Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЕТ

к лабораторной работе № 4, 5

на тему

**РАЗРАБОТКА ЗАЩИЩЕННЫХ ПРИЛОЖЕНИЙ**

Выполнил:

студент гр. 153503

Бобко И.В.

Проверил:

Лещенко Е.А.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc157598705)

[2 Теоретические сведения 4](#_Toc157598706)

[3 Полученные результаты 5](#_Toc157598712)

[Выводы 7](#_Toc157598713)

[Приложение А (обязательное) Листинг кода 8](#_Toc157598714)

# **1 ЦЕЛЬ РАБОТЫ**

Цель лабораторной работы – познакомиться с концепцией ролевого управления доступом и способами защиты программного обеспечения от существующих угроз. Научиться разрабатывать приложения, которые используют ролевое управление доступом для разграничения полномочий пользователей. Получить навыки защиты разработанной программы от несанкционированного копирования и других угроз, которым может подвергаться программное обеспечение.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ** 

DoS атаки популярный вид атак, который заключается в перегрузке сети запросами. По сути это  [хакерская атака](https://ru.wikipedia.org/wiki/Хакерская_атака) на вычислительную систему с целью довести её до отказа, то есть создание таких условий, при которых добросовестные пользователи системы не могут получить доступ к предоставляемым системным ресурсам (серверам), либо этот доступ затруднён. Отказ «вражеской» системы может быть и шагом к овладению системой (если в нештатной ситуации ПО выдаёт какую-либо критическую информацию — например, версию, часть программного кода и т. д.). Но чаще это мера экономического давления: потеря простой службы, приносящей доход, счета от провайдера и меры по уходу от атаки ощутимо бьют «цель» по карману. В настоящее время DoS и DDoS-атаки наиболее популярны, так как позволяют довести до отказа практически любую систему, не оставляя юридически значимых улик.

Переполнение буфера (англ. *Buffer Overflow*) — явление, возникающее, когда компьютерная программа записывает данные за пределами выделенного в памяти буфера.

Переполнение буфера может вызывать аварийное завершение или зависание программы, ведущее к *отказу обслуживания* (denial of service, DoS). Отдельные виды переполнений, например переполнение в стековом кадре, позволяют злоумышленнику загрузить и выполнить произвольный машинный код от имени программы и с правами учетной записи, от которой она выполняется.

Принцип наименьших привилегий, также известный как принцип минимальных привилегий (англ. *Principle of least privilege*) или просто минимальные привилегии, в информационной безопасности, информатике и других областях — принцип организации доступа к ресурсам, когда в тот или иной уровень абстракции от вычислительной среды, каждый модуль (такой, как процесс, пользователь или программа, которые мы рассматриваем) должны иметь доступ к такой информации и ресурсам, которые минимально необходимы для успешного выполнения его рабочей цели.

# **3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

В ходе выполнения данной лабораторной работы было создано приложение, в котором можно размещать статьи. Для начала пользования приложение пользователю необходимо авторизироваться (рисунок 1).

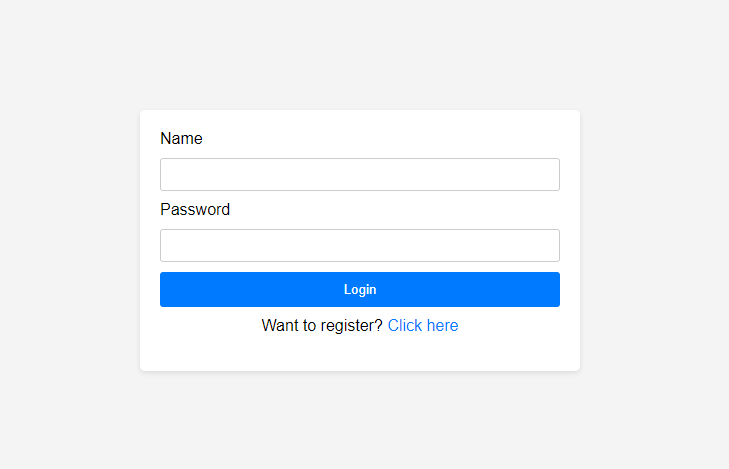


Рисунок 1 – Страница авторизации

Если же у пользователя нет учётной записи, он может зарегистрироваться (рисунок 2).

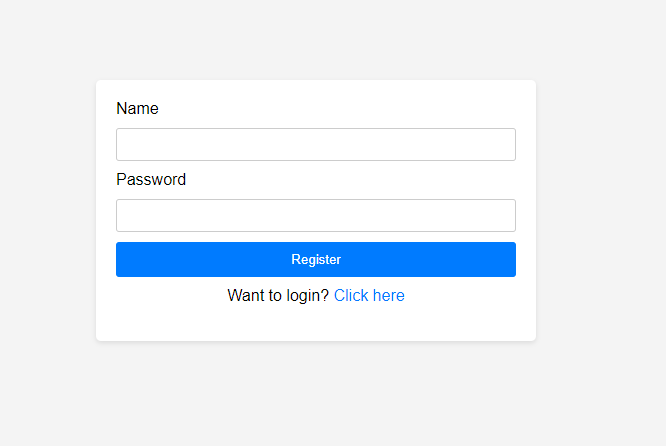


Рисунок 2 – Страница регистрации

После авторизации пользователь может добавить статью либо найти в поиске интересующую его статью. Также у пользователя-админа есть возможность удалять статьи (рисунок 3)..

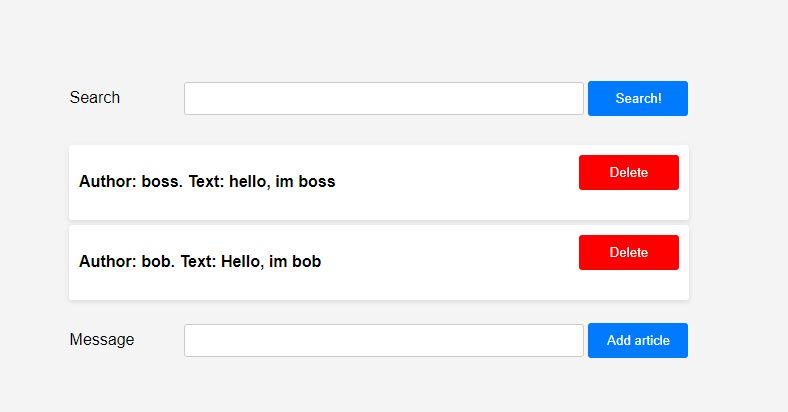


Рисунок 3 – Страница со статьями

Для предотвращения атаки, известной как «переполнение буфера», было установлено ограничение на 10^6 символов для вводимых значений. Таким образом, если пользователь вводит очень длинное значение, оно не будет помещено в буфер.

Для защиты от потенциальной атаки типа DoS (отказ в обслуживании) была добавлена искусственная задержка в 0.2 секунды при нажатии на кнопки, отправляющие формы. Для обычного пользователя такая задержка не будет заметна, но она станет помехой для злоумышленников.

Приложение взаимодействует с базой данных с помощью raw-SQL запросов с использованием библиотеки PgClient. Для защиты от SQL-инъекций было использована технология передачи данных в SQL запрос через параметры, чтобы Postgres обрабатывал эти данные на своей стороне, что позволяет защититься от SQL-инъекции.

Программа-установщик для данного приложения имеет защиту по hash. Если как-то изменить установщик – его hash измениться и приложение дистрибутор не даст файл программы из-за другого hash. После успешной установки в установленное приложение вшита защита, которая не позволяет запускать это приложение на другом ПК.

# **ВЫВОДЫ**

В ходе выполнения данной лабораторной работы было реализовано десктоп приложение, реализующее простой функционал авторизации и взаимодействия с базой данных, а также предусматривающее защиту от атак, основанных на пользовательских привилегиях, от DoS-атак, атак переполнения буфера и SQL-инъекций. Также создана программа-установщик, которая проверяет образ приложение на отсутствие модификаций, а также подтверждает подлинность приложения цифровой подписью.

# **ПРИЛОЖЕНИЕ А (обязательное) Листинг кода**

Листинг **–** Файл *distributor.js*

const http = require('http');

const crypto = require('crypto');

const fs = require('fs');

// some comment

const installerHash = 'ccc6a983128b7b01c0e165f1431e8398';

function hash(s) {

return crypto.createHash('md5').update(s).digest('hex');

}

const s = http.createServer((req, res) => {

const providedInstallerHash = req.headers['installer-hash'];

if (!providedInstallerHash || providedInstallerHash !== installerHash) {

res.writeHead(400);

res.end();

return;

}

const systemInfo = req.headers['system-info'];

if (!systemInfo) {

res.writeHead(400);

res.end();

return;

}

const code = `

const crypto\_ = require('crypto');

const os\_ = require('os');

function getSystemInfo\_() {

return os\_.arch() + os\_.type() + os\_.machine() + os\_.platform() + os\_.cpus()[0].model;

}

function hash\_(s) {

return crypto\_.createHash('md5').update(s).digest('hex');

}

if (hash\_(getSystemInfo\_().trim()) !== '${hash(systemInfo)}') {

console.error('INVALID SIGNATURE');

process.exit(1);

}

`;

const file = code + fs.readFileSync('./index.js').toString();

res.writeHead(200);

res.write(file);

res.end();

});

s.listen(3000);

Листинг **–** Файл *index.js*

const http = require('http');

const { Client } = require('pg');

const crypto = require('crypto');

const hostname = '127.0.0.1';

const port = 8080;

const handlers = [];

const MAX\_LENGTH = 1e6;

async function parseBody(req) {

if (req.method !== 'POST') {

return;

}

let body = '';

for await (const data of req) {

body += data;

if (body.length > MAX\_LENGTH) {

req.destroy();

}

}

req.body = body;

}

let loggedTimes = {};

const RPS = 100;

function rateLimit(req, res) {

const ip = req.socket.remoteAddress;

const times = loggedTimes[ip] ?? 0;

if (times > RPS) {

return res.destroy();

}

loggedTimes[ip] = times + 1;

}

setInterval(() => (loggedTimes = {}), 1000);

const server = http.createServer(async (req, res) => {

try {

res.setHeader('Access-Control-Allow-Origin', req?.headers?.origin ?? '\*');

res.setHeader('Access-Control-Request-Method', '\*');

res.setHeader('Access-Control-Allow-Methods', 'OPTIONS, GET');

res.setHeader('Access-Control-Allow-Headers', '\*');

if (req.method === 'OPTIONS') {

res.writeHead(204);

res.end();

return;

}

rateLimit(req, res);

req.parsedUrl = new URL(`http://${hostname}:${port}${req.url}`);

for (const handler of handlers) {

if (req.method === handler.method && req.parsedUrl.pathname === handler.route) {

await parseBody(req);

let result = handler.listener(req, res);

if (result instanceof Promise) {

result = await result;

}

return;

}

}

response(res, { message: 'Not found' }, 404);

} catch (e) {

console.error(e);

response(res, { message: 'Internal server error' }, 500);

}

});

function post(route, listener) {

handlers.push({

route,

method: 'POST',

listener,

});

}

function get(route, listener) {

handlers.push({

route,

method: 'GET',

listener,

});

}

function response(res, body, statusCode = 200) {

res.statusCode = statusCode;

res.write(typeof body === 'string' ? body : JSON.stringify(body));

res.end();

}

function listen() {

server.listen(port, hostname, () => {

console.log(`Server running at http://${hostname}:${port}/`);

});

}

function parseUrlBody(body) {

return Object.fromEntries(new URLSearchParams(body).entries());

}

const loginHtml = `

<html lang="en">

<head>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

margin: 0;

padding: 0;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

#login-form {

background-color: #fff;

padding: 20px;

border-radius: 5px;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

width: 400px;

}

label {

display: block;

margin-bottom: 10px;

}

input {

width: 100%;

padding: 8px;

margin-bottom: 10px;

border: 1px solid #ccc;

border-radius: 3px;

}

button {

width: 100%;

padding: 10px;

background-color: #007bff;

color: #fff;

border: none;

border-radius: 3px;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

p {

margin-top: 10px;

text-align: center;

}

a {

color: #007bff;

text-decoration: none;

}

a:hover {

text-decoration: underline;

}

</style>

</head>

<body>

<form id="login-form">

<label for="name">Name</label><input name="name" id="name"><br>

<label for="password">Password</label><input name="password" id="password" type="password"><br>

<button type="submit">Login</button>

<p id="register-link">Want to register? <a href="/register">Click here</a></p>

</form>

</body>

<script>

document.getElementById('login-form').addEventListener('submit', ev => {

ev.preventDefault();

const { name, password } = Object.fromEntries(new FormData(ev.target).entries());

document.cookie = \`Authorization=\${btoa(\`\${name}:\${password}\`)}\`;

window.location = '/';

});

</script>

</html>

`;

get('/login', (req, res) => {

response(res, loginHtml);

});

const registerHtml = `

<html lang="en">

<head>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

margin: 0;

padding: 0;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

form {

background-color: #fff;

padding: 20px;

border-radius: 5px;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

width: 400px;

}

label {

display: block;

margin-bottom: 10px;

}

input {

width: 100%;

padding: 8px;

margin-bottom: 10px;

border: 1px solid #ccc;

border-radius: 3px;

}

button {

width: 100%;

padding: 10px;

background-color: #007bff;

color: #fff;

border: none;

border-radius: 3px;

cursor: pointer;

}

button:hover {

background-color: #0056b3;

}

p {

margin-top: 10px;

text-align: center;

}

a {

color: #007bff;

text-decoration: none;

}

a:hover {

text-decoration: underline;

}

</style>

</head>

<body>

<form method="post">

<label for="name">Name</label><input name="name" id="name"><br>

<label for="password">Password</label><input name="password" id="password" type="password"><br>

<button type="submit">Register</button>

<p>Want to login? <a href="/login">Click here</a></p>

</form>

</body>

</html>

`;

const redirectScript = (path) => `setTimeout(() => window.location='${path}', 2000)`;

get('/register', (req, res) => {

response(res, registerHtml);

});

post('/register', async (req, res) => {

const user = parseUrlBody(req.body);

if (!user?.name || !user?.password) {

response(res, `<h1>Invalid form</h1><script>${redirectScript('/register')}</script>`);

return;

}

const { rows } = await pgClient.query(`SELECT \* FROM "user" WHERE "name" = $1`, [user.name]);

const dbUser = rows[0];

if (dbUser) {

response(res, `<h1>User with this name already exists</h1><script>${redirectScript('/register')}</script>`);

return;

}

const hashPass = crypto.createHash('md5').update(user.password).digest('hex');

await pgClient.query(`INSERT INTO "user"("name", "password", "role") VALUES ($1, $2, $3)`, [user.name, hashPass, "user"]);

return response(res, `<h1>Successfully registered</h1><script>${redirectScript('/login')}</script>`);

});

/\*\*

\* @type {Client}

\*/

let pgClient;

async function getAuthUser(authHeader) {

if (!authHeader) {

return;

}

const decoded = Buffer.from(authHeader, 'base64').toString();

if (!decoded) {

return;

}

const [name, pass] = decoded.split(':', 2);

if (!name) {

return undefined;

}

const res = await pgClient.query(`SELECT \* FROM "user" WHERE "name" = $1`, [name]);

const user = res.rows[0];

if (!user) {

return undefined;

}

const hashPass = crypto.createHash('md5').update(pass).digest('hex');

if (user.password !== hashPass) {

return undefined;

}

return user;

}

function escapeHtml(htmlStr) {

//return htmlStr;

return htmlStr.replace(/&/g, '&amp;')

.replace(/</g, '&lt;')

.replace(/>/g, '&gt;')

.replace(/"/g, '&quot;')

.replace(/'/g, '&#39;');

}

post('/delete', async (req, res) => {

const authHeader = req?.headers?.cookie?.split('Authorization=')[1];

const user = await getAuthUser(authHeader);

if (!user) {

return response(res, `<h1>User not logged in</h1><script>${redirectScript('/login')}</script>`);

}

if (user.role !== 'admin') {

return response(res, `<h1>Must be admin</h1><script>${redirectScript('/')}</script>`);

}

const { id } = parseUrlBody(req.body);

await pgClient.query(`DELETE FROM "article" WHERE "id" = $1`, [Number(id)]);

return response(res, `<h1>Article sucessfully deleted</h1><script>${redirectScript('/')}</script>`);

});

// ' UNION SELECT id, password, name from "user" WHERE 1=1 --

get('/', async (req, res) => {

const authHeader = req?.headers?.cookie?.split('Authorization=')[1];

const user = await getAuthUser(authHeader);

if (!user) {

return response(res, `<h1>User not logged in</h1><script>${redirectScript('/login')}</script>`);

}

// const q = req?.parsedUrl?.searchParams?.get('q');

// const { rows: articles } = await pgClient.query(`

// SELECT "article"."id" AS "id", "message", "name"

// FROM "article"

// INNER JOIN "user" ON "user"."id" = "article"."userId"

// ${q ? `WHERE "message" ILIKE $1` : ''}

// `, q ? [`${q}%`] : []);

const q = req?.parsedUrl?.searchParams?.get('q');

const { rows: articles } = await pgClient.query(`

SELECT "article"."id" AS "id", "message", "name"

FROM "article"

INNER JOIN "user" ON "user"."id" = "article"."userId"

${q ? `WHERE "message" ILIKE '${q}%'` : ''}

`);

return response(res, `

<html lang="en">

<head>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

margin: 0;

padding: 0;

display: flex;

flex-direction: column;

justify-content: center;

align-items: center;

height: 100vh;

}

form {

margin-bottom: 20px;

}

label {

display: inline-block;

width: 100px;

margin-right: 10px;

}

input[type="search"],

input[type="text"] {

width: 400px;

padding: 8px;

margin-bottom: 10px;

border: 1px solid #ccc;

border-radius: 3px;

}

button[type="submit"] {

padding: 10px;

width: 100px;

background-color: #007bff;

color: #fff;

border: none;

border-radius: 3px;

cursor: pointer;

}

button[type="submit"]:hover {

background-color: #0056b3;

}

ul {

list-style-type: none;

padding: 0;

margin: 0;

}

li {

display: flex;

justify-content: space-between;

align-items: center;

width: 600px;

padding: 10px;

font-weight: bold;

margin-bottom: 5px;

background-color: #fff;

border-radius: 3px;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

}

</style>

</head>

<body>

${articles?.length ? `

<form>

<label for="q">Search</label> <input type="search" id="q" name="q" />

<button type="submit">Search!</button>

</form>

<ul>

${articles.map(article => {

return `<li>Author: ${escapeHtml(article.name)}. Text: ${escapeHtml(article.message)} ${user.role === 'admin' ?

`<form method="post" action="/delete"><input type="hidden" name="id" value=${article.id}><button type="submit" style="background-color: #FF0000">Delete</button></form>` : ''}</li>`;

}).join('\n')}

</ul>` : 'No articles found'}

<br>

<form method="post">

<label for="message">Message</label> <input type="text" name="message" id="message">

<button type="submit">Add article</button>

</form>

</body>

</html>

`);

});

post('/', async (req, res) => {

const authHeader = req?.headers?.cookie?.split('Authorization=')[1];

const user = await getAuthUser(authHeader);

if (!user) {

return response(res, `<h1>User not logged in</h1><script>${redirectScript('/login')}</script>`);

}

const article = parseUrlBody(req.body);

if (!article?.message) {

return response(res, `<h1>No article name provided</h1><script>${redirectScript('/')}</script>`);

}

await pgClient.query(`INSERT INTO "article"("message", "userId") VALUES ($1, $2)`, [article.message, user.id]);

return response(res, `<h1>Article succesfully added</h1><script>${redirectScript('/')};</script>`);

});

async function main() {

pgClient = new Client({

user: 'postgres',

host: 'localhost',

database: 'aboba',

password: '',

port: 5432,

});

await pgClient.connect();

listen();

}

main();