Universidad de los Andes
Ingeniería de Sistemas y Computación
ISIS 2203 Infraestructura Computacional

Caso de Estudio 2 – Canales Seguros
Logística y Seguridad Aeroportuaria

Objetivos

- Identificar los requerimientos de seguridad de los canales usados para transmisión de la información en el sistema de Logística y Seguridad Aeroportuaria.
- Construir un prototipo a escala del sistema que permita satisfacer algunos de los requerimientos de seguridad identificados, entendiendo las garantías de seguridad y las limitaciones de la implementación propuesta.

Problemática:

Como se indicó en el enunciado del caso, el sistema cuenta con varias aplicaciones: Novasoft financiero en línea y fuera de línea, OpenERP, Sistema Time & Attendance, correo electrónico y página web. En este contexto, surgen diversos problemas de seguridad para algunas de las transacciones que el sistema soporta, tanto a nivel de transmisión, como en procesamiento y almacenaje de datos. Como consecuencia, es necesario evaluar riesgos y determinar medidas para mitigar los problemas detectados.

Su tarea en este caso es actuar como consultor de seguridad y analizar, considerando solo aspectos de seguridad, las tareas relacionadas con el sistema Time & Attendance.

Tareas:

Todos los servidores del sistema implementan control de acceso a nivel del sistema operativo y ejecutan transacciones solo para usuarios autenticados, de acuerdo con los permisos asignados. Las aplicaciones también manejan usuarios, cada una maneja su propio archivo de configuración de usuarios y soporta operaciones de cifrado.

A. [15%] Análisis y Entendimiento del Problema

Considerando el sistema descrito en el párrafo anterior:

1. Identifique los datos que maneja el sistema Time & Attendance y que deben ser protegidos.
2. Identifique los requerimientos de seguridad para cada uno de los datos del punto anterior. Explique su respuesta en cada caso y responda la pregunta: si no se garantiza ese requerimiento para ese dato ¿cómo podría afectar a la entidad?
3. Identifique cuatro vulnerabilidades de este sistema, teniendo en cuenta únicamente aspectos técnicos o de procesos (no organizacionales). Identifique vulnerabilidades no solo en lo relacionado con la comunicación sino también con el almacenamiento y procesamiento de los datos. Explique su respuesta en cada caso.

(*) *Sus explicaciones DEBEN corresponder al contexto planteado. NO escriba respuestas genéricas*.

B. [85%] Implementación del Prototipo

En esta parte del proyecto nos centraremos únicamente en el sistema de intercambio de información con el sistema Time & Attendance. Un agente o supervisor enviará un identificador de usuario y la localización, y el servidor responderá con el identificador de usuario y la hora de registro. Usted solo debe construir la aplicación cliente; este cliente debe comunicarse con el servidor que se le entregará.

Como queremos concentrarnos en el protocolo de comunicaciones y sus requerimientos de seguridad, construiremos una aplicación cliente/servidor simplificada en Java. El cliente y el servidor se comunicarán siguiendo el protocolo descrito en la Figura 1.

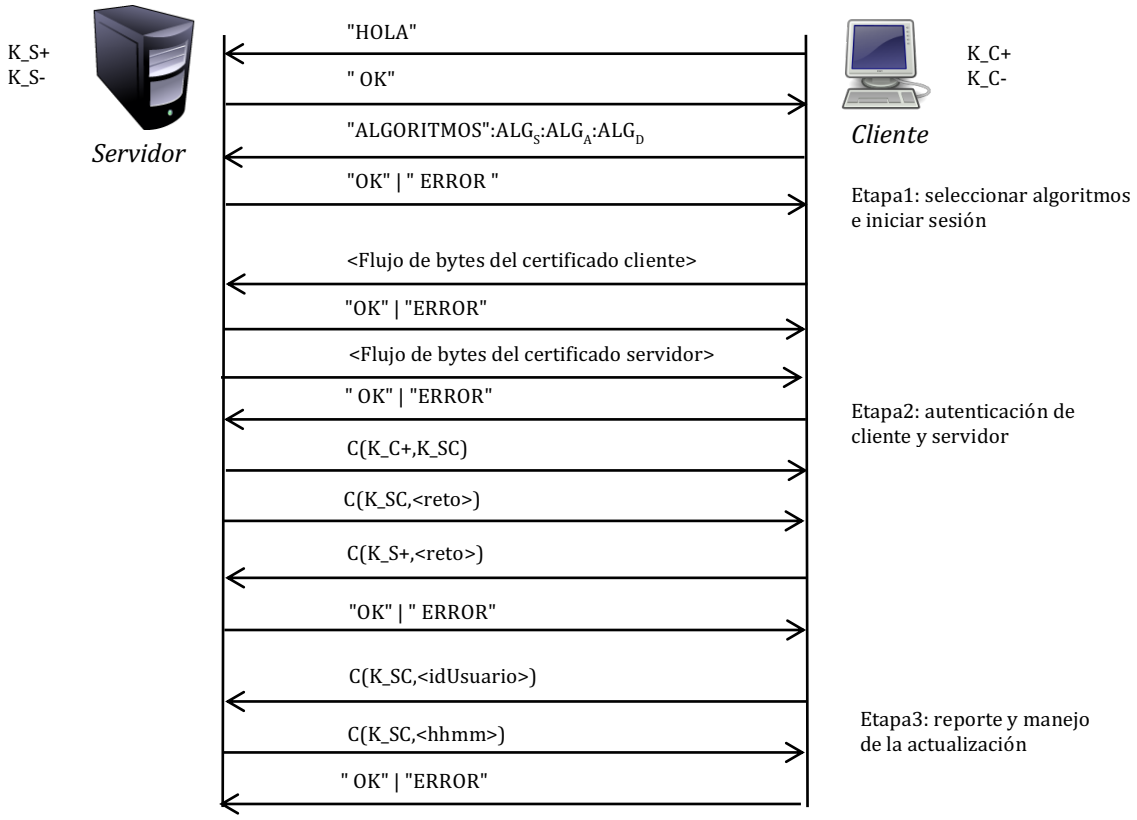

Figura 1. Protocolo de comunicación entre cliente y servidor.

TENGA EN CUENTA:

- El protocolo de comunicación maneja:
  - Cadenas de Control: "HOLA", "ALGORITMOS", "OK", "ERROR".
  - Separador Principal: " : "
- A continuación, se presentan los algoritmos disponibles en el servidor para manejo de integridad y confidencialidad. Es decir, usted debe seleccionar un algoritmo para reemplazar cada una de las cadenas: <ALG_S>, <ALG_A> y <ALG_{HMAC}> en el protocolo. Los algoritmos disponibles son:
  - Simétricos (ALG_S):
    - AES (Configuración por defecto: Modo ECB, esquema de relleno PKCS5, llave de 128 bits).
    - Blowfish (Configuración por defecto: Cifrado por bloques, llave de 128 bits).
  - Asimétricos (ALG_A):
    - RSA. (Configuración por defecto: Llave de 1024 bits.)
  - HMAC (ALG_{HMAC}):
    - HmacSHA1
    - HmacSHA256
    - HmacSHA384
    - HmacSHA512

  Las cadenas que identifican cada uno de los algoritmos son: "AES", "BLOWFISH", "RSA", "HMACSHA1", "HMACSHA256", "HMACSHA384", "HMACSHA512".
- Tanto el servidor como el cliente deben usar el estándar X509 para sus certificados digitales. En un caso real un certificado digital debería ser expedido por una entidad certificadora pero aquí lo generaremos localmente. El certificado debe contener, entre otros, la llave pública para usarla en el proceso de comunicación.
- El servidor usa la librería BouncyCastle para el manejo de certificados.
- La comunicación se realiza a través de sockets de acuerdo con el protocolo de comunicación definido en la Figura 1.

- Las cadenas entre los caracteres "<" y ">" debe reemplazarse por los valores correspondientes.
- El identificador de usuario debe ser un número de 4 dígitos.
- El servidor responde con la hora en formato militar.
- El servidor recibe id de usuario, pero esta información no es verificada en una base de datos. (Para verificar, el protocolo tendría que implementar una fase para registrar un usuario).
- Usted solo debe desarrollar el cliente. La librería BouncyCastle y el .jar del servidor estarán disponibles en SICUA+.
- Para correr el servidor ejecute el comando java -jar <archivo jar> en la línea de comandos. El servidor le pedirá el puerto en el que quiere que corra (deberá configurar su cliente para conectarse a ese puerto).

Entrega:

- Cada grupo debe entregar un zip de un proyecto Java con: La implementación correspondiente al cliente (descrito en la parte B). En el subdirectorio docs debe haber un archivo que incluya el informe con las respuestas a la parte A. **Al comienzo del informe, deben estar los nombres y carnés de los integrantes del grupo**. Si un integrante no aparece en el documento entregado, el grupo podrá informarlo posteriormente. Sin embargo, habrá una penalización: la calificación asignada será distribuida (dividida de forma equitativa) entre los integrantes del grupo.
- El trabajo se realiza en grupos de **2** personas. No debe haber consultas entre grupos.
- El grupo responde solidariamente por el contenido de todo el trabajo, y lo elabora conjuntamente (no es trabajo en grupo repartirse puntos o trabajos diferentes). Se puede solicitar una sustentación a cualquier miembro del grupo sobre cualquier parte del trabajo. Dicha sustentación será parte de la calificación de todos los miembros.
- El proyecto debe ser entregado por Sicua+ por uno solo de los integrantes del grupo.
- **La fecha límite de entrega es sábado 18 de Abril, 2020 a las 23:55 p.m.**

Referencias:

- *Cryptography and network security*, W. Stallings, Ed. Prentice Hall, 2003.
- *Computer Networks*. Andrew S. Tanenbaum. Cuarta edición. Prentice Hall 2003, Caps 7, 8.
- *Blowfish*. Página oficial es: http://www.schneier.com/blowfish.html
- *RSA.* Puede encontrar más información en: http://www.rsa.com/rsalabs/node.asp?id=2125
- *CD X509*. Puede encontrar la especificación en: http://tools.ietf.org/rfc/rfc5280.txt

Anexo

El siguiente fragmento presenta el código usado por el servidor para generar el certificado. El método usa el proveedor BC y métodos de BouncyCastle.

- Para traducir de String a byte[] y viceversa el servidor maneja codificación en base 64. Use DatatypeConverter para construir los métodos apropiados:
```
DatatypeConverter.printBase64Binary(byteArray);
DatatypeConverter.parseBase64Binary(cadena);
```
- Para usar el método `parseBase64Binary(cadena),` la longitud de la cadena debe ser múltiplo de 4. En particular, los valores que el cliente envía al servidor (id de usuario) deben tener una longitud múltiplo de 4. Puede completar la longitud con 0 a la izquierda o a la derecha.
- El servidor siempre envía los datos en formato String y los recibe en el mismo formato:
```
socketParaEnviar.println(cadenaConInformacion);
cadenaConInformacion = socketParaRecibir.readLine();
```
- A continuación, se ilustra el procedimiento que usa el servidor para enviar el certificado.
```
java.security.cert.X509Certificate certificado = generarCertificado(parLlaves);
byte[] certificadoEnBytes = certificado.getEncoded( );
String certificadoEnString = printBase64Binary(certificadoEnBytes);
socketParaComunicacion.println(certificadoEnString);
```

```
public static X509Certificate gc(KeyPair keyPair)
        throws OperatorCreationException, CertificateException {
  Calendar endCalendar = Calendar.getInstance();
  endCalendar.add(Calendar.YEAR, 10);
  X509v3CertificateBuilder x509v3CertificateBuilder = new X509v3CertificateBuilder(
          new X500Name("CN=localhost"),
          BigInteger.valueOf(1),
          Calendar.getInstance().getTime(),
          endCalendar.getTime(),
          new X500Name("CN=localhost"),
          SubjectPublicKeyInfo.getInstance(keyPair.getPublic().getEncoded()));
  ContentSigner contentSigner =
          new JcaContentSignerBuilder("SHA1withRSA").build(keyPair.getPrivate());
  X509CertificateHolder x509CertificateHolder =
          x509v3CertificateBuilder.build(contentSigner);
  return new JcaX509CertificateConverter().setProvider("BC")
          .getCertificate(x509CertificateHolder);
}
```

Sobre Bouncy Castle, pueden encontrar información adicional en el sitio web (https://www.bouncycastle.org/).
Sobre la licencia de Bouncy Castle (https://www.bouncycastle.org/license.html):