

Reporte Técnico del Decodificador de Protocolo Industrial PRT-7

Introducción

El presente informe describe el diseño y funcionamiento del Decodificador de Protocolo Industrial PRT-7. Este software fue desarrollado en C++ para abordar un desafío de ciberseguridad industrial: el ensamblaje de un mensaje oculto a partir de un flujo de instrucciones (tramas) recibidas a través de un puerto serial.

El Protocolo PRT-7 no transmite el mensaje de forma directa, sino como una secuencia de operaciones de Carga (LOAD) y Mapeo (MAP). La complejidad radica en que cada operación de mapeo modifica dinámicamente el "disco de cifrado" (un rotor), alterando el resultado de las operaciones de carga subsiguientes. La solución implementada integra Polimorfismo avanzado y la gestión manual de estructuras de datos críticas (Listas Doblemente Enlazadas y Listas Circulares) bajo estrictos requisitos de desarrollo.

Manual Técnico

2.1 Diseño del Sistema (Arquitectura POO)

El decodificador se basa en una arquitectura de Programación Orientada a Objetos (POO) que garantiza la extensibilidad y la gestión de memoria eficiente. La clave del diseño es el uso de Herencia y Polimorfismo para tratar las tramas de manera uniforme.

Jerarquía Polimórfica de Tramas

1. Clase Base Abstracta: TramaBase

- **Propósito:** Define la interfaz para todas las tramas que deben ser procesadas.
- **Contrato:** Contiene el método virtual puro `void procesar(ListaDeCarga* carga, RotorDeMapeo* rotor) = 0;`, obligando a las clases derivadas a definir su lógica de procesamiento.

- **Gestión de Memoria:** Incluye el **Destructor Virtual** virtual `~TramaBase() {}`, un requisito no funcional **crítico** que asegura la correcta liberación de la memoria al hacer delete sobre un puntero `TramaBase*`, previniendo fugas de memoria.

2. Clases Concretas (Derivadas):

- **TramaLoad:** Implementa la lógica para las tramas L,X. Llama al método `getMapeo()` del rotor para decodificar el carácter X y lo inserta en la `ListaDeCarga`.
- **TramaMap:** Implementa la lógica para las tramas M,N. Llama al método `rotar(N)` del rotor para cambiar el estado de cifrado, sin modificar la lista de carga.

2.2 Componentes de Estructuras de Datos

El sistema utiliza dos estructuras de datos dinámicas implementadas **manualmente** para cumplir con el requisito no funcional de **prohibición de la STL** (`std::list`, `std::vector`, `std::string`).

A. Rotor de Mapeo (RotorDeMapeo)

- **Estructura:** Lista Circular Doblemente Enlazada de nodos `NodoRotor`.
- **Función:** Actúa como el disco de cifrado o Rueda de César.
- **Inicialización:** Se inicializa con el alfabeto **A-Z** en orden circular.
- **Métodos Clave:**
 - `rotar(int n)`: Mueve el puntero **cabeza** N posiciones (positivo o negativo) de manera eficiente.
 - `getMapeo(char in)`: Busca la posición relativa de in dentro del alfabeto y devuelve el carácter que está en esa misma posición **relativa** a la cabeza actual del rotor, realizando así la decodificación dinámica.

B. Lista de Carga (ListaDeCarga)

- **Estructura:** Lista Doblemente Enlazada (no circular) de nodos `NodoCarga`.
- **Función:** Almacena el mensaje decodificado en el orden en que las tramas LOAD son procesadas.

- **Métodos Clave:**

- insertarAlFinal(char dato): Asegura que el mensaje se ensamble secuencialmente.
- imprimirMensaje(): Expone el resultado final.

2.3 Desarrollo y Lógica de Decodificación

El proceso de desarrollo se centró en la robustez de la comunicación de bajo nivel y el parseo manual de las tramas.

Comunicación Serial (Multiplataforma)

El código utiliza directivas de preprocesador (`#ifdef _WIN32`) para soportar la comunicación serial en entornos POSIX (Linux/macOS) y Windows, cumpliendo con el requisito de usar librerías estándar o APIs de bajo nivel.

- **Linux/POSIX:** Utiliza las librerías `<fcntl.h>`, `<unistd.h>`, y `<termios.h>`.
 - `abrirPuertoSerial`: Configura el puerto a **9600 baudios** con ajustes **8N1** y deshabilita el procesamiento de entrada/salida para un modo *raw*.
- **Windows:** Utiliza funciones de la API `windows.h` (`CreateFileA`, `GetCommState`, `SetCommState`).
- `leerLineaSerial`: Implementada manualmente para leer byte a byte, buscando terminadores de línea (`\n` o `\r`), esencial para el análisis de protocolos de bajo nivel.

Parseo y Bucle Principal

El `main()` coordina la lectura, instanciación y procesamiento:

1. **Inicialización:** Se crean el `RotorDeMapeo` y la `ListaDeCarga` dinámicamente.
2. **Bucle:** Lee líneas del serial, esperando la señal END.
3. **Análisis Manual:** El código realiza un análisis de la cadena recibida (`char*`) para determinar el tipo de trama (L o M) y extraer el parámetro asociado.
 - Para M,N, el valor de rotación (N) se convierte a entero manualmente, manejando el signo negativo, en cumplimiento del requisito de **parseo C-style**.

4. **Ejecución Polimórfica:** Se utiliza el puntero de la clase base (TramaBase*) para llamar a trama->procesar(...), aplicando la acción correcta al rotor o a la lista de carga.
5. **Limpieza:** Después de cada procesamiento, se llama a **delete trama;** para liberar la memoria del objeto TramaLoad o TramaMap instanciado, reforzando la correcta gestión de memoria.

El mensaje final se revela al finalizar el flujo de datos mediante miListaDeCarga->imprimirMensaje().