Universidad de Sevilla

Escuela Técnica Superior de Ingeniería Informática

PAI 2 - Verificadores de integridad en la transmisión punto-punto para entidad financiera

Grado en Ingeniería Informática – Ingeniería del Software

Seguridad en Sistemas Informáticos e Internet

Curso 2023 – 2024

Participantes:
Juan Luis Ruano Muriedas

José Joaquin Rojas Romero

Antonio José Suárez García

Índice

Resumen ejecutivo

El sistema implementa un protocolo de seguridad para garantizar la integridad de los mensajes bancarios en un entorno propenso a ataques de ***man-in-the-middle*** y de ***replay***.

Para enfrentar estos desafíos, se ha desarrollado un **código en Python** que utiliza **encriptación *HMAC*** con una clave de 112 bits y el algoritmo *SHA256*. Además, se emplea una secuencia aleatoria llamada ***Nonce*** para evitar ataques de *replay*.

Para el intercambio de claves se recomienda el uso del algoritmo ***Diffie-Hellman***, el cual asegura una clave secreta compartida entre los extremos con ***Perfect Forward Secrecy***.

Se han diseñado **pruebas automáticas** que abordan todos los posibles escenarios de transmisión y se registran los resultados para evaluar el porcentaje de éxito. El sistema es fácilmente mantenible, automatizable y ofrece seguridad y confiabilidad en la transmisión de mensajes bancarios.

Verificador de integridad de los mensajes

Estructura del código

Según la política de integridad de la empresa que hay que cumplir se debe tener en cuenta la integridad de los mensajes bancarios ante los riesgos de ataque de **man-in-the-middle** y de **replay**. Para lidiar con el primero, el equipo ha desarrollado el código de manera que el mensaje está encriptado con un método de encriptación **HMAC** con una key generada de 112 bits y usando el algoritmo **SHA256**. De esta forma un atacante que no conozca la clave y modifique el mensaje no pasará el control de seguridad del servidor y no aceptará el mensaje como válido. Para el **reparto de la clave** por parte de los extremos habrá una sección de los algoritmos que recomendamos más adelante.

En cuanto al riesgo de *replay* se usa una secuencia aleatoria llamada **Nonce** y se adjunta con el mensaje en el cálculo del *HMAC* de tal manera que no puedan cambiar el *Nonce* en el mensaje. El *Nonce* **no se podrá repetir nunca**, en el momento que llegue un mensaje correcto con ese *Nonce* se registra en la base de datos y si llega otro mensaje con el mismo *Nonce*, inmediatamente se **descarta** de tal forma que no haya repetición.

De la misma forma que se envía el mensaje, la **respuesta del servido**r estará **encriptada** y también posee verificación por *Nonce*, además, el cliente y servidor confirman que se trate de la misma transacción a través del *Nonce* enviado con anterioridad por el cliente el cual está incluido en el mensaje.

El código está hecho en ***Python*** y se compone del *script "*socketservidor.py" para la parte del servidor y "clientsocket.py" para la parte del cliente. Consideramos que es un lenguaje apropiado pues contiene todas las librerías necesarias, es sencillo de mantener y podemos automatizar las pruebas con facilidad.

Sobre el intercambio de claves

La clave para mantener la seguridad en este sistema es el uso de una clave completamente secreta que sea conocida únicamente por ambos extremos y para ello, la seguridad en el intercambio de la misma es crucial. Existen métodos físicos seguros para el intercambio de claves, como intercambiarla en persona en un pendrive o en un papel. Sin embargo, estos métodos son imprácticos, ya sea por la distancia entre la empresa y el cliente o ya sea por el elevado número de clientes. Nosotros recomendamos el uso del algoritmo ***Diffie-Hellman*** para el intercambio de claves. Este algoritmo es el **más usado** hoy en día y es nuestra recomendación por varios motivos: el primero, aunque una clave se viera comprometida, el atacante **no tendría acceso a las claves anteriores**, ya que el algoritmo genera claves cada

cierto tiempo, confiriéndole *Perfect Forward Secrecy* al algoritmo; el segundo, la **alta eficiencia computacional** del algoritmo y el tercero es la **independencia del canal** de comunicación, ya que *Diffie-Hellman* funciona tanto en canales privados como públicos. El mayor problema que presenta este algoritmo es la posible falsificación de la identidad por parte del atacante, donde puede hacer creer que está comunicándose con el cliente. Afortunadamente esto tiene fácil solución, ya que solo tiene que pedir el certificado digital a su cliente a la hora de establecer la conexión.

Pruebas

Para las pruebas, se ha diseñado un *script* de pruebas que **comprueba los 3 casos posibles de la transferencia** del mensaje: éxito, error de *HMAC* (caso de *man-in-the-middle*) o error de *Nonce* (caso de *replay*). Se ha comprobado también la seguridad del cliente que puede dar: éxito, error por que el nonce no corresponde al *Nonce* enviado con anterioridad, error de la *HMAC* o error por *Nonce* ya registrado. Todos los casos **cumplen las pruebas de calidad** propuestas sobre integridad de la transmisión.

Por último si examina el archivo *pruebas_automáticas.bat* verá que se han automatizado las pruebas de transmisión al servidor de manera que se pueda comprobar fácilmente solo ejecutando el archivo correspondiente y se guardan en archivos log de windows (Estas pruebas solo estarán disponibles para sistemas Windows). Alternativamente también puede ejecutar la clase de pruebas *pruebasCliente.py*.

```
servidor_prueba1.log


Connected by ('127.0.0.1', 56390)
MENSAJE ---------->  23234 2342 200
NONCE ------------>  garIYXugvvcv1GRDDUJxZwbHUgV00r-3A92ICoBYlIg
MAC CALCULATED -->  46949849102d71128bd8a3beaec846c98bfb753f93e5cea87cf8272d23b43700
MAC RECEIVED ---->  46949849102d71128bd8a3beaec846c98bfb753f93e5cea87cf8272d23b43700
Mensaje recibido con Úxito


servidor_prueba2.log


Connected by ('127.0.0.1', 56392)
Error de integridad en el mensaje recibido


servidor_prueba3.log


Connected by ('127.0.0.1', 56393)
**NONCE DUPLICADO, CANCELANDO OPERACIËN...**
```

Figura 1: Captura de las pruebas automáticas de parte del servidor.

```
cliente_prueba1.log


RESPUESTA ---------->  OK
NONCE ------------>  ktJhVM8agtkbaFAyLBSRQyBk7LBAUXFwslOJdvAYOqY
MAC CALCULATED -->  4e5368b8be7ff63c17c9bf03ff84f5ee227ee81211a4b0029e74f4a60a38d1a0
MAC RECEIVED ---->  4e5368b8be7ff63c17c9bf03ff84f5ee227ee81211a4b0029e74f4a60a38d1a0
Mensaje recibido


cliente_prueba2.log


RESPUESTA ---------->  INTEGRITY ERROR
NONCE ------------>  1qyUFqgYLMesqW7O6Ao3ht5JkmqZITg3aIiVGG9nHfE
MAC CALCULATED -->  c18f22fde192ea27db7f0d20a923263067dec4304326547f169242a7ebbbafa4
MAC RECEIVED ---->  c18f22fde192ea27db7f0d20a923263067dec4304326547f169242a7ebbbafa4
Mensaje recibido


cliente_prueba3.log


RESPUESTA ---------->  NONCE ERROR
NONCE ------------>  OLOwsY-iPKTQXZz9LH4fCw0LI9NGwZqAFgzBLdXjsI0
MAC CALCULATED -->  d63dc493a9046e6d7169e2ad960193e78cfbbd1f75ca90f1fc20a33064d7a07f
MAC RECEIVED ---->  d63dc493a9046e6d7169e2ad960193e78cfbbd1f75ca90f1fc20a33064d7a07f
Mensaje recibido
```

Figura 2: Captura de las pruebas automáticas de parte del cliente.

Registro y KPI

El código posee también la cualidad de hacer un **registro de los resultados** de los mensajes obtenidos por el cliente y guardarlos en el archivo *logs.txt*.

```
2024-03-09 13:34:34.873122+00:00, Resultado: Error nonce duplicado, MAC recibida: 8
2024-03-09 13:56:08.418357+00:00, Resultado: Error de integridad, MAC recibida: cb8
2024-03-09 13:56:25.754159+00:00, Resultado: OK!, MAC recibida: cfe81deb57ab08c7b3c
2024-03-09 14:00:11.781010+00:00, Resultado: OK!, MAC recibida: 4636c03b1e4bc6aba35
2024-03-09 14:00:15.786506+00:00, Resultado: Error de integridad, MAC recibida: 6e9
2024-03-09 14:00:19.853016+00:00, Resultado: Error nonce duplicado, MAC recibida: 0
2024-03-11 16:48:03.638227+00:00, Resultado: OK!, MAC recibida: 46949849102d71128bc
2024-03-11 16:48:07.278042+00:00, Resultado: Error de integridad, MAC recibida: 8b0
2024-03-11 16:48:11.907215+00:00, Resultado: Error nonce duplicado, MAC recibida: 7
```

Figura 3: Captura del archivo logs.txt.

A través de este registro, el script *KPI.py* ofrece la posibilidad de leer el log y calcular el **porcentaje de éxito sobre el total de las transmisiones** registradas en dicho archivo.

```
C:\Users\juanl\OneDrive\Escritorio\Cosas de Juanlu\IISS\PAI2>py KPI.py
Tasa de éxito de las transferencias recibidas:
51.42857142857142%
```

Figura 4: Captura de la ejecución del KPI.

Como instalar el verificador

El código descrito anteriormente está adjunto a este documento o lo puede encontrar en nuestro repositorio de *Github*: https://github.com/tric0ma/CAI-1/tree/main/pai-2.

Para la **instalación del verificador** sólo necesita introducir en el servidor el código incluido en el script *socketservidor.py* y ajustar **los parámetros IP y del puerto** de entrada. Utilizar el mismo proceso con los clientes con el script *clientsocket.py* y seguir las recomendaciones dadas en el apartado anterior sobre el intercambio y generación de claves. No olvide que el *logs.txt* es esencial para la generación del porcentaje del *KPI*.

Conclusiones

Usando los métodos de encriptación *HMAC* con *Nonce* para las transmisiones punto a punto y el algoritmo de *Diffie-Hellman* con el uso del certificado digital para el intercambio seguro de claves aportamos una solución sólida al problema propuesto.