**Izvještaj laboratorijskih vježbi**

Rato Kuzmanić, 250

|  |  |
| --- | --- |
| **Vježba:** | 6. Securing end-2-end communication |
| **Grupa:** | Grupa 2 |
| **Rješenje:** | U najširem smislu, dva objekta nisu jednaka čim se razlikuju za barem jedan bit, stoga i dva autentikacijska taga (bilo u heksadekadskom tekstualnom obliku ili u obliku niza okteta) nisu jednaka čim se razlikuju za jedan znak ili oktet. Sigurna usporedba lokalno izračunatog autentikacijskog taga i onog koji je došao s porukom se zasniva na neutralizaciji te vremenske optimizacije koju izvršava kompajler, odnosno interpreter. Kompjaler i interpreter zaustave izvršavanje usporedbe jednakosti čim naiđu na prvu sadržajnu nejednakost, a kako bismo izbjegli *timing* napade na uspoređivanje autentikacijskog taga i onog pristiglog s porukom izvršavamo usporedbu za cijeli tag, a tek potom vraćamo rezultat usporedbe čime osiguravamo da vrijeme izvršavanja usporedbe ne ovisi o sadržaju već isključivo o duljini taga. Pritom je važno na samome početku provjeriti jesu li lokalni i pristigli tag jednake duljine, te odmah obznaniti nejednakost ukoliko nisu. |

**CBC and HMAC**

securityActions.js

import {

KEY\_GENERATE,

KEY\_GENERATED,

KEY\_DELETE

} from './actionTypes.js'

import pbkdf2 from '../../services/security/pbkdf2.js'

export const generateKey = payload => dispatch => {

dispatch({

type: KEY\_GENERATE,

payload: { id: payload.id }

})

pbkdf2({ secret: payload.secret, salt: payload.id, size: 64 })

.then(key => dispatch({

type: KEY\_GENERATED,

payload: {

id: payload.id,

key: key

}

}))

.catch(error => dispatch({

type: KEY\_GENERATED,

payload: {

id: payload.id,

error: error

},

error: true

}))

}

export const deleteKey = id => ({

type: KEY\_DELETE,

payload: { id }

})

handleMsgOut.js

const crypto = require('crypto')

import { Server, Constants } from 'config'

import serverAPI from 'app/services/server-api/ServerAPI.js'

import { msgSent } from 'app/redux/actions/clientActions.js'

import { loadKey, splitKey } from './utils.js'

import CryptoProvider from '../../../services/security/CryptoProvider.js'

import { randomBytes } from 'crypto'

import { hash } from '../../../services/security/hmac.js'

const { MsgType } = Constants

export default ({ getState, dispatch }, next, action) => {

const { meta: { wrapped } } = action

if (wrapped) return next(action)

const {

client: { nickname, id },

credentials

} = getState()

const key = loadKey(id, credentials)

const message = {

type: MsgType.BROADCAST,

id,

nickname,

timestamp: Date.now()

}

if(key) {

const { symmetricKey, hmacKey } = splitKey(key);

const { ciphertext, iv } = CryptoProvider.encrypt('CBC', {

key: symmetricKey,

iv: randomBytes(16),

plaintext: action.payload

})

Object.assign(message, { content: ciphertext, iv });

const authTag = hash({ key: hmacKey, message });

Object.assign(message, { authTag });

}

else {

Object.assign(message, { content: action.payload });

}

serverAPI.send(message).then(

dispatch(msgSent(Object.assign({}, message, { content: action.payload })))

)

}

handleMsgIn.js

import { serverMsg } from 'app/redux/actions/serverActions.js'

import { JSONparse } from 'app/utils/safeJSON.js'

import { clientError } from 'app/redux/actions/clientActions.js'

import { loadKey, splitKey } from './utils.js'

import CryptoProvider from '../../../services/security/CryptoProvider.js'

import { hash, isValidHash } from '../../../services/security/hmac.js'

import { isReplayAttack } from '../../../services/security/replay.js'

export default ({ getState, dispatch }, next, action) => {

const { meta: { serialized } } = action

if (!serialized) return next(action)

let message = JSONparse(action.payload)

if (Object.is(message, undefined)) {

return dispatch(clientError(`JSON.parse error: ${data}`))

}

if (message.id) {

const { credentials } = getState()

const key = loadKey(message.id, credentials)

if (key) {

const { symmetricKey, hmacKey } = splitKey(key);

const messageWithoutAnAuthTag = Object.assign({}, message, { authTag: undefined });

if(isReplayAttack(message.timestamp)) {

message.content = 'REPLAY ATTACK'

}

else {

if(isValidHash({ hash: message.authTag, key: hmacKey, message: messageWithoutAnAuthTag })) {

const { plaintext } = CryptoProvider.decrypt('CBC', {

key: symmetricKey,

iv: Buffer.from(message.iv, 'hex'),

ciphertext: message.content

});

message.content = plaintext;

}

else {

message.content = 'AUTHENTICATION FAILURE'

}

}

}

}

dispatch(serverMsg(message))

}

utils.js

function loadKey(id, state) {

if (!(id in state)) return undefined

const { symmetric: { key } } = state[id]

return key

}

function splitKey(key) {

const symmetricKey = Buffer.alloc(32);

const hmacKey = Buffer.alloc(32);

key.copy(symmetricKey, 0, 0, key.byteLength / 2);

key.copy(hmacKey, 0, key.byteLength / 2, 64);

return { symmetricKey, hmacKey };

}

export { loadKey, splitKey }

hmac.js

const crypto = require('crypto')

const algorithm = 'sha256';

const hash = ({ key, message }) => {

const objectToHash = typeof(message) === 'string'

? message

: JSON.stringify(message);

const hmac = crypto.createHmac(algorithm, key);

hmac.update(objectToHash);

return hmac.digest().toString('hex').slice(0, 32);

}

const isValidHash = ({ hash, key, message }) => {

const digest = hash({ key, message });

if(digest.length !== hash.length) {

return false;

}

let isValid = true;

for(let i = 0; i < digest.length; i++) {

if(digest[i] !== hash[i]) {

isValid = false;

}

}

return isValid;

}

export { hash, isValidHash }

replay.js

const messageValidityInSeconds = 1;

const isReplayAttack = (messageSendTime) =>

messageSendTime + messageValidityInSeconds\*1000 <= Date.now();

export { isReplayAttack }

**GCM**

securityActions.js

import {

KEY\_GENERATE,

KEY\_GENERATED,

KEY\_DELETE

} from './actionTypes.js'

import pbkdf2 from '../../services/security/pbkdf2.js'

export const generateKey = payload => dispatch => {

dispatch({

type: KEY\_GENERATE,

payload: { id: payload.id }

})

pbkdf2({ secret: payload.secret, salt: payload.id })

.then(key => dispatch({

type: KEY\_GENERATED,

payload: {

id: payload.id,

key: key

}

}))

.catch(error => dispatch({

type: KEY\_GENERATED,

payload: {

id: payload.id,

error: error

},

error: true

}))

}

export const deleteKey = id => ({

type: KEY\_DELETE,

payload: { id }

})

handleMsgOut.js

const crypto = require('crypto')

import { Server, Constants } from 'config'

import serverAPI from 'app/services/server-api/ServerAPI.js'

import { msgSent } from 'app/redux/actions/clientActions.js'

import { loadKey } from './utils.js'

import CryptoProvider from '../../../services/security/CryptoProvider.js'

import { randomBytes } from 'crypto'

const { MsgType } = Constants

export default ({ getState, dispatch }, next, action) => {

const { meta: { wrapped } } = action

if (wrapped) return next(action)

const {

client: { nickname, id },

credentials

} = getState()

const key = loadKey(id, credentials)

const message = {

type: MsgType.BROADCAST,

id,

nickname,

timestamp: Date.now()

}

if(key) {

const { ciphertext, iv, tag } = CryptoProvider.encrypt('GCM', {

key,

iv: randomBytes(12),

plaintext: action.payload

});

Object.assign(message, { content: ciphertext, iv, tag });

}

else {

Object.assign(message, { content: action.payload });

}

serverAPI.send(message).then(

dispatch(msgSent(Object.assign({}, message, { content: action.payload })))

)

}

handleMsgIn.js

import { serverMsg } from 'app/redux/actions/serverActions.js'

import { JSONparse } from 'app/utils/safeJSON.js'

import { clientError } from 'app/redux/actions/clientActions.js'

import { loadKey } from './utils.js'

import CryptoProvider from '../../../services/security/CryptoProvider.js'

import { isReplayAttack } from '../../../services/security/replay.js'

export default ({ getState, dispatch }, next, action) => {

const { meta: { serialized } } = action

if (!serialized) return next(action)

let message = JSONparse(action.payload)

if (Object.is(message, undefined)) {

return dispatch(clientError(`JSON.parse error: ${data}`))

}

if (message.id) {

const { credentials } = getState()

const key = loadKey(message.id, credentials)

if (key) {

if(isReplayAttack(message.timestamp)) {

message.content = 'REPLAY ATTACK'

}

else {

try {

const msgContent = message.iv + message.content + message.tag;

const plaintext = CryptoProvider.decrypt('GCM', {

key,

msgContent

});

message.content = plaintext;

}

catch (e) {

message.content = 'AUTHENTICATION FAILURE'

}

}

}

}

dispatch(serverMsg(message))

}

replay.js

const messageValidityInSeconds = 1;

const isReplayAttack = (messageSendTime) =>

messageSendTime + messageValidityInSeconds\*1000 <= Date.now();

export { isReplayAttack }