**Izvještaj laboratorijskih vježbi**

Rato Kuzmanić, 250

|  |  |
| --- | --- |
| **Vježba:** | 3. CBC mode |
| **Grupa:** | Grupa 2 |
| **Rješenje:** | taco |

client.js

const fs = require('fs');

const http = require('http');

const xor = require('buffer-xor');

const pkcs7 = require('pkcs7');

const incrementIv = require('./utils');

const { subtract } = require('math-buffer');

const { prettyLogSuccess, prettyLogError } = require('./logger');

const { app, request: { get: getRequest, post: postRequest } } = require('./config');

const wordlist = fs.readFileSync('wordlist.txt').toString().split("\n");

getChallenge = () =>

new Promise((resolve, reject) => {

const request = http.request(getRequest, response => {

let data = '';

response.on('data', chunk => data += chunk);

response.on('end', () => resolve(JSON.parse(data)));

});

request.end();

});

getIvAndCiphertext = plaintext =>

new Promise((resolve, reject) => {

const data = JSON.stringify({ plaintext });

const request = http.request(postRequest, response => {

response.setEncoding('utf8');

response.on('data', data => resolve(JSON.parse(data)));

response.on('error', error => {

prettyLogError('Error on POST request', error);

reject(error);

});

});

request.write(data);

request.end();

});

async function getIncrementSize() {

const { iv: firstIv } = await getIvAndCiphertext('test');

const { iv: secondIv } = await getIvAndCiphertext('test');

const diff = subtract(Buffer.from(secondIv, 'hex'), Buffer.from(firstIv, 'hex'));

return parseInt(diff.toString('hex'));

}

isHit = (possibleCiphertextHit, challengeCiphertext) =>

possibleCiphertextHit.slice(0, app.ciphertextBlockSize) === challengeCiphertext;

(async () => {

const { iv, ciphertext } = await getChallenge();

const challengeIv = Buffer.from(iv, 'hex');

const incrementSize = await getIncrementSize();

const { iv: currentIv } = await getIvAndCiphertext('test');

let iterationIv = Buffer.from(currentIv, 'hex');

incrementIv(iterationIv, incrementSize);

for(let index in wordlist) {

const plaintext = Buffer.from(wordlist[index], 'utf8');

const paddedPlaintext = Buffer.from(pkcs7.pad(plaintext));

const payload = xor(xor(iterationIv, challengeIv), paddedPlaintext);

const { ciphertext: possibleCiphertextHit } = await getIvAndCiphertext(payload.toString('hex'));

if(isHit(possibleCiphertextHit, ciphertext)) {

prettyLogSuccess('Seeked word found', wordlist[index]);

break;

}

incrementIv(iterationIv, incrementSize);

}

})();

logger.js

const chalk = require('chalk');

String.prototype.addWhitespacePadding = function(numberOfWhitespaces = 8) {

return `${' '.repeat(numberOfWhitespaces)}${this}${' '.repeat(numberOfWhitespaces)}`;

}

logError = (title, error) => {

console.log(`\n${chalk.white.bgRed(title.addWhitespacePadding())}`);

console.log(`Details: ${error}\n`);

}

logSuccess = (title, details) => {

console.log(`\n${chalk.black.bgGreen(title.addWhitespacePadding())}`);

console.log(`Details: ${details}\n`);

}

module.exports = {

prettyLogError: logError,

prettyLogSuccess: logSuccess

}

utils.js

const MAX\_32\_INTEGER = (Math.pow(2, 32) - 1)

const incrementUInt32By = (bigint, addend=1, offset=12) => {

if (offset < 0) return

const current = bigint.readUInt32BE(offset)

const sum = current + addend

if (sum <= MAX\_32\_INTEGER) {

return bigint.writeUInt32BE(sum, offset)

}

const reminder = sum % (MAX\_32\_INTEGER + 1)

const carry = Math.floor(sum/MAX\_32\_INTEGER)

bigint.writeUInt32BE(reminder, offset)

incrementUInt32By(bigint, carry, offset - 4)

}

module.exports = incrementUInt32By

config.js

const app = {

ciphertextBlockSize: 32

};

const commonRequest = {

host: '10.0.0.6',

port: 80,

headers: {

'Content-Type': 'application/json'

}

};

const getRequest = {

...commonRequest,

path: '/cbc/iv/challenge',

method: 'GET'

};

const postRequest = {

...commonRequest,

path: '/cbc/iv',

method: 'POST'

};

module.exports = {

app: app,

request: {

get: getRequest,

post: postRequest

}

}