Documentación Sistema de Transacciones Seguras con Clave Dinámica:

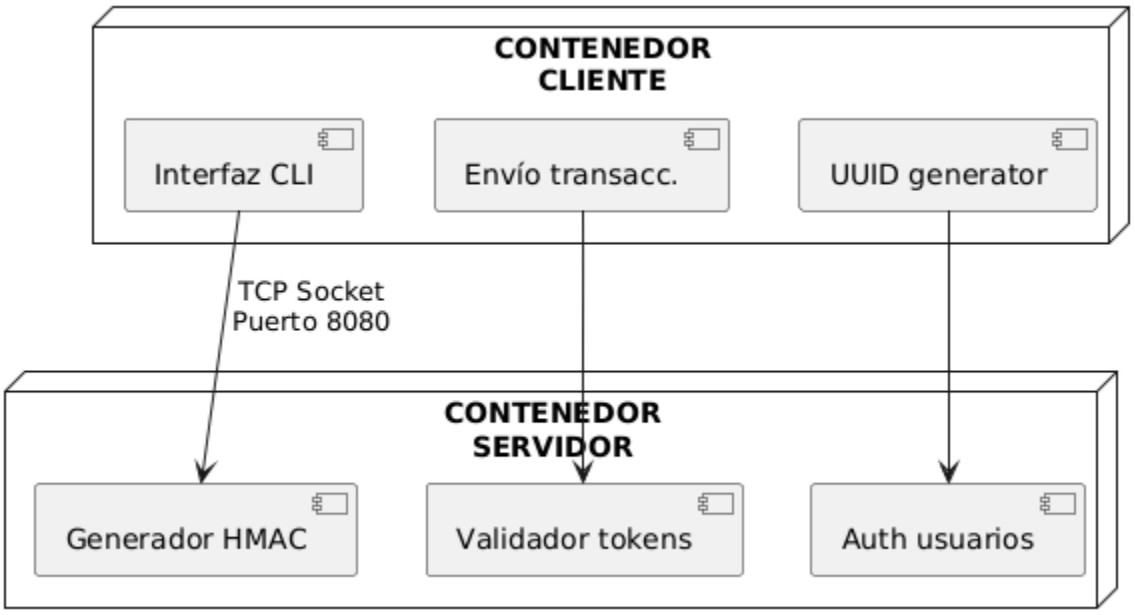
Laura Escobar

Ziuvar Ruiz

1) Descripción del Proyecto

Sistema de transacciones seguras implementado con dos contenedores Docker que se comunican mediante sockets TCP, utilizando un sistema de clave dinámica similar al de Bancolombia basado en tokens temporales con algoritmos criptográficos.

Arquitectura del Sistema



Imágenes Docker Utilizadas

Imagen Base: debian:bullseye-slim

- **Razón de elección**: Tamaño mínimo (~80MB) con herramientas esenciales
- **Ventajas**: Menor superficie de ataque, descargas más rápidas, menor uso de disco

Dependencias Instaladas:

```
RUN apt-get update && apt-get install -y \
    g++ \              # Instala el compilador de C++ (GNU Compiler Collection)
    libssl-dev \       # Instala las bibliotecas de desarrollo de OpenSSL
(necesarias para HMAC, SHA, etc.)
    uuid-dev \         # Instala las bibliotecas para generar UUIDs (libuuid)
    && rm -rf /var/lib/apt/lists/*  # Elimina la caché de paquetes para reducir
el tamaño de la imagen
```

Tamaño Final de Contenedores:

- **Servidor**: ~160MB
- **Cliente**: ~160MB

Sistema de Seguridad Implementado

Algoritmo Criptográfico: HMAC-SHA256

```
// Función que genera una clave dinámica de 8 caracteres a partir de un secreto y
unos datos utilizando HMAC-SHA256
std::string generate_dynamic_key(const std::string& secret, const std::string&
data) {

    // Longitud del hash SHA256 (32 bytes)
    unsigned int len = SHA256_DIGEST_LENGTH;

    // Genera el HMAC usando SHA256.
    // - `EVP_sha256()` indica que se usará SHA256 como algoritmo de hash.
    // - `secret.c_str()` es el secreto usado como clave HMAC.
    // - `data.c_str()` es el mensaje o datos a firmar.
    // El resultado es un puntero a un arreglo de bytes (hash).
```

```
    unsigned char* result = HMAC(
        EVP_sha256(),                          // Algoritmo de hash (SHA256)
        secret.c_str(),                        // Clave secreta
        secret.length(),                       // Longitud de la clave
        reinterpret_cast<const unsigned char*>(data.c_str()), // Datos a firmar
        data.length(),                         // Longitud de los datos
        nullptr,                               // Parámetro opcional (salida)
        nullptr                                // Parámetro opcional (longitud de
salida)
    );

    // Si el resultado es nulo (falló la operación HMAC), se retorna una cadena
por defecto.
    if (!result) return "00000000";  // Fallback si falla HMAC

    // Se prepara un arreglo de caracteres para almacenar la cadena hexadecimal
(64 caracteres + null terminator)
    char output[2 * SHA256_DIGEST_LENGTH + 1] = {0};

    // Recorre cada byte del resultado HMAC y lo convierte a una cadena
hexadecimal (2 caracteres por byte)
    for (unsigned int i = 0; i < len; i++) {
        snprintf(&output[i * 2], 3, "%02x", result[i]); // Escribe 2 caracteres
hexadecimales en la posición adecuada
    }

    // Devuelve los primeros 8 caracteres del hash hexadecimal como clave
dinámica
    return std::string(output).substr(0, 8);
}
```

Componentes de Seguridad:

1. **Clave secreta compartida**: "clave_secreta_123"
2. **Timestamp sincronizado**: Períodos de 30 segundos
3. **Token HMAC**: 8 caracteres hexadecimales
4. **Validación de credenciales**: Usuario/contraseña

Estructura de Transacción

Formato: ID|Timestamp|Monto|Usuario|Contraseña|Clave_Dinámica

Campo	Descripción	Ejemplo
ID	UUID único	550e8400-e29b-41d4-a716-446655440000
Timestamp	ISO 8601 UTC	2024-12-10T15:30:00Z
Monto	Valor transacción	1000.50
Usuario	Nombre usuario	user1
Contraseña	Contraseña usuario	pass123
Clave Dinámica	Token de 8 caracteres	a1b2c3d4

Despliegue

Prerrequisitos

- Docker Engine
- Docker Compose

Imágenes en DockerHub

- **Servidor**: ziuvarruiz/clavedinamicadocker-server:latest
- **Cliente**: ziuvarruiz/clavedinamicadocker-client:latest

Comandos de Despliegue

1. **Iniciar sistema**:

docker-compose up -d

2. **Ver clave dinámica actual**:

docker logs clavedinamicadocker-server

3. **Ejecutar cliente**:

docker exec -it clavedinamicadocker-client ./client

Usuarios de Prueba

- user1 / pass123
- user2 / pass456

Configuración Docker Compose

```yaml
  server:
    image: ziuvarruiz/clavedinamicadocker-server:latest  # Usa la imagen del
servidor desde Docker Hub
    ports:
      - "0.0.0.0:8080:8080"        # Expone el puerto 8080 del contenedor a todos
los interfaces del host
    container_name: clavedinamicadocker-server  # Nombre personalizado del
contenedor
    tty: true                     # Asigna una terminal al contenedor (útil para
modo interactivo)
    stdin_open: true              # Mantiene abierta la entrada estándar

  client:
    image: ziuvarruiz/clavedinamicadocker-client:latest  # Usa la imagen del
cliente desde Docker Hub
    container_name: clavedinamicadocker-client           # Nombre personalizado
del contenedor
    depends_on:
      - server                    # Espera a que el contenedor del servidor esté
listo antes de iniciar
    tty: true                     # Asigna una terminal al contenedor
    stdin_open: true              # Mantiene abierta la entrada estándar
```

Flujo de Autenticación

1. **Servidor genera token**: HMAC-SHA256(secret, timestamp_30s)
2. **Cliente solicita datos**: Usuario, contraseña, monto, clave dinámica
3. **Transmisión**: Cliente envía transacción completa por socket TCP
4. **Validación servidor**:
    a. Verifica credenciales
    b. Valida token actual o anterior (60s ventana)
5. **Respuesta**: Autorización o rechazo

Optimizaciones Implementadas

Tamaño de Contenedores

- **Imagen base mínima**: debian:bullseye-slim
- **Compilación interna**: Código C++ compilado dentro del contenedor
- **Limpieza de cache**: rm -rf /var/lib/apt/lists/*
- **Dependencias mínimas**: Solo librerías esenciales

Control de Dependencias

- **Sin frameworks**: Implementación directa con OpenSSL
- **Sockets nativos**: POSIX sockets sin abstracción
- **Criptografía directa**: HMAC-SHA256 sin wrappers
- **UUID nativo**: libuuid sin dependencias adicionales

Conexión entre Contenedores

Configuración de Red

- **Protocolo**: TCP
- **Puerto**: 8080
- **Resolución DNS**: Docker interno (server hostname)
- **Red**: Bridge por defecto

Manejo de Errores de Conexión

```
        // Resolver hostname "server"
        struct hostent* host = gethostbyname("server");
        if (host == nullptr) {
            std::cerr << "\n  Error: No se pudo conectar con el servidor.
Presione ENTER para continuar.";
            close(sock);       // Cierra el socket antes de reintentar
            std::cin.get();    // Espera que el usuario presione ENTER
            continue;          // Vuelve al inicio del bucle para intentar otra
vez
        }
```

Seguridad y Validaciones

Validaciones Cliente

- Campos obligatorios no vacíos
- Formato numérico para montos
- Conexión TCP exitosa

Validaciones Servidor

- Formato transacción (6 campos separados por |)
- Credenciales válidas en mapa de usuarios
- Token HMAC válido (actual o período anterior)

Sincronización Temporal

```
auto now = std::time(nullptr);
now = now - (now % 30); // Redondeo a múltiplos de 30s
```

2) Replicación del Proyecto

Pasos para Otro Grupo

1. **Clonar repositorio**:

```
git clone https://github.com/ziuvar/DockerClaveDinamica.git
cd clave-dinamica-docker
```

2. **Ejecutar sistema**:

```
docker-compose up -d
```

3. **Probar transacciones**:

```
# Terminal 1: Ver clave actual
docker logs -f clavedinamicadocker-server

# Terminal 2: Ejecutar cliente
docker exec -it clavedinamicadocker-client ./client
```

Archivos Necesarios

- docker-compose.yml: Configuración de servicios
- server/Dockerfile: Imagen servidor
- client/Dockerfile: Imagen cliente
- server/server.cpp: Código servidor
- client/client.cpp: Código cliente

Troubleshooting

Error	Solución
Contenedores no se conectan	Verificar puerto 8080 y hostname `server`
Clave dinámica inválida	Sincronizar tiempo, usar clave actual del servidor
Usuario incorrecto	Usar credenciales de prueba definidas

Construcion en local

El sistema requiere que las imágenes estén disponibles en DockerHub para funcionar:

1. No hay construcción local: El docker-compose no tiene directivas build:
2. Imágenes remotas: Usa image: apuntando a DockerHub
3. Sin Dockerfiles locales ejecutándose: Los Dockerfiles están solo como referencia

Para Eliminar la Dependencia de DockerHub

Si se quiere que el sistema funcione sin DockerHub, se tendría que modificar el docker-compose.yml así:

```
version: '3.9'

services:
  # Servicio del servidor
  server:
    build:
      context: ./server        # Ruta al Dockerfile del servidor
    ports:
      - "8080:8080"            # Expone el puerto 8080 del contenedor al host
    container_name: clavedinamicadocker-server  # Nombre personalizado del
contenedor
```

```
    tty: true                   # Mantiene el contenedor corriendo con una
terminal asignada
    stdin_open: true            # Permite entrada estándar (útil para pruebas
interactivas)

  # Servicio del cliente
  client:
    build:
      context: ./client         # Ruta al Dockerfile del cliente
    container_name: clavedinamicadocker-client  # Nombre personalizado del
contenedor
    depends_on:
      - server                  # Espera que el contenedor del servidor esté listo
antes de iniciar
    tty: true                   # Mantiene el contenedor corriendo con una
terminal asignada
    stdin_open: true            # Permite entrada estándar (útil para pruebas
interactivas)

services:
```