéndice G

Pruebas

En esta sección se explicará el proceso de pruebas llevado a cabo para cada uno de los apartados del proyecto (MD5 en local, MD5 en remoto y TLS en local).

La estructura general de una prueba es:

1. Nombre de la prueba.
2. Categoría de la prueba.
3. Descripción de la prueba.
4. Resultados esperados.
5. Resultados obtenidos.
6. Trazas.
7. Realización.

La manera de ejecutar las pruebas, a menos que en la propia prueba se especifique otra cosa, es:

1. Colocarse en el directorio /opendiameter-1.0.7-i/libpana2/
2. Teclear: ./scripts/pruebalocal.sh número de prueba (si se trata de una prueba en local)
3. Teclear: ./scripts/pruebaremota.sh número de prueba (si se trata de una prueba remota)
4. Teclear: ./scripts/pruebatls.sh número de prueba (si se trata de una prueba de TLS)

Cabe destacar que el orden de las pruebas para el script no es el mismo que el orden en el que están aquí descritas, así que el número de prueba para el script se especificará en cada caso.

G.1. Pruebas del Escenario con MD5 en local

G.1.1. Pruebas del NAS y el servidor AAA sin cliente

Prueba 0.1 : NAS y servidor AAA sin parámetros

- Categoría: Pruebas de configuración.
- Descripción: En esta prueba arrancaremos el NAS y el servidor AAA sin pasarles ningún parámetros por la linea de comandos.
- Resultado esperado: Ambos deben ser capaces de encontrar e interpretar el fichero de configuración que tienen por defecto incluido en el código (en caso del que tanto el fichero como la ruta sean correctos)
- Resultado obtenido: Las dos aplicaciones arrancan usando los ficheros de configuración por defecto.
- Resultado de la prueba: Pasada.
- Trazas: No se precisan.
- Realización : No es necesario ejecutar ningún script, sino simplemente ejecutar el NAS y el AAA.

Prueba 0.2 : NAS y servidor AAA con parámetros

- Categoría: Pruebas de configuración.
- Descripción: En esta prueba arrancaremos el NAS y el servidor AAA pasándoles el fichero de configuración por la línea de comandos, e incluyendo otra serie de parámetros detrás de este.
- Resultado esperado: Ambos deben ser capaces de encontrar e interpretar el fichero de configuración que se les pasa por la línea de comandos (si la ruta y el fichero son correctos) y deben ignorar el resto de parámetros.
- Resultado obtenido: Las dos aplicaciones arrancan usando los ficheros de configuración pasados por la línea de comandos.

- Resultado de la prueba: Pasada.
- Trazas: No se precisan.
- Realización : No es necesario ejecutar ningún script, sino simplemente ejecutar el NAS y el AAA.

Prueba 1 : Arranque de NAS y servidor AAA en distinto orden

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en arrancar el NAS y el servidor AAA en distinto orden.
- Resultado esperado: Independientemente del orden, ambos extremos deben realizar el intercambio de capacidades y enviarse mutuamente los mensajes de tipo Wacthdog.
- Resultado obtenido:La prueba se efectuó 50 veces. De estas 50 veces se observó que, si el orden de arranque era primero el NAS y después el AAA, la comunicación no fue correcta por falta de mensajes Watchdog en alguno de los dos extremos en 34 ocasiones. Si el orden de arranque era el contrario (primero el NAS y posteriormente el servidor AAA) la comunicación falló en 16 ocasiones.
  Con esta simple estimación puede apreciarse que la tasa de error en el primer caso es de 68 % y en el segundo caso de 32 %.
  Las estimaciones no son del todo correctas ni rigurosas, pero dan una idea de que debe arrancarse primero el NAS y de que las aplicaciones no funcionan del todo correctamente.
  En ningún apartado de la descripción de OpenDiameter se habla de este problema, y en algunas máquinas no es reproducible.
- Resultado de la prueba: No pasada.
- Trazas: No se precisan.
- Realización : No es necesario ejecutar ningún script, sino simplemente ejecutar el NAS y el AAA.

Prueba 2 : Incongruencias en los ficheros de diccionario del NAS y del servidor AAA

- Categoría: Pruebas de configuración.

- Descripción: Esta prueba consiste en tener ficheros de diccionario de Diameter-EAP distintos en el NAS y el servidor AAA, por ejemplo, tener en uno de ellos un flag "mandatory"que no aparece en el otro (debe modificarse algún AVP que sepamos que será usado).
- Resultado esperado: Deben producirse errores de análisis al interpretar los mensajes enviados de una entidad a la otra.
- Resultado obtenido: Para realizar esta prueba, elegimos el AVP username, y en el servidor AAA lo marcamos como -mandatory-, mientras que en el NAS no lo marcamos de esta manera. El error de análisis apareció como traza en pantalla.
- Resultado de la prueba: Pasada.
- Trazas:

```
AAA: Parser Error. "M" flag must be set to 1 in User-Name AVP.
```

- Realización : No es necesario ejecutar ningún script, sino simplemente ejecutar el NAS y el AAA habiendo modificado algún fichero de diccionario.

Prueba 3 : Conexión de varios NAS con el mismo servidor AAA

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en conectar varios NAS al mismo servidor AAA. Hay que tener en cuenta que si se realiza en local habrá que asignar puertos distintos a cada uno de ellos y modificar el fichero de configuración del AAA para que todos ellos figuren como Peers.
- Resultado esperado: El AAA debe intercambiar capacidades con todos los NAS y recibir y enviar los Watchdogs de y a cada uno de ellos.
- Resultado obtenido: El comportamiento experimental es el descrito en el apartado Resultado esperado.
- Resultado de la prueba: Pasada.
- Trazas: No se precisan.
- Realización : No es necesario ejecutar ningún script, sino simplemente ejecutar el NAS y el AAA.

Prueba 4 : Conexión del NAS a varios servidores AAA

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en conectar el NAS a varios servidores AAA al mismo tiempo. De nuevo habrá que asignar puertos distintos para hacerlo en local y modificar el fichero de configuración, en este caso, del NAS
- Resultado esperado: El NAS debe intercambiar capacidades con todos los servidores AAA y enviar y recibir los watchdogs correctamente.
- Resultado obtenido: El comportamiento experimental es el descrito en el apartado Resultado esperado.
- Resultado de la prueba: Pasada.
- Trazas: No se precisan.
- Realización : No es necesario ejecutar ningún script, sino simplemente ejecutar el NAS y el AAA.

Prueba 5: Intento de comunicación con ninguna aplicación común

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en intentar comunicar un NAS y un servidor AAA con ninguna aplicación en común (dentro de los ficheros de configuración, la parte marcada como auth_application_id).
- Resultado esperado: La comunicación debe acabar con un mensaje del tipo DIAMETER_NO_COMMON_APPLICATION.
- Resultado obtenido: El Resultado obtenido es el especificado en el resultado esperado.
- Resultado de la prueba: Pasada.
- Trazas:

```
(18796|2929261456) Acceptor Peer failed establishing state [5010], closing
```

- Realización :
  1. Lanzar el servidor AAA de manera normal.
  2. Para lanzar el NAS hacer: ./scripts/pruebalocal.sh 10 desde el directorio /libpana2/

Prueba 6: Intento de comunicación sin métodos de seguridad comunes

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en intentar comunicar un NAS y un servidor AAA sin métodos de seguridad en común, en nuestro caso, en un fichero de configuración habilitaremos TLS y en el otro no.
- Resultado esperado: La comunicación debe acabar con un mensaje del tipo DIAMETER_NO_COMMON_SECURITY.
- Resultado obtenido: El Resultado obtenido es el especificado en el resultado esperado.
- Resultado de la prueba: Pasada.
- Trazas:

```
(26292|2977926032) Acceptor Peer failed establishing state [3004]
                                                        , closing
```

- Realización :
  1. Lanzar el servidor AAA de manera normal.
  2. Para lanzar el NAS hacer: ./scripts/pruebalocal.sh 11 desde el directorio /libpana2/

Prueba 7: Comprobación de temporizadores

- Categoría: Pruebas de configuración.
- Descripción: Esta prueba consiste en comprobar si los tiempos especificados por los temporizadores se cumplen. Dado que la conexión entre el NAS y el servidor AAA no debe interrumpirse mientras haya mensajes de Watchdog recibiéndose, el único parámetro relevante aquí es el temporizador del Watchdog.
- Resultado esperado: Los mensajes de Watchdog deben intercambiarse con la frecuencia que se marque en el fichero de configuración.
- Resultado obtenido: El Resultado obtenido es el especificado en el resultado esperado.
- Resultado de la prueba: Pasada.

- Trazas: No se aplica
- Realización : No es necesario ejecutar ningún script, sino simplemente ejecutar el NAS y el AAA con las modificaciones de temporización adecuadas.

G.1.2. Pruebas del cliente PANA en local

En esta sección se tratarán las pruebas sobre el cliente PANA programado para el proyecto. Las pruebas siempre deben llevarse a cabo en un escenario en el que las comunicaciones entre el NAS y el servidor AAA estén libres de errores.

Prueba 0: Intento de conexión con un NAS fuera de linea

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en arrancar el cliente PANA sin haber arrancado el NAS.
- Resultado esperado: El cliente debe reenviar sus mensajes hasta que el temporizador venza, y una vez este hecho sucede debe interrumpirse la conexión.
- Resultado obtenido: El comportamiento experimental es el descrito en el apartado Resultado esperado, se marca como causa de desconexión la 8: timeout.
- Resultado de la prueba: Pasada.
- Trazas:

```
(17703|3062950800) Event: 160 occurring
(17703|3062950800) TxPCI: id=0 seq=0
(17703|3062950800) From state: OFFLINE to OFFLINE
(17703|3037772688) Event: 32768 occurring
(17703|3037772688) Re-transmitting last request
(17703|3037772688) From state: OFFLINE to OFFLINE
(17703|3054558096) Event: 32768 occurring
(17703|3054558096) Re-transmitting last request
(17703|3054558096) From state: OFFLINE to OFFLINE
(17703|3037772688) Event: 32768 occurring
(17703|3037772688) Re-transmission giving up
(17703|3037772688) From state: OFFLINE to OFFLINE
```

```
(17703|3037772688) Event: 16384 occurring
(17703|3037772688) Disconnect: cause=8
(17703|3037772688) From state: OFFLINE to WAIT_PEA
```

- Realización : No es necesario ejecutar ningún script, sino simplemente ejecutar el cliente PANA sin haber ejecutado el NAS.

Prueba 1: Conexión de un cliente no perteneciente al dominio (Realm)

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en enviar un cliente PANA al NAS que no pertenezca al dominio del AAA
- Resultado esperado: El NAS debe enviar directamente un mensaje de error al cliente PANA al no poder enrutarlo.
- Resultado obtenido: El comportamiento experimental es ligeramente diferente al Resultado esperado, el NAS no notifica al cliente el error, sino que éste reenvía el mensaje el número de veces marcado en su fichero antes de dar por terminada la conexión.
- Resultado de la prueba: Pasada.
- Trazas: NAS:

```
(18919|3037453200) DestRealm(localxxx.net) in routing
                          table but no matching app Id
(18919|3037453200) Router cannot deliver message,
                            sending back with an error
(18919|3037453200) *** Router rejected request message ***
```

Cliente PANA:

```
(17703|3062950800) Event: 160 occurring
(17703|3062950800) TxPCI: id=0 seq=0
(17703|3062950800) From state: OFFLINE to OFFLINE
(17703|3037772688) Event: 32768 occurring
(17703|3037772688) Re-transmitting last request
(17703|3037772688) From state: OFFLINE to OFFLINE
(17703|3054558096) Event: 32768 occurring
(17703|3054558096) Re-transmitting last request
(17703|3054558096) From state: OFFLINE to OFFLINE
(17703|3037772688) Event: 32768 occurring
(17703|3037772688) Re-transmission giving up
```

```
(17703|3037772688) From state: OFFLINE to OFFLINE
(17703|3037772688) Event: 16384 occurring
(17703|3037772688) Disconnect: cause=8
(17703|3037772688) From state: OFFLINE to WAIT_PEA
```

- Realización :

  1. Lanzar el NAS de manera normal.
  2. Lanzar el servidor AAA de manera normal.
  3. Para lanzar el cliente hacer: ./scripts/pruebalocal.sh 1 desde el directorio /libpana2/

Prueba 2: Nombre de usuario correcto pero método de autenticación distinto en el cliente y el servidor AAA

- Categoría: Pruebas de funcionalidad.

- Descripción: En esta prueba NAS y servidor AAA estarán correctamente conectados. Lanzaremos el cliente PANA con un nombre de usuario que pertenezca al dominio del servidor AAA, es decir, que el NAS será capaz de enrutar hacia el servidor AAA. El cliente tiene registrado un método de autenticación distinto al que tiene en su fichero de clientes el servidor AAA para ese cliente.
  También es válida para cualquier cliente perteneciente al dominio pero no registrado en el fichero, ya que el fichero de clientes del servidor AAA tiene una entrada por defecto para cualquier cliente desconocido.

- Resultado esperado: El NAS debe enviar los mensajes del cliente al servidor AAA, pero este debe devolverle un resultado de error en la autenticación tras el mensaje de Nak del cliente para intentar cambiar de método[1].

- Resultado obtenido: El comportamiento experimental es el descrito en el apartado Resultado esperado. El error indicado es error de autenticación.

- Resultado de la prueba: Pasada.

- Trazas:
  En el lado del cliente PANA:

[1]La política de Naks del servidor AAA es no dar soporte a este tipo de mensajes, para evitar fisuras de seguridad

```
Peer:  Parse Request.
Peer: Parsing Identity request.
Peer: Building Identity response.
Peer:  Sent Response.
checking key availability.
EAP Response sent from Peer
Peer:  Parse Request.
Peer:  New Method.
Peer:  Do Policy Check.
Peer:  Sent Nak.
EAP Response sent from Peer
Peer: Failure received.
Peer:  Failure.
Authentication failure detected at Peer
Sorry, lalala@localdomain1.net try next time !!!
```

En el lado del nasd:

```
Error was received.
Result-Code=4001.
Passthrough: EAP authentication failed.
(18071|3037731728) Disconnect: cause=0
```

Y por último en el lado del aaad:

```
AuthIdentityStateTable: Do Identity Check.
(18082|3046595472) *** Match User [lalala@localdomain1.net]: default
Backend: Trying another legacy method.
AuthArchie: Building a request message.
Backend: Request sent and timer started.
[] EAP Request received from backend.
[] Sending DEA with continue result-code.
Sent DEA Message.
[] Copying DER to DEA.
[] forwarding EAP Response to authenticator.
Backend: Reset method due to receiving Nak.
Backend: Update policy on Nak.
Backend: Composing Failure message.
[] EAP Failure received from backend.
[] Sending DEA due to authentication failure.
```

- Realización :

  1. Lanzar el NAS de manera normal.
  2. Lanzar el servidor AAA de manera normal.
  3. Para lanzar el cliente hacer: ./scripts/pruebalocal.sh 2 desde el directorio /libpana2/

Prueba 3: Password Incorrecta

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en enviar un cliente PANA que sí pertenece al dominio, sí tiene el mismo método de autenticación registrado que el servidor AAA para él, pero que tiene una password incorrecta.
- Resultado esperado: El AAA debe notificar el error de autenticación tras recibir el nonce mal procesado.
- Resultado obtenido: El Resultado obtenido es el mismo que el descrito en Resultado Esperado. El código de error de Diameter-EAP 4001 (Authentication Rejected) es recibido.
- Resultado de la prueba: Pasada.
- Trazas: En el lado del cliente PANA:

```
Peer: Building Identity response.
Peer: Sent response.
chcking key availability.
EAPResponse sent from Peer
Peer:  Parse Request.
Peer:  New Method.
Peer:  Do Policy Check.
Peer:  Creating Method.
Peer:  Do Integrity Check.
PeerMDChallenge: Do Identity Check.
EapPeerMD5Challenge: Identifier = 2.
PeerMD5ChallengeStateTable: Response Prepared.
Peer:  Sent Response.
checking key availability.
EAP Response sent from Peer
Peer: Failure received.
Peer:  Failure.
Authentication failure detected at Peer
Sorry, lalala@localdomain1.net try next time !!!
```

En el lado del nasd:

```
[] AAA_MULTI_ROUND_AUTH received.
[] forwarding EAP Request to passthrough.
Passthrough: Request sent and timer started.
Passthough: Integrity check.
[] EAP-Response received from passthrough.
```

```
[] sending DER.
Sent DER Message.
[] Error was received.
[]   Result-Code=4001.
Passthrough: EAP authentication failed.
```

Por último, en el lado del aaad:

```
(19227|3038305168) New auth session
(19227|3038305168) Negotiated session state: 1
(19227|3038305168) Accepted client auth lifetime hint: 29
[] forwarding EAP Response to authenticator.
Backend: Integrity Check.
AuthIdentityStateTable: Do Identity Check.
Backend: Trying another legacy method.
AuthMD5ChallengeST: Request Prepared.
Backend: Request sent and timer started.
[] EAP Request received from backend.
[] Sending DEA with continue result-code.
[] forwarding EAP Response to authenticator.
Backend: Integrity Check.
AuthMD5ChallengeStateTable: Do Identity Check.
EapAuthMD5Challenge: Identifier = 2.
AuthMD5ChallengeStateTable: Invalid response.
Backend: Composing Failure message.
[] EAP Failure received from backend.
[] Sending DEA due to authentication failure.
```

- Realización :

  1. Lanzar el NAS de manera normal.
  2. Lanzar el servidor AAA de manera normal.
  3. Para lanzar el cliente hacer: ./scripts/pruebalocal.sh 3 desde el directorio /libpana2/

Prueba 4: Todos los parámetros correctos

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en enviar un cliente PANA que sí pertenece al dominio, sí tiene el mismo método de autenticación registrado que el servidor AAA para él y con la password correcta.
- Resultado esperado: El AAA debe notificar el éxito de autenticación.

- Resultado obtenido: El Resultado obtenido es el mismo que el descrito en Resultado Esperado. Este caso se explicó en detalle en otra sección de este documento, por lo que no se incluirá ninguna traza.
- Resultado de la prueba: Pasada.
- Trazas: No son necesarias.
- Realización :
  1. Lanzar el NAS de manera normal.
  2. Lanzar el servidor AAA de manera normal.
  3. Para lanzar el cliente hacer: ./scripts/pruebalocal.sh 4 desde el directorio /libpana2/

Prueba 5: Batería de clientes

- Categoría: Pruebas de carga.
- Descripción: Esta prueba consiste en enviar una batería de clientes PANA al NAS.
- Resultado esperado: El NAS debe ser capaz de enrutar los mensajes de los cliente correctamente, hasta que el número de clientes sea demasiado elevado.
- Resultado obtenido: El Resultado obtenido es que el NAS sólo puede procesar un cliente cada vez, esto se debe a que el NAS escucha PANA en un sólo puerto y su implementación no crea varios hilos según van llegando clientes o aplica algún otro método de multiplexación.
- Resultado de la prueba: No pasada.
- Trazas: No son necesarias.
- Realización :
  1. Lanzar el NAS de manera normal.
  2. Lanzar el servidor AAA de manera normal.
  3. Para lanzar el cliente hacer: ./scripts/pruebalocal.sh 5 desde el directorio /libpana2/

Prueba 6: Servidor AAA fuera de línea

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en enviar el cliente PANA al NAS, perteneciendo el cliente a un dominio que el NAS sí tiene registrado pero que no se encuentra conectado.
- Resultado esperado: El NAS debe enviar un mensaje de error al Cliente PANA para notificarle que no puede entregar sus mensajes.
- Resultado obtenido: El Resultado obtenido es que el NAS no notifica al cliente el error, sino que éste, tras el número de intentos marcados en su fichero de configuración, da la conexión por fallida.
- Resultado de la prueba: Pasada.
- Trazas: consultar prueba número 1 de este mismo apartado.
- Realización :
  1. Lanzar el NAS de manera normal.
  2. Para lanzar el cliente hacer: ./scripts/pruebalocal.sh 6 desde el directorio /libpana2/

Prueba 7: Conexión directa del cliente PANA con el servidor AAA

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en enviar el cliente PANA al servidor AAA sin pasar por el NAS.
- Resultado esperado: El servidor AAA debe descartar silenciosamente los mensajes del cliente y el cliente debe desconectarse tras el número de intentos de conexión marcados en su fichero de configuración.
- Resultado obtenido: El Resultado obtenido es el especificado en el resultado esperado.
- Resultado de la prueba: Pasada.
- Trazas: En el lado del cliente (El único que muestra trazas en este test): Tenemos las mismas trazas que en el caso de la prueba 0.
- Realización :

1. Lanzar el servidor AAA de manera normal.
2. Para lanzar el cliente hacer: ./scripts/pruebalocal.sh 7 desde el directorio /libpana2/

Prueba 8: Llamada al cliente PANA sin parámetros

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en ejecutar el cliente PANA sin ningún parámetro (El cliente exige al menos un parámetros que será el fichero de configuración.
- Resultado esperado: El cliente interrumpirá su ejecución con algún mensaje de error.
- Resultado obtenido: El Resultado obtenido es el especificado en el resultado esperado.
- Resultado de la prueba: Pasada.
- Trazas:

```
Usage: clientepana [-c] -f [configuration file]
```

- Realización :

1. Para lanzar el cliente hacer: ./scripts/pruebalocal.sh 8 desde el directorio /libpana2/

Prueba 9: Llamada al cliente PANA con un fichero de configuración erróneo o inexistente

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en ejecutar el cliente PANA con un fichero de configuración con errores o cuya ruta no es correcta.
- Resultado esperado: El cliente interrumpirá su ejecución con algún mensaje de error.
- Resultado obtenido: El Resultado obtenido es el especificado en el resultado esperado.
- Resultado de la prueba: Pasada.

- Trazas:

```
SAX Parsing exception: Failed to open XML file: ./config/pacla.xml
terminate called after throwing an instance of 'PANA_Exception'
Cancelado
```

- Realización :

  1. Para lanzar el cliente hacer: ./scripts/pruebalocal.sh 9 desde el directorio /libpana2/

Prueba 10: Gestión de temporizadores

- Categoría: Pruebas de configuración.
- Descripción: Esta prueba consiste en probar distintos Auth-Life-Time y Grace-period-Time para el cliente.
- Resultado esperado: Los tiempos deben ser los especificados
- Resultado obtenido: El Resultado obtenido es el especificado en el resultado esperado, al autenticación del cliente es correcta.
- Resultado de la prueba: Pasada.
- Trazas: No se aplica
- Realización :

  1. Lanzar el NAS de manera normal con algún temporizador modificado.
  2. Lanzar el servidor AAA con algún temporizador modificado.
  3. Para lanzar el cliente hacer: ./scripts/pruebalocal.sh 4 desde el directorio /libpana2/

Prueba 11: Distinto MD5-Digest

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en calcular de manera distinta el MD5-Digest en el cliente PANA y en el servidor de AAA.
- Resultado esperado: El servidor debe enviar un mensaje de error de autenticación al final de la comunicación, como último de sus mensajes.

- Resultado obtenido: El Resultado obtenido es el especificado en el resultado esperado. El resultado es equivalente al de la prueba de password incorrecta.
- Resultado de la prueba: Pasada.
- Trazas:
  En el lado del cliente PANA:

```
Peer: Building Identity response.
Peer: Sent response.
chcking key availability.
EAPResponse sent from Peer
Peer:  Parse Request.
Peer:  New Method.
Peer:  Do Policy Check.
Peer:  Creating Method.
Peer:  Do Integrity Check.
PeerMDChallenge: Do Identity Check.
EapPeerMD5Challenge: Identifier = 2.
PeerMD5ChallengeStateTable: Response Prepared.
Peer:  Sent Response.
checking key availability.
EAP Response sent from Peer
Peer: Failure received.
Peer:  Failure.
Authentication failure detected at Peer
Sorry, lalala@localdomain1.net try next time !!!
```

  En el lado del nasd:

```
[] AAA_MULTI_ROUND_AUTH received.
[] forwarding EAP Request to passthrough.
Passthrough: Request sent and timer started.
Passthough: Integrity check.
[] EAP-Response received from passthrough.
[] sending DER.
Sent DER Message.
[] Error was received.
[]   Result-Code=4001.
Passthrough: EAP authentication failed.
```

  Por último, en el lado del aaad:

```
(19227|3038305168) New auth session
(19227|3038305168) Negotiated session state: 1
(19227|3038305168) Accepted client auth lifetime hint: 29
```

```
[] Copying DER to DEA.
[] forwarding EAP Response to authenticator.
Backend: Integrity Check.
AuthIdentityStateTable: Do Identity Check.
Backend: Trying another legacy method.
AuthMD5ChallengeST: Request Prepared.
Backend: Request sent and timer started.
[] EAP Request received from backend.
[] Sending DEA with continue result-code.
Sent DEA Message.
[] Copying DER to DEA.
[] forwarding EAP Response to authenticator.
Backend: Integrity Check.
AuthMD5ChallengeStateTable: Do Identity Check.
EapAuthMD5Challenge: Identifier = 2.
AuthMD5ChallengeStateTable: Invalid response.
Backend: Composing Failure message.
[] EAP Failure received from backend.
[] Sending DEA due to authentication failure.
```

- Realización : Para realizar este test es necesario modificar el código del fichero eap_md5.cxx que es donde se encuentra la manera de realizar el MD5 Digest. Dado que el resultado es equivalente a escribir una password incorrecta no se considera necesario realizar todo este proceso.[2]

G.1.3. Errores de comunicación entre el NAS y el cliente PANA

Estos errores suceden de manera aleatoria independientemente de la prueba que se intente pasar.
Consisten en que, a veces, con la comunicación entre el NAS y el servidor AAA correctamente establecida, al intentar introducir el cliente en la comunicación el NAS interrumpe su funcionamiento.
No se pudo determinar cuando y porqué sucedían estos errores, pero se pudieron capturar tres mensajes de error distintos:

1. Primer error:

```
terminate called without an active exception
PassThrough: Trying a legacy method.
Cancelado
```

[2]En realidad, modificar los parámetros de entrada del MD5-Digest equivale a poner uno de ellos incorrecto, ya que la salida tiene un número determinado y fijo de bytes (es una función de hash).

2. Segundo error:

```
nasd: /usr/include/boost/shared_array.hpp:87: T&
                boost::shared_array<T>::operator[]
                 (ptrdiff_t) const [with T = std::
                  list<int, std::allocator<int> >]:
                        Assertion 'px != 0' failed.
(14574|3045059472) From state: OFFLINE to WAIT_SUCC_PBA
Cancelado
```

3. Tercer error:

```
Virtual pure method called
Fallo de segmentación
```

G.2. Pruebas del escenario con MD5 en remoto

G.2.1. Pruebas del NAS y el servidor HSS sin cliente

Prueba 1 : Arranque de NAS y HSS en distinto orden

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en arrancar el NAS y el HSS en distinto orden.
- Resultado esperado: Independientemente del orden, ambos extremos deben realizar el intercambio de capacidades y enviarse mutuamente los mensajes de tipo Wacthdog.
- Resultado obtenido:La prueba se efectuó 50 veces. De estas 50 veces se observó que, si el orden de arranque era primero el NAS y después el HSS, el intercambio de capacidades no llega a producirse; el HSS es un servidor complejo con muchos interfaces que tarda en arrancar. Si el orden de arranque era el contrario (primero el NAS y posteriormente el HSS) la comunicación falló en sólo 1 ocasión.
  Con esta simple estimación puede apreciarse que la tasa de error en el primer caso es de 100 % y en el segundo caso de 2 %.
  Las estimaciones no son del todo correctas ni rigurosas, pero dan una idea de que debe arrancarse el NAS una vez el HSS esté arrancado
  Queda patente que los problemas que se producían cuando se comunicaban el NAS y el servidor AAA en local no se producen en este caso.

- Resultado de la prueba: Pasada.
- Trazas: No se precisan.
- Realización : No es necesario ejecutar ningún script, sino simplemente ejecutar el NAS con el HSS arrancado.

Prueba 2 : Incongruencias en los ficheros de diccionario del NAS y el HSS

- Categoría: Pruebas de configuración.
- Descripción: Esta prueba consiste en tener ficheros de diccionario de Diameter-EAP distintos en el NAS y HSS, por ejemplo, tener en uno de ellos un flag -mandatory- que no aparece en el otro (debe modificarse algún AVP que sepamos que será usado).
- Resultado esperado: Deben producirse errores de análisis al interpretar los mensajes enviados de una entidad a la otra.
- Resultado obtenido: La prueba no pudo ser realizada, al no conseguir descubrir cómo realiza el análisis el HSS.
- Resultado de la prueba: No aplicable.
- Trazas: No aplicable
- Realización : No aplicable.

Prueba 3 : Conexión de varios NAS con el mismo HSS

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en conectar varios NAS al mismo HSS.
- Resultado esperado: El HSS debe intercambiar capacidades con todos los NAS y recibir y enviar los Watchdogs de y a cada uno de ellos.
- Resultado obtenido: El comportamiento experimental es el descrito en el apartado Resultado esperado.
- Resultado de la prueba: Pasada.
- Trazas: No se precisan.

- Realización : No es necesario ejecutar ningún script, sino simplemente ejecutar varios NAS con distintos puertos apuntando al HSS.

Prueba 4 : Conexión del NAS a varios HSS

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en conectar el NAS a varios HSS al mismo tiempo. De nuevo habrá que asignar puertos distintos para hacerlo en local y modificar el fichero de configuración, en este caso, del NAS
- Resultado esperado: El NAS debe intercambiar capacidades con todos los HSS y enviar y recibir los watchdogs correctamente.
- Resultado obtenido: La prueba no pudo ser realizada, al no conocer más servidores de Diameter-EAP y no poder instalar el HSS en otras máquinas por ser un proceso complicado.
- Resultado de la prueba: No aplicable.
- Trazas: No aplicable.
- Realización : No aplicable.

Prueba 5: Intento de comunicación con ninguna aplicación común

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en intentar comunicar un NAS y un servidor AAA con ninguna aplicación en común (dentro de los ficheros de configuración, la parte marcada como auth_application_id.
- Resultado esperado: La comunicación debe acabar con un mensaje del tipo DIAMETER_NO_COMMON_APPLICATION.
- Resultado obtenido: El resultado obtenido es distinto al esperado. El intercambio de capacidades y el Wacthdog se intercambian a pesar de no haber aplicaciones comunes.
- Resultado de la prueba: No pasada.
- Trazas: No son necesarias
- Realización :

1. Para lanzar el NAS hacer: ./scripts/pruebaremota.sh 7 desde el directorio /libpana2/

Prueba 6: Intento de comunicación sin métodos de seguridad comunes

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en intentar comunicar un NAS y un servidor AAA sin métodos de seguridad en común, en nuestro caso, en un fichero de configuración habilitaremos TLS y en el otro no.
- Resultado esperado: La comunicación debe acabar con un mensaje del tipo DIAMETER_NO_COMMON_SECURITY.
- Resultado obtenido: La prueba no pudo realizarse ya que el HSS acepta conexiones tanto con TLS como sin TLS
- Resultado de la prueba: No aplicable.
- Trazas: No aplicable
- Realización : No aplicable.

Prueba 7: Comprobación de temporizadores

- Categoría: Pruebas de configuración.
- Descripción: Esta prueba consiste en comprobar si los tiempos especificados por los temporizadores se cumplen. Dado que la conexión entre el NAS y el HSS no debe interrumpirse mientras haya mensajes de Watchdog recibiéndose, el único parámetro relevante aquí es el temporizador del Watchdog del NAS.
- Resultado esperado: Los mensajes de Watchdog deben intercambiarse con la frecuencia que se marque en el fichero de configuración.
- Resultado obtenido: El Resultado obtenido es el especificado en el resultado esperado.
- Resultado de la prueba: Pasada.
- Trazas: No son necesarias.
- Realización : No es necesario ejecutar ningún script, sino simplemente ejecutar el NAS con las modificaciones de temporización adecuadas.

G.2.2. Pruebas del cliente PANA en el escenario remoto

Prueba 0: Conexión de un cliente no perteneciente al dominio

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en enviar un cliente PANA al NAS que no pertenezca al dominio del HSS.
- Resultado esperado: El NAS debe enviar directamente un mensaje de error al cliente PANA al no poder enrutarlo.
- Resultado obtenido: El comportamiento experimental es ligeramente diferente al Resultado esperado, el NAS no notifica al cliente el error, sino que éste reenvía el mensaje el número de veces marcado en su fichero antes de dar por terminada la conexión.
- Resultado de la prueba: Pasada.
- Trazas: Al tratarse de una prueba en la que el HSS no llega a intervenir, pueden consultarse las trazas de la Prueba 1 del escenario local, el resultado es idéntico.
- Realización :
  1. Lanzar el NAS de manera normal.
  2. Para lanzar el cliente hacer: ./scripts/pruebalocal.sh 1 desde el directorio /libpana2/ (podemos usar el script local porque el HSS no interviene en realidad)

Prueba 1: Nombre de usuario correcto pero método de autenticación distinto en el cliente y en el HSS

- Categoría: Pruebas de funcionalidad.
- Descripción: En esta prueba NAS y HSS estarán correctamente conectados. Lanzaremos el cliente PANA con un nombre de usuario que pertenezca al dominio del HSS, es decir, que el NAS será capaz de enrutar hacia el HSS. El cliente tiene registrado un método de autenticación distinto al que intentará aplicar el HSS.
- Resultado esperado: El NAS debe enviar los mensajes del cliente al HSS, pero este debe devolverle un resultado de error en la autenticación tras el mensaje de Nak del cliente para intentar cambiar de método.

- Resultado obtenido: El comportamiento experimental es distinto al descrito. El Cliente envía el Nak, pero el HSS no sabe interpretar este resultado, contesta con el reto MD5. El NAS recibe algo que no espera recibir y da un error de análisis y acto seguido un -segmentation fault-[3] como puede apreciarse en las trazas

- Resultado de la prueba: No pasada.

- Trazas:
  En el lado del cliente PANA:

```
(19650|3046812560) TxPCI: id=0 seq=0
(19650|3046812560) From state: OFFLINE to OFFLINE
(19650|3038419856) Event: 2 occurring
(19650|3038419856) RxPSR: id=1242511503 seq=1574923749
(19650|3038419856) TxPSA: id=1242511503 seq=1574923749
(19650|3038419856) From state: OFFLINE to WAIT_EAP_MSG_IN_INIT
Peer:  Timer Started.
(19650|3046812560) Event: 4100 occurring
(19650|3046812560) RxPAR: id=1242511503 seq=1574923750
(19650|3046812560) From state: WAIT_EAP_MSG_IN_INIT to WAIT_SUCC_PBA
Peer:  Parse Request.
Peer: Parsing Identity request.
Peer: Building Identity response.
Peer:  Sent Response.
checking key availability.
EAP Response sent from Peer
(19650|3038419856) Event: 6144 occurring
(19650|3038419856) TxPAN: id=1242511503 seq=1574923750
(19650|3038419856) From state: WAIT_SUCC_PBA to WAIT_EAP_MSG_IN_INIT
(19650|3055205264) Event: 4100 occurring
(19650|3055205264) RxPAR: id=1242511503 seq=1574923751
Peer:  Parse Request.
Peer:  New Method.
Peer:  Do Policy Check.
Peer:  Sent Nak.
EAP Response sent from Peer
(19650|3055205264) From state: WAIT_EAP_MSG_IN_INIT to WAIT_SUCC_PBA
(19650|3030027152) Event: 6144 occurring
(19650|3030027152) TxPAN: id=1242511503 seq=1574923751
(19650|3030027152) From state: WAIT_SUCC_PBA to WAIT_EAP_MSG_IN_INIT
Peer:  Failure.
Authentication failure detected at Peer
Sorry, bob@carissimi.gast.it.uc3m.es try next time !!!
(19650|3046812560) Event: 1024 occurring
```

[3]Tenemos en este hecho una nueva muestra de que la programación de los demonios de OpenDiameter no es demasiado cuidadosa

```
StateMachine[PANA] cannot accept event 1024 at state 2.
(19650|3046812560) From state: WAIT_EAP_MSG_IN_INIT to WAIT_EAP_MSG_IN_INIT
```

En el lado del nasd:

```
(19538|3062553488) RxPCI: id=0 seq=0
(19538|3062553488) PAA is acting stateless
(19538|3062553488) TxPSR: id=1242534950 seq=1455505208
(19538|3062553488) RxPSA: Stateless, id=1242534950 seq=1455505208
(19538|3062553488) New session created [stateless handshake]
PassThrough: Trying a legacy method.
(19538|3045768080) Event: 3 occurring
(19538|3045768080) RxPSA: id=1242534950 seq=1455505208
(19538|3045768080) From state: OFFLINE to WAIT_SUCC_PBA
PassThrough: Trying a legacy method.
AuthIdentityStateTable: Request Prepared.
Passthrough: Request sent and timer started.
(19538|3045768080) Event: 96 occurring
(19538|3045768080) TxPAR: id=1242534950 seq=1455505209
(19538|3045768080) From state: WAIT_SUCC_PBA to OPEN
(19538|3054160784) Event: 2097157 occurring
(19538|3054160784) RxPAN: id=1242534950 seq=1455505209
(19538|3054160784) From state: OPEN to WAIT_SUCC_PBA
Passthrough: Integrity Check.
AuthIdentityStateTable: Do Identity Check.
(19538|3054160784) Routing call for [bob@carissimi.gast.it.uc3m.es]
Sent DER Message.
missing Auth-Request-Type avp.
Parse error
Fallo de segmentación
```

- Realización :
  1. Lanzar el NAS de manera normal.
  2. Para lanzar el cliente hacer: ./scripts/pruebaremota.sh 1 desde el directorio /libpana2/

Prueba 2: Nombre de usuario Incorrecto

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en enviar un cliente PANA que sí pertenece al dominio, pero no esta registrado en la base de datos del HSS
- Resultado esperado: El HSS debe contestar con un mensaje de error al NAS.

- Resultado obtenido: El Resultado obtenido no es el mismo que el descrito en Resultado Esperado. El HSS no contesta nada al NAS al comprobar que el cliente no está en la base de datos. El cliente espera hasta que vence el temporizador marcado en su fichero de configuración y posteriormente interrumpe la conexión.

- Resultado de la prueba: Pasada.

- Trazas: En el lado del cliente PANA:

```
20311|3063167888) TxPCI: id=0 seq=0
(20311|3063167888) From state: OFFLINE to OFFLINE
(20311|3071560592) Event: 2 occurring
(20311|3071560592) RxPSR: id=1241946880 seq=236776713
(20311|3071560592) TxPSA: id=1241946880 seq=236776713
(20311|3071560592) From state: OFFLINE to WAIT_EAP_MSG_IN_INIT
Peer:  Timer Started.
(20311|3037989776) Event: 4100 occurring
(20311|3037989776) RxPAR: id=1241946880 seq=236776714
Peer:  Parse Request.
Peer: Parsing Identity request.
(20311|3037989776) From state: WAIT_EAP_MSG_IN_INIT to WAIT_SUCC_PBA
Peer: Building Identity response.
Peer:  Sent Response.
checking key availability.
EAP Response sent from Peer
(20311|3037989776) Event: 6144 occurring
(20311|3037989776) TxPAN: id=1241946880 seq=236776714
(20311|3037989776) From state: WAIT_SUCC_PBA to WAIT_EAP_MSG_IN_INIT
Peer:  Failure.
Authentication failure detected at Peer
Sorry, bobo@carissimi.gast.it.uc3m.es try next time !!!
```

- Realización :

  1. Lanzar el NAS de manera normal.
  2. Para lanzar el cliente hacer: ./scripts/pruebaremota.sh 2 desde el directorio /libpana2/

Prueba 3: Password Incorrecta

- Categoría: Pruebas de funcionalidad.

- Descripción: Esta prueba consiste en enviar un cliente PANA que sí pertenece al dominio y sí pertenece a la base de datos del HSS, pero que tiene una password incorrecta.

- Resultado esperado: El HSS debe notificar el error de autenticación tras recibir el nonce mal procesado.
- Resultado obtenido: El Resultado obtenido es el mismo que el descrito en Resultado Esperado. El código de error de Diameter-EAP 4001 (authentication Rejected) es recibido.
- Resultado de la prueba: Pasada.
- Trazas: En el lado del cliente PANA:

```
Peer: Building Identity response.
Peer: Sent response.
chcking key availability.
EAPResponse sent from Peer
Peer:  Parse Request.
Peer:  New Method.
Peer:  Do Policy Check.
Peer:  Creating Method.
Peer:  Do Integrity Check.
PeerMDChallenge: Do Identity Check.
EapPeerMD5Challenge: Identifier = 2.
PeerMD5ChallengeStateTable: Response Prepared.
Peer:  Sent Response.
checking key availability.
EAP Response sent from Peer
Peer: Failure received.
Peer:  Failure.
Authentication failure detected at Peer
Sorry, bob@carissimi.gast.it.uc3m.es try next time !!!
```

  En el lado del nasd:

```
[] AAA_MULTI_ROUND_AUTH received.
[] forwarding EAP Request to passthrough.
Passthrough: Request sent and timer started.
Passthough: Integrity check.
[] EAP-Response received from passthrough.
[] sending DER.
Sent DER Message.
[] Error was received.
[]   Result-Code=4001.
Passthrough: EAP authentication failed.
```

- Realización :
  1. Lanzar el NAS de manera normal.
  2. Para lanzar el cliente hacer: ./scripts/pruebaremota.sh 3 desde el directorio /libpana2/

Prueba 4: Todos los parámetros correctos

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en enviar un cliente PANA que sí pertenece al dominio, sí está registrado en la base de datos del HSS y tiene la password correcta.
- Resultado esperado: El HSS debe notificar el éxito de autenticación.
- Resultado obtenido: El Resultado obtenido es el mismo que el descrito en Resultado Esperado. Este caso se explicó en detalle en otra sección de este documento, por lo que no se incluirá ninguna traza.
- Resultado de la prueba: Pasada.
- Trazas: No son necesarias.
- Realización :
  1. Lanzar el NAS de manera normal.
  2. Para lanzar el cliente hacer: ./scripts/pruebaremota.sh 4 desde el directorio /libpana2/

Prueba 5: Batería de clientes

- Categoría: Pruebas de carga.
- Descripción: Esta prueba consiste en enviar una batería de clientes PANA al NAS.
- Resultado esperado: El NAS debe ser capaz de enrutar los mensajes de los cliente correctamente, hasta que el número de clientes sea demasiado elevado.
- Resultado obtenido: El Resultado obtenido es que el NAS sólo puede procesar un cliente cada vez, esto se debe a que el NAS escucha PANA en un sólo puerto y su implementación no crea varios hilos según van llegando clientes o aplica algún otro método de multiplexación.
- Resultado de la prueba: No pasada.
- Trazas: No son necesarias.
- Realización :

1. Lanzar el NAS de manera normal.
2. Lanzar el servidor AAA de manera normal.
3. Para lanzar el cliente hacer: ./scripts/pruebaremota.sh 5 desde el directorio /libpana2/

Prueba 6: Servidor HSS fuera de línea

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en enviar el cliente PANA al NAS, perteneciendo el cliente a un dominio que el NAS sí tiene registrado pero que no se encuentra conectado.
- Resultado esperado: El NAS debe enviar un mensaje de error al Cliente PANA para notificarle que no puede entregar sus mensajes.
- Resultado obtenido: El Resultado obtenido es que el NAS no notifica al cliente el error, sino que éste, tras el número de intentos marcados en su fichero de configuración, da la conexión por fallida.
- Resultado de la prueba: Pasada.
- Trazas: consultar prueba número 1 de este mismo apartado.
- Realización :
  1. Lanzar el NAS de manera normal pero con el HSS no conectado (o modificar la dirección de enrutamiento).
  2. Para lanzar el cliente hacer: ./scripts/pruebalremota.sh 4 desde el directorio /libpana2/

Prueba 7: Conexión directa del cliente PANA con el HSS

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en enviar el cliente PANA al HSS sin pasar por el NAS.
- Resultado esperado: El servidor AAA debe descartar silenciosamente los mensajes del cliente y el cliente debe desconectarse tras el número de intentos de conexión marcados en su fichero de configuración.
- Resultado obtenido: El Resultado obtenido es el especificado en el resultado esperado.

- Resultado de la prueba: Pasada.
- Trazas: En el lado del cliente (El único que muestra trazas en este test): Mismas trazas que en la prueba 0.
- Realización :
    1. Para lanzar el cliente hacer: ./scripts/pruebalocal.sh 7 desde el directorio /libpana2/

Prueba 10: Gestión de temporizadores

- Categoría: Pruebas de configuración.
- Descripción: Esta prueba consiste en probar distintos Auth-Life-Time y Grace-period-Time para el cliente.
- Resultado esperado: Los tiempos deben ser los especificados
- Resultado obtenido: El Resultado obtenido es el especificado en el resultado esperado, al autenticación del cliente es correcta.
- Resultado de la prueba: Pasada.
- Trazas: No se aplica
- Realización :
    1. Lanzar el NAS de manera normal con algún temporizador modificado.
    2. Para lanzar el cliente hacer: ./scripts/pruebaremota.sh 4 desde el directorio /libpana2/

Prueba 11: Distinto MD5-Digest

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en calcular de manera distinta el MD5-Digest en el cliente PANA y en el HSS.
- Resultado esperado: El servidor debe enviar un mensaje de error de autenticación al final de la comunicación, como último de sus mensajes.
- Resultado obtenido: El Resultado obtenido es el especificado en el resultado esperado. El resultado es equivalente al de la prueba de password incorrecta.

- Resultado de la prueba: Pasada.

- Trazas:
  En el lado del cliente PANA:

```
Peer: Building Identity response.
Peer: Sent response.
chcking key availability.
EAPResponse sent from Peer
Peer:  Parse Request.
Peer:  New Method.
Peer:  Do Policy Check.
Peer:  Creating Method.
Peer:  Do Integrity Check.
PeerMDChallenge: Do Identity Check.
EapPeerMD5Challenge: Identifier = 2.
PeerMD5ChallengeStateTable: Response Prepared.
Peer:  Sent Response.
checking key availability.
EAP Response sent from Peer
Peer: Failure received.
Peer:  Failure.
Authentication failure detected at Peer
Sorry, bob@carissimi.gast.it.uc3m.es try next time !!!
```

  En el lado del nasd:

```
[] AAA_MULTI_ROUND_AUTH received.
[] forwarding EAP Request to passthrough.
Passthrough: Request sent and timer started.
Passthough: Integrity check.
[] EAP-Response received from passthrough.
[] sending DER.
Sent DER Message.
[] Error was received.
[]   Result-Code=4001.
Passthrough: EAP authentication failed.
```

- Realización : Para realizar este test es necesario modificar el código del fichero eap_md5.cxx que es donde se encuentra la manera de realizar el MD5 Digest. Dado que el resultado es equivalente a escribir una password incorrecta no se considera necesario realizar todo este proceso. También sería válido modificar el modo de calcularlo en el HSS[4]

[4]En realidad, modificar los parámetros de entrada del MD5-Digest equivale a poner uno de ellos incorrecto, ya que la salida tiene un número determinado de bytes.

G.3. Pruebas con EAP-TLS

Nota importante: Es recomendable consultar el anexo de manual de usuario para iniciar ambas partes.

G.3.1. Pruebas del NAS y el servidor AAA sin cliente

Prueba 1 : Arranque de NAS y el servidor AAA en distinto orden

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en arrancar el NAS y el servidor AAA en distinto.
- Resultado esperado: Independientemente del orden, ambos extremos deben realizar el intercambio de capacidades y enviarse mutuamente los mensajes de tipo Wacthdog.
- Resultado obtenido:La prueba se efectuó 50 veces. De estas 50 veces se observó que, si el orden de arranque era primero el NAS y después el servidor AAA, se producía el intercambio de mensajes Wacthdog en ambos extremos en 31 ocasiones. Si el orden de arranque era el contrario (primero el NAS y posteriormente el servidor AAA) la comunicación falló en 22 ocasiones.
  Con esta simple estimación puede apreciarse que la tasa de error en el primer caso es de 62 % y en el segundo caso de 44 %.
  Las estimaciones no son del todo correctas ni rigurosas, pero de nuevo nos encontramos con el problema de comunicación entre el NAS y servidor AAA que observábamos con el otro servidor AAA. El hecho de cambiar de servidor AAA no ha solucionado esta contingencia.
- Resultado de la prueba: No pasada.
- Trazas: No se precisan.
- Realización : No es necesario ejecutar ningún script.

Prueba 2: Intento de comunicación con ninguna aplicación común

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en intentar comunicar un NAS y un servidor AAA con ninguna aplicación en común (dentro de los ficheros de configuración, la parte marcada como auth_application_id.

- Resultado esperado: La comunicación debe acabar con un mensaje del tipo DIAMETER_NO_COMMON_APPLICATION.
- Resultado obtenido: El resultado obtenido es distinto al esperado. El intercambio de capacidades y el Wacthdog se intercambian a pesar de no haber aplicaciones comunes.
- Resultado de la prueba: No pasada.
- Trazas:

```
(18796|2929261456) Acceptor Peer failed establishing state
                                          [5010], closing
```

- Realización :
  1. lanzar el servidor AAA con TLS.
  2. Para lanzar el NAS hacer: ./scripts/pruebaremota.sh 7 desde el directorio /libpana2/

Prueba 3: Intento de comunicación sin métodos de seguridad comunes

- Categoría: Pruebas de funcionalidad.
- Descripción: Esta prueba consiste en intentar comunicar un NAS y un servidor AAA sin métodos de seguridad en común, en nuestro caso, en un fichero de configuración habilitaremos TLS y en el otro no.
- Resultado esperado: La comunicación debe acabar con un mensaje del tipo DIAMETER_NO_COMMON_SECURITY.
- Resultado obtenido: El resultado obtenido es el explicado en Resultado esperado
- Resultado de la prueba: Pasada.
- Trazas:

```
(26292|2977926032) Acceptor Peer failed establishing state
                                       [3004], closing
```

- Realización : Para la realización basta con intentar comunicar el NAS que usábamos para MD5, es decir, con el fichero de configuración con la opción de TLS deshabilitada, con el servidor AAA con TLS.

G.3.2. Pruebas del cliente EAP-TLS

Para la realización de estas pruebas se asume que el servidor de AAA y el NAS están correctamente conectados.
No escribiremos las trazas completas en este caso, dado que, en general, las trazas serán más concisas que en caso de MD5, debido a que gran parte de la comunicación, especialmente la parte de PANA, es idéntica. Por otro lado, la parte de TLS es mucho más larga que la parte de EAP-MD5, eliminando trazas innecesarias ganaremos espacio para esta parte. En total son 8 pruebas:

- Prueba 1: Método de autenticación incorrecto: En esta prueba intentaremos autenticar un cliente de PANA con MD5 en un servidor que sólo soporta EAP-TLS.
- Prueba 2: Autenticación correcta.
- Prueba 3: certificado incorrecto: En esta prueba se intentará la autenticación TLS con un certificado mal construido (en el lado del cliente o en el del servidor).
- Prueba 4: Clave privada de usuario incorrecta.
- Prueba 5: Ficheros inexistentes: En esta prueba se intentará la autenticación TLS con alguno de los ficheros necesarios no presente.
- Prueba 6: Rutas de los ficheros mal especificadas.
- Prueba 7: CA desconocida. Uno de los extremos no conoce a al autoridad de certificación que se usará.
- Prueba 8: Certificados caducados.

Prueba 1: Método de autenticación incorrecto

- Categoría: Pruebas de funcionalidad.
- Descripción: En esta prueba intentaremos autenticar un cliente de PANA con MD5 en un servidor que sólo soporta EAP-TLS.
- Resultado esperado: La comunicación debe acabar en error tras el envío del NAK por parte del cliente PANA al recibir el Server-Hello. El servidor no aceptará el mensaje de Nak ya que no desea bajar el nivel de seguridad de la conexión, y lo descartará silenciosamente.
- Resultado obtenido: El resultado obtenido es el explicado en Resultado esperado.

- Resultado de la prueba: Pasada.
- Trazas: En el lado del cliente:

```
 EAP Response sent from Peer
(18022|3055045520) Event: 6144 occurring
(18022|3055045520) TxPAN: id=1242303721 seq=1908367891
(18022|3055045520) From state: WAIT_SUCC_PBA to WAIT_EAP_MSG_IN_INIT
(18022|3055045520) Event: 4100 occurring
(18022|3055045520) RxPAR: id=1242303721 seq=1908367892
Peer:  Parse Request.
Peer:  New Method.
Peer:  Do Policy Check.
Peer:  Sent Nak.
EAP Response sent from Peer
(18022|3055045520) From state: WAIT_EAP_MSG_IN_INIT to WAIT_SUCC_PBA
(18022|3046652816) Event: 6144 occurring
(18022|3046652816) TxPAN: id=1242303721 seq=1908367892
(18022|3046652816) From state: WAIT_SUCC_PBA to WAIT_EAP_MSG_IN_INIT
Peer:  Failure.
Authentication failure detected at Peer
Sorry, testuser@localdomain1.net try next time !!!
```

Servidor AAA:

```
State Machine Started with Type
<---------- AuthTls: Building a Start message.
EapRequestTlsParser::parseAppToRaw LENGTH PACKET 0
 Backend: Request sent and timer started.
EAP Request sent from backend authenticator
[] EAP Request received from backend.
[] Sending DEA with continue result-code.
(17945|3002051472) Server session in stateless mode,
                            extending access time
(17945|3002051472) Session Timeout: 30
(17945|3002051472) Auth Lifetime  : 29
(17945|3002051472) Grace Period   : 30
Sent DEA Message.
[] Copying DER to DEA.
[] forwarding EAP Response to authenticator.
EAP Response forwarded to backend authenticator
Backend: Message Discarded.
```

- Realización :
  - Arrancar el NAS y el servidor AAA con TLS.
  - Arrancar la prueba tecleando desde /libpana2 : ./scripts/pruebatls.sh 1

Prueba 2: Autenticación correcta

- Categoría: Pruebas de funcionalidad.
- Descripción: En esta prueba conseguiremos una autenticación con TLS correcta.
- Resultado esperado: La comunicación TLS es correcta.
- Resultado obtenido: La comunicación TLS es correcta.
- Resultado de la prueba: Pasada.
- Trazas: No es necesario incluirlas, ya que este caso de ejecución se explico exhaustivamente durante el desarrollo del proyecto.
- Realización :
  - Arrancar el NAS y el servidor AAA con TLS.
  - Arrancar la prueba tecleando desde /libpana2 : ./scripts/pruebatls.sh 2

Prueba 3: certificado incorrecto

- Categoría: Pruebas de funcionalidad.
- Descripción: En esta prueba se intentará la autenticación TLS con un certificado mal construido (en el lado del cliente o en el del servidor).
- Resultado esperado: La comunicación TLS se interrumpe por culpa de este certificado.
- Resultado obtenido: El resultado obtenido coincide con el esperado. Se produce un error de certificado a nivel de SSL. Sin embargo, el servidor de AAA acaba de con un segmentation fault.
- Resultado de la prueba:No pasada.
- Trazas:

```
13115:error:0906D06C:PEM routines:
   PEM_read_bio:no start line:pem_lib.c:647:Expecting: CERTIFICATE
13115:error:20074002:BIO routines:
       FILE_CTRL:system lib:bss_file.c:354:
13115:error:140AD002:SSL routines:
     SSL_CTX_use_certificate_file:system lib:ssl_rsa.c:470:
```

```
rlm_eap_tls: Error reading certificate fileC
 rlm_eap_tls: Error creating new SSL12836:
    error:140BA0C3:SSL routines:SSL_new:null ssl ctx:ssl_lib.c:245:
Fallo de segmentación
```

- Realización : Modificar uno de los ficheros de certificados del cliente de manera aleatoria.

Prueba 4: clave de usuario incorrecta

- Categoría: Pruebas de funcionalidad.
- Descripción: En esta prueba se intentará la autenticación TLS con una clave privada de usuario incorrecta.
- Resultado esperado: La comunicación TLS se interrumpe por culpa de la clave del usuario.
- Resultado obtenido: El resultado obtenido coincide con el esperado. Se produce un error de clave a nivel de SSL.
- Resultado de la prueba: Pasada.
- Trazas:

```
22810:error:0B080074:x509 certificate routines:
          X509_check_private_key:key values mismatch:
                   x509_cmp.c:399:
rlm_eap_tls: Error reading private key file
```

- Realización : Modificar uno de los ficheros de claves del cliente.

Prueba 5: Ficheros inexistentes

- Categoría: Pruebas de funcionalidad.
- Descripción: En esta prueba se intentará la autenticación TLS con alguno de los ficheros necesarios no presente.
- Resultado esperado: La comunicación TLS se interrumpe por culpa del fichero perdido.
- Resultado obtenido: El resultado obtenido coincide con el esperado. Se produce un error de clave a nivel de SSL.
- Resultado de la prueba: Pasada.

- Trazas:

```
13115:error:0906D06C:PEM routines:PEM_read_bio:no start line:
                    pem_lib.c:647:Expecting: CERTIFICATE
13115:error:02001002:system library:fopen:No such file
    or directory:bss_file.c:352:fopen
   ('/home/guillermo/Escritorio/opendiameter-1.0.7-i/
   libdiametereap/test/tls_config/auth/srv-cert.pem','r')
13115:error:20074002:BIO routines:FILE_CTRL:system lib:bss_file.c:354:
13115:error:140AD002:SSL routines:SSL_CTX_use_certificate_file:
                                    system lib:ssl_rsa.c:470:
rlm_eap_tls: Error reading certificate file
```

- Realización : Eliminar del directorio alguno de los ficheros de certificados.

Prueba 6: Rutas de los ficheros mal especificadas

- Categoría: Pruebas de funcionalidad.

- Descripción: En esta prueba se intentará la autenticación TLS con las rutas a los ficheros de claves y certificados mal especificadas

- Resultado esperado: La comunicación TLS se interrumpe por culpa de las rutas y se avisa por pantalla de los sucedido

- Resultado obtenido: El resultado obtenido es que se produce la interrupción de la comunicación y se avisa al usuario de los motivos por pantalla.

- Resultado de la prueba: Pasada.

- Trazas:

```
22692:error:02001002:system library:fopen:No such file or directory:
                  bss_file.c:352:fopen('/home/guillermo/Escritorio/
                opendiameter-1.0.7-i/libdiametereap/test/tls_config/
                                        auth/srv-cert.pem','r')
```

- Realización : Modificar las rutas de los ficheros de configuración de modo que sean incorrectas.

Prueba 7: CA desconocida

- Categoría: Pruebas de funcionalidad.
- Descripción: En esta prueba se intentará la autenticación TLS usando una CA desconocida por uno de los Peers
- Resultado esperado: La comunicación TLS se interrumpe cuando uno de los extremo examina el certificado expedido por la autoridad desconocida
- Resultado obtenido: El resultado obtenido es que se produce la interrupción de la comunicación por los motivos especificados en Resultado esperado.
- Resultado de la prueba: Pasada.
- Trazas: En el cliente:

```
 rlm_eap_tls: <<< TLS 1.0 Handshake [length 004a],
                                     ServerHello ...

TLS_connect: NULL
rlm_eap_tls: <<< TLS 1.0 Handshake [length 05c8],
                                  Certificate ...

--> verify error:num=19:self signed certificate
                          in certificate chain
rlm_eap_tls: >>> TLS 1.0 Alert [length 0002],
                       fatal ...  unknown_ca

TLS Alert write:fatal:unknown CA
TLS_connect:error in NULL
rlm_eap_tls: SSL_read Error23380:error:14090086:SSL routines:
                 SSL3_GET_SERVER_CERTIFICATE:
            certificate verify failed: s3_clnt.c:951:
 Error code is ..... 5
 Error in SSL ..... 5
rlm_eap_tls: Readed bytes 7
PeerTls: AcProcessRequestFinish: AlertSend.
PeerTls: AcBuildResponseAlert
EapRequestTlsParser::parseAppToRaw LENGTH PACKET 7
 Peer:  Sent Response.
Peer: Failure received.
Peer:  Failure.
Authentication failure detected at Peer
Sorry, testuser@localdomain1.net try next time !!!
```

En el lado del servidor:

```
TLS Alert read:fatal:unknown CA
TLS_accept:failed in NULL
rlm_eap_tls: SSL_read Error23339:error:14094418:SSL routines:
             SSL3_READ_BYTES:tlsv1 alert unknown ca:s3_pkt.c:
               1053:SSL alert number 48
23339:error:140940E5:SSL routines:SSL3_READ_BYTES:ssl handshake
                           failure:s3_pkt.c:838:
 Error code is ..... 5
 Error in SSL ..... 5
rlm_eap_tls: BIO_read Error Error code is ..... 5
 Error in SSL ..... 5
```

- Realización : cambiar los certificados de uno de los Peers por certificados expedidos por otra CA.

Prueba 8: Certificados caducados

- Categoría: Pruebas de funcionalidad.
- Descripción: En esta prueba se intentará una autenticación TLS en la que al menos uno de los extremos tiene los certificados caducados.
- Resultado esperado: La comunicación TLS se interrumpirá cuando el Peer detecte que el certificado está caducado.
- Resultado obtenido: El resultado obtenido es el descrito en Resultado esperado.
- Resultado de la prueba: Pasada.
- Trazas:

```
 rlm_eap_tls: <<< TLS 1.0 Handshake [length 004a],
                                      ServerHello ...

TLS_connect: NULL
rlm_eap_tls: <<< TLS 1.0 Handshake [length 05c8],
                                    Certificate ...

--> verify error:num=19:self signed certificate
                         in certificate chain
rlm_eap_tls: >>> TLS 1.0 Alert [length 0002], fatal ...
            certificate_out_of_date
```

```
TLS Alert write:fatal:Certificate Out of Date
TLS_connect:error in NULL
rlm_eap_tls: SSL_read Error23380:error
                    :14090086:SSL routines:
      SSL3_GET_SERVER_CERTIFICATE:certificate
                verify failed: s3_clnt.c:951:
 Error code is ..... 5
 Error in SSL ..... 5
rlm_eap_tls: Readed bytes 7
PeerTls: AcProcessRequestFinish: AlertSend.
PeerTls: AcBuildResponseAlert
EapRequestTlsParser::parseAppToRaw LENGTH PACKET 7
 Peer:  Sent Response.
checking key availability.
Peer: Failure received.
Peer:  Failure.
Authentication failure detected at Peer
Sorry, testuser@localdomain1.net try next time !!!
```

En el lado del servidor:

```
TLS Alert read:fatal:Certificate Out of Date
TLS_accept:failed in NULL
rlm_eap_tls: SSL_read Error23329:error:14094418:SSL routines:
              SSL3_READ_BYTES:tlsv1 alert unknown ca:s3_pkt.c:1053:
                      SSL alert number 48
23339:error:140940E5:SSL routines:SSL3_READ_BYTES:ssl handshake
                                failure:s3_pkt.c:838:
 Error code is ..... 5
 Error in SSL ..... 5
rlm_eap_tls: BIO_read Error Error code is ..... 5
 Error in SSL ..... 5
```

- Realización : Modificar el certificado de uno de los Peers, cambiándolo por un certificado caducado. Será necesario cambiar además las claves para que coincidan con las de este certificado.

Bibliografía

[1] D. Simon B. Ohba and B. Hursk. The EAP-TLS Authentication Protocol (EAP-TLS). (RFC 5216), 2008.

[2] J. Vollbrecht B. Ohba, L. Blunk and J. Carlson. Extensible Authentication Protocol (EAP). (RFC 3748), 2004.

[3] A. Rubens C. Rigney, S. Willens and W. Simpson. Remote Authentication Dial In User Service (RADIUS). (RFC 2865), 2000.

[4] B. Patil H. Tschofenig D. Forsberg, Y. Ohba and A. Yegin. Protocol for Carrying Authentication for Network Access (PANA). (RFC 5191), 2008.

[5] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. (RFC 4346), 2006.

[6] J. Veizades E. Guttman, C. Perkins and M. Day. Service Location Protocol, Version 2. (RFC 2165), 1999.

[7] C. Ginfalon and Miguel A. Garcia-Martin. *3G IP Multimedia Subsystem IMS*. Second edition, 2006.

[8] N. Petroni Y. Ohba J. Vollbrecht, P. Eronen. State Machines for Extensible Authentication Protocol (EAP), Peer and Authenticator. (Internet Draft), 2008.

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

[10] R. Daniel M. Mealling. Network Time Protocol (version 3) Specification, Implementation and Analysis. (RFC 2915), 2000.

[11] David L. Mills. Network Time Protocol (version 3) Specification, Implementation and Analysis. (RFC 1305), 1992.

[12] P. Mockapetris. Domain Names - Implementation and Specification. (RFC 1035), 1987.

[13] P.Ñesser N. Haller, C. Metz and M. Straw. A One-Time Password System. (RFC 2289), 1998.

[14] Defense Advanced Research Projects Agency Information Processing Techniques Office. Transmission Control Protocol. (RFC 793), 1981.

[15] D. Mitton P. Calhoun, D. Spence. Diameter Network Access Server Application. (RFC 4005), 2005.

[16] J. Loughney P. Calhoun and E. Guttman. The Diameter Base Protocol. (RFC 3588), 2003.

[17] T. Hiller P. Eronen and G. Zorn. Diameter Extensible Authentication Protocol (EAP) Application. (RFC 4072), 2005.

[18] C. Perkins. IP Mobility Support for IPv4. (RFC 3344), 2002.

[19] K. Morneault C. Sharp H. Schwarzbauer T. Taylor I. Rytina M. Kalla R. Stewart, Q. Xie and L. Zhang. Stream Control Transmission Protocol. (RFC 2960), 2000.

[20] R. Rivest. The MD5 Message-Digest Algorithm. (RFC 1321), 1992.

[21] R. Lopez V. Fajardo, T. ohba. State Machines for Protocol for Carrying Authentication for Network Access (PANA). (Internet Draft), 2008.