Grado en Ingeniería Informática-Ingeniería del Software

2021-2022

Curso de Seguridad en Sistemas Informáticos y en Internet

PAI 2.

VERIFICADORES DE INTEGRIDAD EN LA TRANSMISIÓN PUNTO-PUNTO PARA ENTIDAD FINANCIERA

SECURITY TEAM 11

Matilde Ghidini

Matteo Halilaga

Gabriele Petroni

INDICE

INTRODUCCIÓN

Existen numerosas API en los lenguajes de programación, comandos en los sistemas operativos y herramientas comerciales/software libre que se pueden utilizar para comprobar la integridad de los datos en el almacenamiento mediante la correspondiente generación de resúmenes de mensajes (message digests o checksums). En este Proyecto de Aseguramiento de la Información usaremos técnicas para poder verificar la integridad de datos en la transmisión por redes públicas como Internet y evitar los diferentes tipos de posibles ataques. En este PAI, la Política de Seguridad sobre la Integridad de las Transmisiones de un entidad financera establece que:

"En todas las transferencias bancarias por medios electrónicos no seguros se debe conservar la integridad y confidencialidad de las comunicaciones"

Objetivos del proyecto

1. Desarrollar/Seleccionar el verificador de integridad para los mensajes de transferencia bancaria que se transmiten a través de las redes públicas evitando los ataques de man-in-the-middle y de replay (tanto en el servidor como en el cliente).

2. Desplegar un verificador de integridad en los sistemas cliente/servidor para llevar a cabo la realización de la verificación de forma práctica de los mensajes transmitidos entre un servidor y un cliente.

TECNOLOGIAS Y LIBRERÍAS UTILIZADAS

Lenguaje de programación: Python

Hemos decidido desarrollar nuestro proyecto utilizando el lenguaje Python.

Sqlite3

Para crear una base de datos donde almacenar los NONCEs de los mensajes hemos utilizado la librería Python `Sqlite3`, que es un base de datos SQL self-contained y file-based.

Algoritmo SHA256

Para generar el hash de los mensajes hemos decidido utilizar el algoritmo SHA256.
La implementación ha sido realizada a través de las librerías Python `hashlib` (`sha256`) y `hmac`. Hemos utilizados, con los cambios necesarios, el algoritmo utilizado en el PAI1.

JSON

Hemos utilizado la librería Python `json`, para la serialización y deserialización e los datos enviados del client hasta el server.

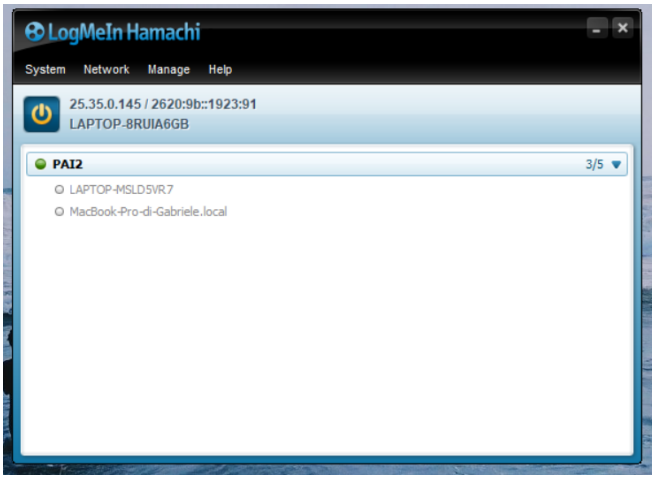
Logging

Hemos utilizado la librería Python `logging`, para

Socket
Hemos utilizado la librería Python `socket`, para implementar construir nuestra arquitectura cliente-servidor.

Uuid
Hemos utilizado la librería Python `uuid`, para la generación de los nonces.

Hamachi
Hamachi es un software de virtualización de redes que permite emular una red local (LAN) a los dispositivos conectados por WAN. Con Hamachi se puede generar una red local aunque los dispositivos se encuentren en distintos lugares repartidos por el mundo. Para ello, Hamachi hace uso de redes privadas virtuales (VPN). Hemos utilizado. Hamachi para simular el funcionamiento de nuestro programa en diferentes dispositivos.



SOLUCION
El proyecto se basa en un arquitectura cliente-servidor, en la que el client (el cliente) envía un mensaje al server (organización financiera). El mensaje que se envía entre el cliente y el servidor de la organización financiera tiene el siguiente formato: “Cuenta origen, Cuenta destino, Cantidad transferida”.

Input del sistema
El mensaje (Cuenta Origen, Cuenta Destino, Cantidad) a verificar la integridad en su transmisión, nombre del algoritmo que se usará para verificar la integridad, clave utilizada por el cliente y el servidor.

Output del sistema
Indicación en el cliente y servidor si se ha conservado la integridad o no se ha conservado. La salida podría ser presentada en una ventana al emisor del mensaje y en el servidor dejar constancia en un fichero de logs de los mensajes que no han llegado de forma íntegra.

Funcionamiento

El cliente envía un mensaje al servidor. El usuario solo tiene que inserir la cantidad de dinero que desea transferir; para el desarrollo de este proyecto hemos decidido tener la cuenta de origen y la cuenta de destino como constantes fijas.
A este mensaje se le añade un nonce, generado casualmente y luego se genera un código hash del mensaje + nonce.
A este punto, lo que se envía al servidor es un objeto en formato JSON en el que se encuentran el mensaje, el nonce y el hash.
El servidor primero verifica la integridad del mensaje, generando nuevamente el hash del mensaje recibido, para averiguar que los dos sean iguales. Si no lo son, hay una falla de integridad y el servidor lo escribe en el log.
Luego el servidor hace una verificación del nonce, para verificar que no se haya utilizado ya otra vez. Hacemos esta comprobación, almacenando en una base de datos todos los nonce. De esta manera, el servidor confronta el nonce del mensaje recibido con los que ya están en la base de datos. Si no lo encuentra significa que el control del mensaje ha ido bien; por otro lado, si el nonce ya está en la base de datos, significa que este código ya ha sido utilizado y puede tratarse de un replay attack; entonces el servidor rechaza la transacción y escribe un aviso en el log.
Al final, en el fichero de log se pueden ver todas la informaciones sobre la ejecución del programa.

PRUEBAS REALIZADAS

Ejecución sin fallas de integridad

Client & Server:

```
[SERVER]: Starting program...
[SERVER]: Database found!
[SERVER]: Establishing connection...
[SERVER]: Connection to database established successfully!
[SERVER]: STARTING SERVER - Server IP: 25.35.53.183 - Server Port: 3030
[SERVER]: Established connection with address: ('25.35.0.145', 61179)
[SERVER]: Message received. Processing message...
[SERVER]: Requested the transfer of: 200€ from SenderAccount: 1234 to ReceiverAccount: 5678
[SERVER]: Checking message authenticity...
[SERVER]: Message authenticity successfully verified. Authorizing operation...
```

```
clientsocket
C:\Users\matil\AppData\Local\Programs\Python\Python310\python.exe C:/Users/matil/PycharmProjects/SSII_PAI2/clientsocket.py
insert amount
200
b'The requested operation has been authorized and will be executed shortly'

Process finished with exit code 0
```

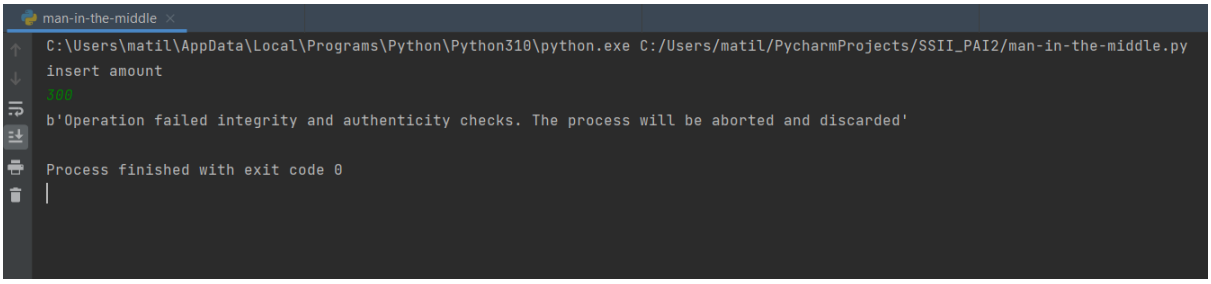
```
INFO 2022-03-24 16:23:38,081 - PROGRAM STARTED
INFO 2022-03-24 16:23:38,082 - Found a database in root dir. Trying to estabilish a connection..
INFO 2022-03-24 16:23:38,084 - Established connection to main database
INFO 2022-03-24 16:23:48,021 - Estabilished a connection with client on address: ('25.35.0.145', 61179)
INFO 2022-03-24 16:23:48,034 - Received data, processing message
INFO 2022-03-24 16:23:48,034 - Requested the transfer of: 200€ from SenderAccount: 1234+toReceiverAccount: 5678
INFO 2022-03-24 16:23:48,034 - Checking the autheticity of the message
INFO 2022-03-24 16:23:48,042 - Registered new nonce into database.
INFO 2022-03-24 16:23:48,042 - Message authenticity verified. The operation will be authorized
```

En este caso, el mensaje ha llegado al Servidor como esperado (sin fallas). La comprobación del nonce y del hash han salido correctamente.

Simulación ataque de man-in-the-middle

En este tipo de ataque, supongamos que el Atacante (Man-in-the-middle) no tiene acceso a la Clave de la funciòn HASH (en este caso SHA256) utilizada en el Servidor y el Cliente durante la conexión. El servidor, al llegar del mensaje, utiliza la clave para comprobar el hmac recibido junto al mensaje y el hmac generado al momento. Ya que los dos no son iguales porque han sido generados con claves diferentes, el servidor rechaza la transferencia.

Client:



Server:

```
[SERVER]: Starting program...
[SERVER]: Database found!
[SERVER]: Establishing connection...
[SERVER]: Connection to database established successfully!
[SERVER]: STARTING SERVER - Server IP: 25.35.53.183 - Server Port: 3030
[SERVER]: Established connection with address: ('25.35.0.145', 61291)
[SERVER]: Message received. Processing message...
[SERVER]: Requested the transfer of: 300€ from SenderAccount: 1234 to ReceiverAccount: 5678
[SERVER]: Checking message authenticity...
[SERVER]: WARNING! Failed hash check. Operation aborted
```

Log:

```
INFO 2022-03-24 16:35:14,569 - PROGRAM STARTED
INFO 2022-03-24 16:35:14,569 - Found a database in root dir. Trying to estabilish a connection..
INFO 2022-03-24 16:35:14,572 - Established connection to main database
INFO 2022-03-24 16:35:20,863 - Estabilished a connection with client on address: ('25.35.0.145', 61291)
INFO 2022-03-24 16:35:20,863 - Received data, processing message
INFO 2022-03-24 16:35:20,864 - Requested the transfer of: 300€ from SenderAccount: 1234+toReceiverAccount: 5678
INFO 2022-03-24 16:35:20,864 - Checking the autheticity of the message
ERROR 2022-03-24 16:35:20,864 - Failed hash check. Operation will be aborted
```

Simulación replay-attack

En este caso, el Atacante intercepta el mensaje del cliente durante la comunicación y no lo edita. El mensaje original del cliente llega al servidor y el nonce es almacenado en la base de datos del servidor y ejecutado con éxito. Después, el man in the middle intenta enviar el mismo mensaje, el servidor comprueba el nonce recibido y dado que ya lo tiene almacenado el la base de datos, buscándolo a través de una query, lo encuentra y por lo tanto rechaza la transferencia.

Server:

```
[SERVER]: Starting program...
[SERVER]: Database found!
[SERVER]: Establishing connection...
[SERVER]: Connection to database established successfully!
[SERVER]: STARTING SERVER - Server IP: 25.35.53.183 - Server Port: 3030
[SERVER]: Established connection with address: ('25.35.0.145', 61388)
[SERVER]: Message received. Processing message...
[SERVER]: Requested the transfer of: 500€ from SenderAccount: 1234 to ReceiverAccount: 5678
[SERVER]: Checking message authenticity...
[SERVER]: WARNING! Failed nonce check. Operation aborted
```

Client (Codigo) & database

	nonce
	Filtro
1	fc4e5bab-368b-4902-b475-7af282378711
2	6e0fd210-f9b0-4e08-bc55-88c5807c70f7
3	68e7d827-46af-45c1-9a69-dd01dcf310ce
4	418391c1-01fb-4ea2-91d8-f139ab95fbf6
5	31175ac0-6c77-4244-9ee8-b5c12c65228c
6	3c68a0d0-bf72-4fb3-9e77-d403e38aec8f
7	3575e764-f40e-4927-a7b0-069565924cb6

replayattack.py

```
print('insert amount')

sender = "1234"
receiver = "5678"
amount = input()  # str type
nonce = "3575e764-f40e-4927-a7b0-069565924cb6"  # using a NONCE that has already been used
message = sender + receiver + amount + nonce
hashed_message = hashfunction.hash_msg(message)


mydict = {
    "sender": sender,
    "receiver": receiver,
    "amount": amount,
    "nonce": nonce,
    "hmac": hashed_message}

serialized = json.dumps(mydict)

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(bytes(serialized, "utf-8"))
    data = s.recv(1024)
    print(data)
```

Log:

```
INFO 2022-03-24 16:57:57,638 - PROGRAM STARTED
INFO 2022-03-24 16:57:57,639 - Found a database in root dir. Trying to estabilish a connection..
INFO 2022-03-24 16:57:57,640 - Established connection to main database
INFO 2022-03-24 16:58:04,198 - Estabilished a connection with client on address: ('25.35.0.145', 61388)
INFO 2022-03-24 16:58:04,199 - Received data, processing message
INFO 2022-03-24 16:58:04,199 - Requested the transfer of: 500€ from SenderAccount: 1234+toReceiverAccount: 5678
INFO 2022-03-24 16:58:04,199 - Checking the autheticity of the message
ERROR 2022-03-24 16:58:04,201 - Failed nonce check. Operation will be aborted
```

ESTABLECIMIENTO DE CLAVES

Para que el cliente y servidor tengan la misma clave para hacer la comprobación de la integridad ha sido aplicado el siguiente razonamiento.
La criptografía simétrica es más insegura ya que el hecho de pasar la clave es una gran vulnerabilidad, pero se puede cifrar y descifrar en menor tiempo del que tarda la criptografía asimétrica, que es el principal inconveniente y es la razón por la que existe la criptografía híbrida.
Los sistemas de criptografía asimétricos pueden ser combinados con procesos simétricos. En este caso, las claves se comparten primero con criptografía asimétrica, pero la siguiente comunicación es criptografía de manera simétrica. Este sistema de criptografía híbrido se utiliza cuando los usuarios quieren tener la

velocidad de la criptografía simétrica y al mismo tiempo la seguridad de la criptografía asimétrica.

Criptografía híbrida

Este sistema es la unión de las ventajas de los dos, debemos de partir que el problema de ambos sistemas criptográficos es que el simétrico es inseguro y el asimétrico es lento.

El proceso para usar un sistema criptográfico híbrido es el siguiente (para enviar un archivo):

- Generar una clave pública y otra privada (en el receptor).
- Cifrar un archivo de forma síncrona.
- El receptor nos envía su clave pública.
- Ciframos la clave que hemos usado para encriptar el archivo con la clave pública del receptor.
- Enviamos el archivo cifrado (síncronamente) y la clave del archivo cifrada (asíncronamente y solo puede ver el receptor).