PAI 2. VERIFICADORES DE INTEGRIDAD EN LA TRANSMISIÓN PUNTO-PUNTO PARA ENTIDAD FINANCIERA

Seguridad en Sistemas Informáticos y en Internet

Grupo 5

Jesús Aparicio Ortiz
David Brincau Cano
Víctor Javier Granero Gil

Índice

Resumen Ejecutivo

Se ha realizado un verificador de la integridad de los mensajes para el cliente y el servidor, además de simular una serie de ataques del tipo Man in the Middle (modificación del mensaje original y reply attack).

Tecnologías Utilizadas

A continuación se detallan las tecnologías utilizadas en el desarrollo de los verificadores:

Nodejs

Se elige NodeJs para el desarrollo del verificador, ya que es más rápido en términos de ejecución que Python y Java.

Entre las librerías utilizadas en NodeJS caben destacar las siguientes:

- ws: Librería utilizada para crear sockets cliente y servidor.
- Crypto: Librería utilizada para crear los hashes de los archivos, además de otras funciones útiles para el proyecto.
- Math: Librería utilizada para generar números aleatorios para poder corromper archivos y generar fallos en la ejecución.

Solución

Con fin de asegurar la integridad de los mensajes del cliente y el servidor en las transmisiones, cada mensaje será acompañado de un nonce para evitar ataques de reply y con un hmac para evitar posibles ataques de man in the middle.

El nonce será generado para cada operación que se quiera realizar, tanto el cliente como el servidor generarán nonces para sus mensajes.

Para generar el hmac el cliente y el servidor necesitan de una clave secreta. No contemplamos el intercambio de esta clave en el código ya que consideramos que la entidad financiera posee un protocolo seguro para el reparto de dichas claves, más adelante se detalla cuál podría ser dicho protocolo.

Para realizar esto, hemos creado un servidor y un cliente gracias a nodejs. Los scripts creados por tanto son:

SERVIDOR (server.js)

Al ejecutarlo, se solicita por una entrada en la consola la clave secreta previamente mencionada:

```
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

rl.question('Escriba la clave simétrica\n', function (secretKey) {
    secret = secretKey;
    rl.close();
});
```

Fig 1. Inputs del servidor

Tras esto, se genera el servidor que queda a la espera de distintas peticiones de clientes, y tras recibir el mensaje en formato de objeto, hace varias comprobaciones:

- Comprueba que el nonce no ha sido utilizado previamente, es decir, que ese mensaje no ha sido enviado más de una vez seguida (para lo que guarda los nonces en un array)
- Obtiene el mensaje y calcula un nuevo HMAC para comprobar que es el mismo que el enviado por el cliente para comprobar que no ha sido modificado ni interceptada la clave

Después de las comprobaciones, envía al cliente todo lo que el propio servidor ha sacado en conclusión con un nuevo nonce y hmac.

```
ws.on('message', function incoming(message) {
    objectReceived = JSON.parse(message);

    let object2send = {
        message: "",
        hmac: "",
        hashType: objectReceived.hashType,
        nonce: createNonce()
    };


    if (nonces.includes(objectReceived.nonce)) {
        createReport("serverLog.txt", "El mensaje recibido es una respuesta a un ataque MiTM\n"+message+"\n");
        object2send.message = "Nonce already used"
    } else {
        nonces.push(objectReceived.nonce);
        if (createHmac(objectReceived.message, objectReceived.nonce, objectReceived.hashType) === objectReceived.hmac) {
            object2send.message = "OK"
        } else {
            object2send.message = "HMAC incorrecto"
        }
    }
    createReport("serverLog.txt", object2send.message+"\n"+message+"\n");
    console.log(object2send.message);
    object2send.hmac = createHmac(object2send.message, object2send.nonce, object2send.hashType);
    ws.send(JSON.stringify(object2send));
});
```

Fig 2. Verificación de la integridad de los datos enviados por el cliente al servidor

CLIENTE(client.js)

Al ejecutarlo, se solicitan varias cosas al usuario:

- Tipo de hash que se va a utilizar
- La clave secreta
- Mensaje (cuentaOrigen cuentaDestino cantidad)
- Tipo de ataque que se desea simular:
  - ➢ Ninguno
  - ➢ Reply
  - ➢ Modificación de mensaje
  - ➢ Modificación de mensaje con creación de HMAC

```
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

rl.question('¿Qué función hash quiere utilizar?\n0 => sha256\n1 => sha512\n2 => sha384\n', function (hashInputI) {
    hashInput = hashInputI >= 0 && hashInputI <= 2 ? hashInputI : 0;
    rl.question('Escriba la clave simétrica\n', function (secretKey) {
        secret = secretKey;
        rl.question('Introduzca los campos Cuenta Origen, Cuenta Destino, Cantidad, separados por espacios:\n', function (message) {
            sendingMessage = message;
            rl.question('Desea simular algún tipo de ataque:\n0 => Ninguno\n1 => Reply\n2 => MiTM (modificación de mensaje)\n
3 => MiTM (modificación de mensaje y de HMAC)\n', function (aSim) {
                attackSimulation = aSim;
                rl.close();
            });
        });
    });
});
```

Fig 3. Inputs del cliente

Tras esto se abre la conexión de un cliente con el servidor, donde se simulan los distintos tipos de ataques o ninguno, según lo seleccionado.

```
object2send.nonce = createNonce();
object2send.hmac = createHmac(object2send.message, object2send.nonce, secret); // HMAC created by the client
switch (attackSimulation) {
    // No attack
    case "0":
        createReport("clientLog.txt", "Mensaje enviado sin ataques\n"+JSON.stringify(object2send)+"\n");
        break;
    // MiTM attack: Reply attack
    case "1":
        createReport("clientLog.txt", "Mensaje simulando un ataque de reply\n"+JSON.stringify(object2send)+"\n");
        socket.send(JSON.stringify(object2send)); // Message resent (reply)
        break;
    // MiTM attack: message modification
    case "2":
        var oldObject = object2send;
        object2send.message += "0"; // Message modified by the Man In The Middle
        createReport("clientLog.txt", "Mensaje simulando un ataque de modificación de mensaje\nEnvio Original:\n"+JSON.stringify(oldObject)+"\n
Envio del Man In The Middle:\n"+JSON.stringify(object2send)+"\n");
        break;
    // MiTM attack: message modification with new HMAC
    case "3":
        var messageSplit = object2send.message.split(" ");
        var oldObject = object2send;
        object2send.message = messageSplit[0] + ' 3545331 ' + messageSplit[2]; // Message modified by the Man In The Middle
        object2send.hmac = createHmac(object2send.message, object2send.nonce, "secretomalcreadoporelmaninthemiddle"); // HMAC created by the Man In The Middle
        createReport("clientLog.txt", "Mensaje simulando un ataque de modificación de mensaje con intento de creación de HMAC\nEnvio Original:\n"+JSON.stringify(
oldObject)+"\nEnvio del Man In The Middle:\n"+JSON.stringify(object2send)+"\n");
        break;
    default:
        console.log("No ha elegido ninguna opción correcta")
        process.exit(1);
}
socket.send(JSON.stringify(object2send));
```

Fig 4. Envío del mensaje del cliente

Finalmente, el cliente recibe una respuesta por parte del servidor y además, almacena el Nonce generado por el servidor si no ha sido utilizado previamente, es decir, si no ha habido ataque de reply de vuelta del servidor al cliente. También se verifica que el mensaje tiene un HMAC correcto, para evitar modificaciones del mensaje y mensajes falsos del servidor al cliente (por ejemplo que un MiTM envíe un mensaje de que todo es correcto si ha habido un fallo)

```
socket.addEventListener('message', function (event) {
        objectReceived = JSON.parse(event.data);
        console.log('\nMensaje del servidor:\n', objectReceived);

        if (nonces.includes(objectReceived.nonce)) {
            createReport("clientLog.txt", "El mensaje del servidor al cliente ya ha sido recibido previamente (nonce repetido)\n");
            console.log("\nEl mensaje del servidor al cliente ya ha sido recibido previamente (nonce repetido)");
        } else {
            if (objectReceived.hmac === createHmac(objectReceived.message, objectReceived.nonce, secret)) {
                createReport("clientLog.txt", "El mensaje del servidor al cliente ha sido recibido correctamente\n");
                console.log("\nEl mensaje del servidor al cliente ha sido recibido correctamente");
                nonces.push(objectReceived.nonce);
            } else {
                createReport("clientLog.txt", "El mensaje del servidor al cliente ha sido recibido incorrectamente\n");
                console.log("\nEl mensaje del servidor al cliente ha sido recibido incorrectamente");
            }
        }
        createReport("clientLog.txt", "Mensaje del servidor:\n"+JSON.stringify(objectReceived)+"\n");
        socket.close();
    });
```

Fig 5. Verificación de la integridad del mensaje del servidor por parte del cliente

Además de todo esto, tanto el cliente como el servidor van generando reportes a medida que ocurren los diferentes procesos a lo largo de los mismos.

```
// Function to create a report
function createReport(reportFile, content) {
    reportFile = "./Reports/" + reportFile;
    fs.appendFile(reportFile, "\n"+content, (err) => {
        if (err) {
            console.log(err);
        } else {
            console.log("Report created");
        }
    });
}
```

Fig 6. Generación de logs

Reports

Se generan dos archivos de reporte donde se deja constancia de los mensajes enviados por el cliente y los recibidos por el servidor.
El reporte generado en el cliente guardará el mensaje que el cliente ha querido enviar, el ataque que se ha querido simular en dicha conexión y el mensaje que recibe el cliente de parte del servidor.

CLIENT LOGS

Mensaje enviado sin ataques
{"message":"123456 897564
300","hmac":"3ebf9dbd1798341181f9d7ff009e605804c8659a57ab53334bf2c0a0b
5df8daa","hashType":"0","nonce":"61c52cc6389bf899f9c2b124ad3c7b1ac8d6fa1af
1acfed488904d143296486101dda6d9ba15a87748705239440973715ce1f6a24356
30f196e904e29547b4e51913000748a4070c83f199463ff7614720d22b4ed1e155c4
812a867f106e0530780e96f84b97189be3f079119fa9cea8eeec547882ff7aaeae2b9
87442216d34"}

El mensaje del servidor al cliente ha sido recibido correctamente

Mensaje del servidor:
{"message":"La operación se ha realizado
correctamente","nonce":"dcd3231f2750888495e84b513c10bd43382c3b680665c5f
1f3fa4e301625249e8e370b0e6af72fbe19803dbe447e24b030390dc043ff5bdabe12
506fcc37335395e0f11861e8dac2060e1e0e6708e7f5ce6b9469302bd7e059de3b17
c7fa518bb071b714304e0ae1d1beed7c29350893237c56b5fc404b0a4a1fd856943
cbcc0","hmac":"07eefb3a2397092a8a7df4ba3e15939b3f470eef2b191bcda0c35c7
550d72e71","hashType":"0"}

El reporte generado en el servidor almacena un mensaje que detalla si el mensaje recibido es correcto o ha sufrido algún ataque:

SERVER LOGS

La operación se ha realizado correctamente
{"message":"123456 897564
300","hmac":"3ebf9dbd1798341181f9d7ff009e605804c8659a57ab53334bf2c0a0b
5df8daa","hashType":"0","nonce":"61c52cc6389bf899f9c2b124ad3c7b1ac8d6fa1af
1acfed488904d143296486101dda6d9ba15a87748705239440973715ce1f6a24356
30f196e904e29547b4e51913000748a4070c83f199463ff7614720d22b4ed1e155c4
812a867f106e0530780e96f84b97189be3f079119fa9cea8eeec547882ff7aaeae2b9
87442216d34"}

Intercambio de clave simétrica cliente/servidor

Para el intercambio de claves entre el cliente y el servidor se recomienda realizar un envío sms al dispositivo móvil del cliente con la clave simétrica, de este modo el servidor almacena dicha clave que se envía asociada al cliente y dicho cliente tiene en su poder del mismo modo la clave simétrica. La eficiencia de esta solución es alta, ya que únicamente se necesita un sistema de envío de sms al cliente y asociar la clave enviada al cliente correspondiente.

Conclusión

Finalmente, podemos decir que hemos creado una conexión cliente servidor en la que se asegura la integridad, haciendo uso de sockets. Además podemos señalar tres rasgos importantes que aprendemos de este proyecto:

- El primero es el uso de nonces a la hora de realizar hmac, ya que pueden ser generados de forma sencilla y tan solo con almacenarlos cada vez que se reciben, tanto en servidor como en cliente, se asegura fácilmente la robustez del sistema ante ataques de tipo reply.
- En segundo lugar, señalar la importancia del uso de hmac y nonces. Tanto en la petición al servidor como en la respuesta del servidor ya que de nada nos serviría asegurar la integridad en el envío si en la respuesta no se comprueba. Esto podría aprovecharlo un atacante man in the middle que capture las respuestas del servidor e intentase que el cliente realizara una operación varias veces enviando una respuesta de error al cliente cuando realmente sí se ha realizado.
- Finalmente, es muy importante asegurar que la clave simétrica solo la conocen el cliente y el servidor ya que de esta forma aseguramos que aunque las conexiones hayan sido capturadas por algún atacante, este no podrá interactuar con el servidor de forma exitosa si no posee dicha clave.

Manual de usuario

En primer lugar, para poder ejecutar la solución debe tener instalado NodeJS, el cual permitirá ejecutar código javascript fuera del navegador. Para ello puede visitar la página NodeJS donde podrá instalar la versión adecuada para su equipo. Seguidamente, debe instalar las dependencias necesarias, lo cual lo puede realizar con el siguiente comando:
"**npm install**"

A continuación, para ejecutar la solución debe realizar dos comandos, uno para arrancar el cliente que envía el mensaje y otro para el servidor que lo recibe:
"**node server.js**"

Tras ejecutar el primer comando aparecerán una serie de preguntas por consola, las cuales servirán a modo de configuración para el cliente:

1. Clave simétrica.

"**node client.js**"
Tras ejecutar el primer comando aparecerán una serie de preguntas por consola, las cuales servirán a modo de configuración para el cliente:

1. Función de hash a utilizar entre *[sha256,sha512,sha384]*, eligiendo con un número del 0 al 2 respectivamente.
2. Clave simétrica.
3. Introducir los campos Cuenta Origen, Cuenta Destino, Cantidad, separados por espacios, siguiendo el formato: "XXX YYY ZZZ"
4. Indicar si se quiere realizar algún tipo de ataque, eligiendo una de las siguientes opciones [*Ninguno, Reply, MiTM (modificación de mensaje), MiTM (modificación de mensaje y de HMAC)*], eligiendo con un número del 0 al 3 respectivamente.

Bibliografía

NodeJS vs Python
https://kinsta.com/es/blog/nodejs-vs-python/#velocidad

Javascript Crypto
https://nodejs.org/api/crypto.html

WS Node Library
https://www.npmjs.com/package/ws