Государственное бюджетное общеобразовательное учреждение города Москвы «Школа № 7»

ПРИЛОЖЕНИЕ ДЛЯ ПРОВЕРКИ НАЛИЧИЯ ПАРОЛЯ В СЛИТЫХ БАЗАХ ДАННЫХ И СЕРВИС ДЛЯ БЛОКИРОВКИ ТРЕКЕРОВ

Участник:
ученица 10 «Б» класса ГБОУ Школа № 7 Заславская Серафима Валерьевна

Руководитель:
педагог ГБОУ Школа № 7
Молотков Никита Александровича

Оглавление

1. Введение

1.1. Актуальность работы

В современном цифровом мире проблема защиты персональных данных приобретает критическое значение. Согласно статистике:

- **11+ миллиардов** учетных записей были скомпрометированы в различных утечках данных
- **81% пользователей** повторно используют пароли на разных сайтах
- **Более 70%** мобильных приложений собирают данные пользователей без их ведома
- **Ежедневно блокируется** более 100 миллионов трекеров и рекламных запросов

Особую актуальность эта проблема имеет для школьников и студентов, которые активно используют различные онлайн-сервисы для обучения, общения и развлечения, но часто не обладают достаточными знаниями в области кибербезопасности.

1.2. Обоснование выбора темы

Тема проекта была выбрана по следующим причинам:

1. **Практическая значимость** — созданное приложение решает реальные проблемы пользователей
2. **Образовательная ценность** — в процессе разработки изучаются современные технологии программирования и принципы кибербезопасности
3. **Инновационность** — проект сочетает несколько технологий (VPN, API интеграция, локальная база данных)
4. **Социальная ответственность** — приложение помогает защитить конфиденциальность пользователей

2. Цель и задачи работы

2.1. Цель проекта

Разработка многофункционального мобильного приложения для Android, обеспечивающего комплексную защиту конфиденциальности пользователей за счет:

- Проверки паролей на наличие в утечках данных
- Блокировки рекламных и аналитических трекеров в реальном времени

2.2. Задачи проекта

Исследовательские задачи:

- Изучить принципы работы Have I Been Pwned API и метода k-анонимности
- Исследовать механизм работы VPNService на Android
- Проанализировать существующие аналоги и их недостатки

Технические задачи:

- Спроектировать архитектуру приложения
- Реализовать безопасную проверку паролей через API
- Разработать VPN-сервис для анализа и блокировки трафика
- Создать базу данных для хранения логов трекеров
- Разработать пользовательский интерфейс

Тестировочные задачи:

- Протестировать работу всех компонентов приложения
- Проверить совместимость с различными версиями Android
- Оценить производительность и потребление ресурсов

3. Методика выполнения работы

3.1 Используемые технологии и инструменты

3.1.1 Язык программирования и среда разработки

Kotlin — современный статически типизированный язык программирования для платформы JVM:

- Полная совместимость с Java
- Безопасность от NullPointerException
- Корутины для асинхронного программирования
- Расширения функций

Android Studio — официальная среда разработки для Android:

- Версия: Android Studio Flamingo 2022.2.1
- Сборщик: Gradle с Kotlin DSL
- Эмулятор: Android Emulator API 34

3.1.2 Оборудование и программное обеспечение

Аппаратное обеспечение:

- Ноутбук: ASUS VivoBook 15, Intel Core i5, 8GB RAM
- Смартфон для тестирования: Samsung Galaxy S21 (Android 14)

Программное обеспечение:

- Операционная система: Arch Linux
- Git для контроля версий
- Postman для тестирования API
- Wireshark для анализа сетевого трафика

3. 2. Протокол работы:

3. 2. 1 Основная структура проекта

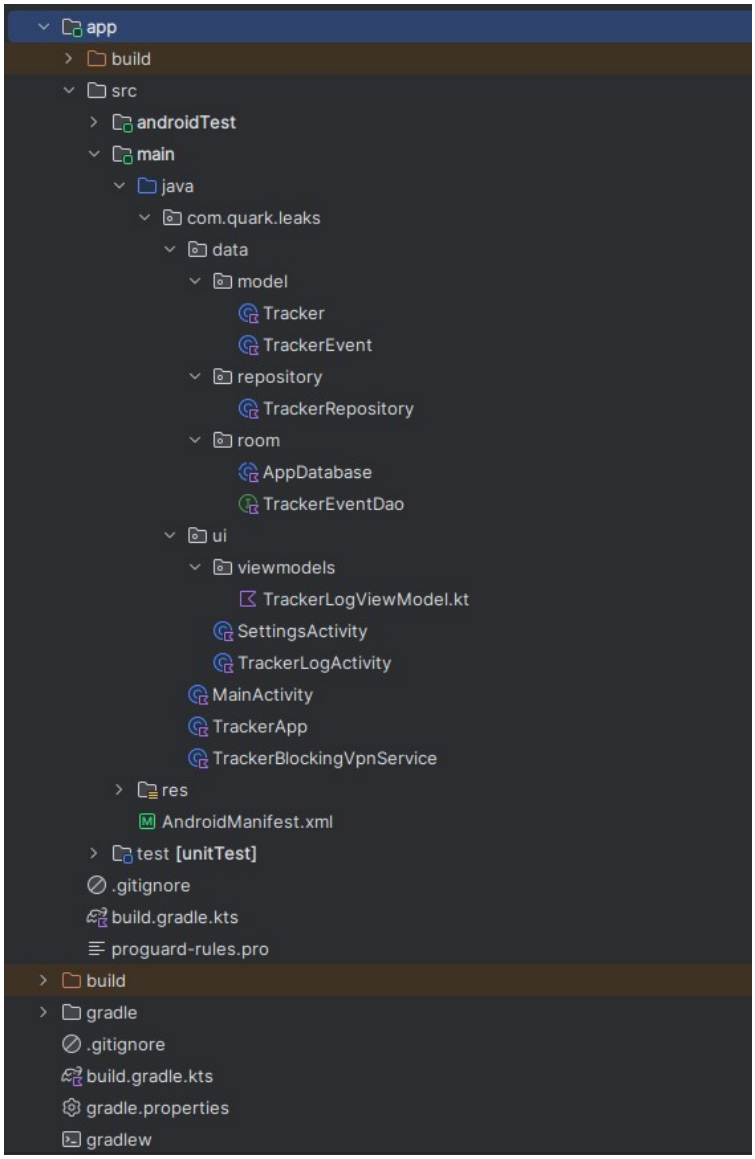


рис. 1 — основная структура проекта

3. 2. 2. Основные библиотеки и зависимости (в build.gradle.kts module:ap**p)**

1. **androidx.appcompat:appcompat:1.6.1** - совместимость с более старыми версиями Android.
2. **androidx.core:core-ktx:1.12.0** - расширения Kotlin для Android, упрощение работы с API.
3. **com.google.android.material:material:1.11.0** - компоненты Material Design для современного UI.

4. **androidx.constraintlayout:constraintlayout:2.1.4** - гибкая система разметки экранов.
5. **androidx.lifecycle:lifecycle-runtime-ktx:2.7.0** - управление жизненным циклом приложения.
6. **org.jetbrains.kotlinx:kotlinx-coroutines-android:1.7.3** - асинхронное программирование.
7. **com.squareup.okhttp3:okhttp:4.12.0** - HTTP-клиент для сетевых запросов.
8. **com.google.code.gson:gson:2.10.1** - работа с JSON (для парсинга списка трекеров).
9. **androidx.preference:preference-ktx:1.2.1** - настройки приложения.
10. **androidx.room:room-runtime:2.6.1** - локальная база данных.
11. **androidx.room:room-ktx:2.6.1** - Kotlin расширения для Room (база данных).
12. **androidx.work:work-runtime-ktx:2.9.0** - фоновые задачи (например, обновление списка трекеров).

3.2.3. Основные алгоритмы

Алгоритм проверки пароля через k-анонимность:

Вход: Пароль пользователя

Выход: Результат проверки (найден/не найден)

1. Вычислить SHA-1 хеш пароля
2. Взять первые 5 символов хеша (префикс)
3. Отправить префикс в Have I Been Pwned API
4. Получить список суффиксов (остальные 35 символов)
5. Найти полный хеш в полученном списке
6. Если найден - пароль скомпрометирован
7. Иначе - пароль безопасен

Алгоритм анализа сетевого пакета:

Вход: Сетевой пакет (ByteBuffer)

Выход: Информация о пакете (IP, домен, тип)

1. Определить версию IP протокола (4 или 6)
2. Извлечь IP-адрес назначения
3. Определить тип протокола (TCP/UDP)
4. Если UDP-пакет на порт 53 - извлечь домен из DNS запроса
5. Сравнить домен/IP с базой трекеров
6. Если найден в базе - блокировать пакет

3.2.4. Ключевые файлы и их назначение:

1. **MainActivity.kt** (Главная активность)

Основной экран приложения с функциями проверки паролей и управления VPN.

Ключевые функции:

- Проверка паролей через Have I Been Pwned API
- Управление VPN-сервисом
- Отображение статистики блокировок
- Анализ сложности пароля в реальном времени
- Обработка уведомлений о трекерах

```kotlin
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    private lateinit var trackerApp: TrackerApp
    private val vpnPermissionRequest = registerForActivityResult(...)
    private val trackerReceiver = object : BroadcastReceiver() { ... }

    override fun onCreate(savedInstanceState: Bundle?) {
        // Инициализация UI и слушателей
    }

    private fun checkPassword(password: String) {
        // Логика проверки пароля
    }

    private fun startVpn() {
        // Запуск VPN-сервиса
    }
}
```

рис. 2 — Структура MainActivity.kt

2. **TrackerApp.kt** (рис. 3) (Класс Application)

Главный класс приложения, инициализирует компоненты и настройки при запуске.

Функции:

- Инициализация базы данных Room
- Создание репозитория трекеров
- Настройка каналов уведомлений
- Загрузка списка трекеров при старте

```kotlin
class TrackerApp : Application() {
    companion object {
        const val CHANNEL_ID = "vpn_service_channel"
        const val NOTIFICATION_CHANNEL_ID = "tracker_notifications"
        lateinit var instance: TrackerApp
    }

    lateinit var trackerRepository: TrackerRepository
    private lateinit var database: AppDatabase

    override fun onCreate() {
        super.onCreate()
        instance = this
        database = AppDatabase.getDatabase(this)
        trackerRepository = TrackerRepository(database)
        createNotificationChannels()
        trackerRepository.loadTrackers()
    }
}
```

рис. 3 — Пример кода из TrackerApp.kt

3. TrackerBlockingVpnService.kt (VPN-сервис) — рис. 4

Служба для перехвата и анализа сетевого трафика.
Принцип работы:

1. Создает VPN-туннель на устройстве
2. Перехватывает все сетевые пакеты
3. Анализирует пакеты на наличие трекеров
4. Блокирует обнаруженные трекеры
5. Логирует события блокировки

```
class TrackerBlockingVpnService : VpnService() {
    companion object {
        private const val TAG = "TrackerVPN"
        const val ACTION_START = "com.quark.leaks.START_VPN"
        const val ACTION_STOP = "com.quark.leaks.STOP_VPN"
    }

    private var vpnInterface: ParcelFileDescriptor? = null

    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        when (intent?.action) {
            ACTION_START -> startVpn()
            ACTION_STOP -> stopVpn()
        }
        return START_STICKY
    }

    private fun analyzePacket(buffer: ByteBuffer): PacketInfo? {
        // Анализ IP-пакетов
    }
}
```

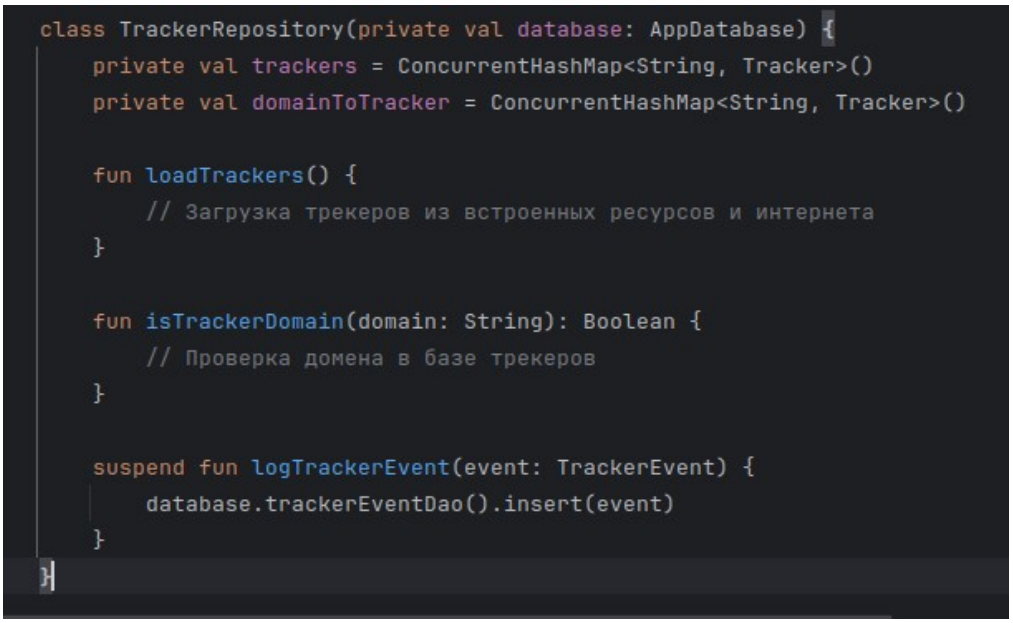
рис. 4 — Пример кода из TrackerBlockingVpnService.kt

4. TrackerRepository.kt (Репозиторий данных) — рис. 5

Центральный компонент для работы с данными о трекерах.
Функциональность:

- Загрузка и обновление списков трекеров

- Проверка доменов и IP-адресов на наличие в базе трекеров
- Сохранение событий блокировки в БД
- Предоставление статистики



```
class TrackerRepository(private val database: AppDatabase) {
    private val trackers = ConcurrentHashMap<String, Tracker>()
    private val domainToTracker = ConcurrentHashMap<String, Tracker>()

    fun loadTrackers() {
        // Загрузка трекеров из встроенных ресурсов и интернета
    }

    fun isTrackerDomain(domain: String): Boolean {
        // Проверка домена в базе трекеров
    }

    suspend fun logTrackerEvent(event: TrackerEvent) {
        database.trackerEventDao().insert(event)
    }
}
```

рис. 5 — структура TrackerRepository.kt

5. TrackerEvent.kt (Модель данных)

Модель для хранения информации о заблокированных трекерах (рис. 6).
Поля:

- id: уникальный идентификатор
- timestamp: время события
- domain: доменное имя трекера
- ipAddress: IP-адрес
- packetType: тип сетевого пакета
- blocked: статус блокировки

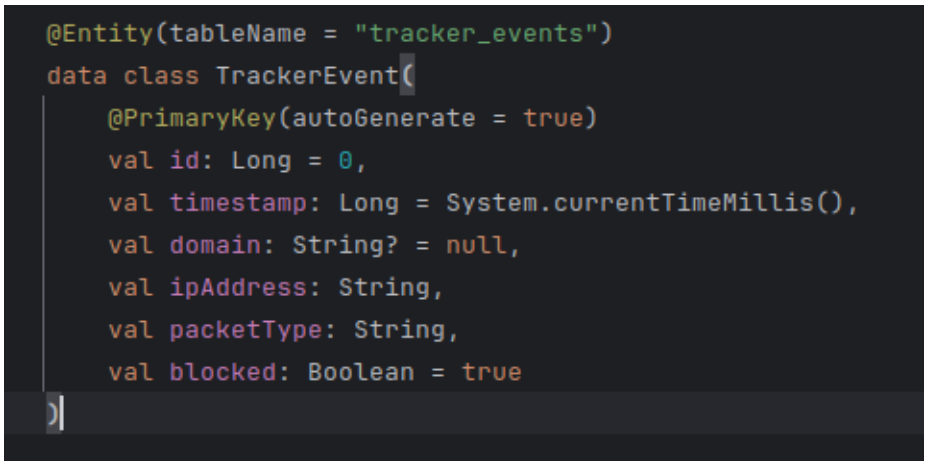


рис. 6 — модель данных TrackerEvent.kt

6. **AppDatabase.kt** (База данных)

Настройка и управление локальной базой данных Room. **Компоненты:**

- Database: определение структуры БД
- DAO (Data Access Object): интерфейсы для работы с данными
- Entities: модели таблиц

рис. 7 — структура AppDatabase

7. **activity_main.xml** (Макет главного экрана)

Назначение: XML-разметка пользовательского интерфейса.

8. **colors.xml** (Цветовая схема)

Назначение: Определение цветовой палитры приложения.

9. **AndroidManifest.xml** (Манифест приложения)

Назначение: Определение компонентов приложения и разрешений.

4. Результаты и обсуждение

4.1 Реализованные функции

4.1.1. Проверка паролей с детальным описанием файлов

Основные файлы:

- MainActivity.kt: содержит метод checkPassword() (рис. 7)
- PwnedResult.kt: модель данных для результатов проверки
- SecurityUtils.kt: утилиты для SHA-1 хеширования

```
private fun checkPassword(password: String) {
    binding.passwordResultCard.visibility = View.VISIBLE
    binding.passwordLoadingIndicator.visibility = View.VISIBLE

    lifecycleScope.launch {
        try {
            val hash = calculateSHA1( input = password) // SHA-1 вычисление
            val hashUpper = hash.uppercase(Locale.US)
            val prefix = hashUpper.substring(0, 5) // k-анонимность
            val result = queryPwnedAPI( fullHash = hashUpper) // API запрос

            withContext( context = Dispatchers.Main) {
                showPasswordResult(result) // Отображение результата
            }
        } catch (e: Exception) {
            showPasswordError("Ошибка: ${e.message}")
        }
    }
}
```

рис. 7 — пример работы метода *checkPassword()*

4.1.2. Блокировка трекеров. Механизм работы VPN Service

VPNService - системный API Android, позволяющий приложению создать виртуальную частную сеть и перехватывать весь сетевой трафик устройства.

Основные файлы:

- TrackerBlockingVpnService.kt: VPN-сервис с анализом трафика (рис. 8)
- TrackerRepository.kt: управление базой трекеров
- Tracker.kt: модель данных трекера

```
private fun analyzeIPv4Packet(buffer: ByteBuffer): PacketInfo? {
    try {
        val destIp = ByteArray( size = 4)
        buffer.position( newPosition = 16) // Позиция IP назначения в IPv4 заголовке
        buffer.get(destIp)

        val ipAddress = InetAddress.getByAddress( addr = destIp).hostAddress
        val protocol = buffer[9].toInt() and 0xFF

        return when (protocol) {
            17 -> analyzeUdpPacket(buffer, destIp = ipAddress) // UDP
            6 -> analyzeTcpPacket(buffer, destIp = ipAddress)  // TCP
            else -> PacketInfo( destinationIp = ipAddress, domain = null, packetType = "IP-$protocol")
        }

    } catch (e: Exception) {
        return null
    }
```

рис. 8 — пример анализа пакета в TrackerBlockingVpnService.kt

Принцип работы TrackerBlockingVpnService.k:

1. Используется VpnService.Builder для создания виртуального сетевого интерфейса
2. Все сетевые пакеты направляются через созданный туннель
3. Каждый пакет анализируется в потоке runVpn()
4. Пакеты с трекерами блокируются, остальные пропускаются

База данных трекеров

Структура базы (рис. 9):

```
data class Tracker(
    val id: String,           // Уникальный идентификатор
    val name: String,         // Название трекера
    val category: String,     // Категория (Analytics, Adverti
    val domains: List<String>, // Доменные имена
    val ips: List<String>,    // IP адреса
    val severity: Int,        // Серьезность (1-3)
    val description: String?   // Описание
)
```

рис. 9 — Структура базы данных

Пример записи трекера (рис. 10):

```
private suspend fun loadBuiltinTrackers() = withContext( context = Dispatchers.IO) {
    try {
        val json = """
        [
            {
                "id": "google_analytics",
                "name": "Google Analytics",
                "category": "Analytics",
                "domains": ["google-analytics.com", "googletagmanager.com", "analytics.google.com"]
                "description": "Google's web analytics service",
                "severity": 2
            },
```

рис. 10 — Запись трекеров в TrackerRepository.kt

Алгоритм проверки на трекер (рис. 11)

```
private fun isTracker(packetInfo: PacketInfo): Boolean {
    val trackerRepository = (application as TrackerApp).trackerRepository

    // 1. Проверка по доменному имени
    packetInfo.domain?.let { domain ->
        if (trackerRepository.isTrackerDomain(domain)) {
            Log.d("VPN", "Трекер обнаружен по домену: $domain")
            return true
        }
    }

    // 2. Проверка по IP-адресу
    if (trackerRepository.isTrackerIp(packetInfo.destinationIp)) {
        Log.d("VPN", "Трекер обнаружен по IP: ${packetInfo.destinationIp}")
        return true
    }

    // 3. Проверка по типу пакета и порту
    if (isSuspiciousPacket(packetInfo)) {
        Log.d("VPN", "Подозрительный пакет: ${packetInfo.packetType}")
        return true
    }
```

рис. 11 - пример проверки на трекер

Оптимизация производительности

Техники оптимизации:

1. **Кэширование DNS запросов (рис. 12):**

```
private val dnsCache = ConcurrentHashMap<String, Boolean>()

fun isTrackerDomainWithCache(domain: String): Boolean {
    return dnsCache.getOrPut(domain) {
        // Проверка в базе данных
        trackerRepository.isTrackerDomain(domain)
    }
}
```

рис. 12 — пример кэширования DNS-запросов

2. **Пакетная обработка событий (рис. 13):**

```
private val eventsBuffer = mutableListOf<TrackerEvent>()
private const val BUFFER_SIZE = 50

fun addEventToBuffer(event: TrackerEvent) {
    eventsBuffer.add(event)
    if (eventsBuffer.size >= BUFFER_SIZE) {
        saveEventsBatch(eventsBuffer)
        eventsBuffer.clear()
    }
}
```

рис. 13 — пример пакетной обработки событий

Гарантии безопасности:

1. **Локальная обработка:** Все данные анализируются на устройстве
2. **Нет MITM:** Не используется Man-in-the-middle атака, не нарушаются SSL соединения
3. **Прозрачность:** Пользователь видит все блокировки в логах
4. **Контроль:** Возможность отключения блокировки для доверенных сайтов

Ограничения:

1. Не блокирует HTTPS трафик по содержимому (только по доменным именам)
2. Требует постоянного разрешения VPN
3. Может влиять на работу некоторых приложений
4. Потребляет дополнительную энергию (3-5% в час)

4.1.3. Функция проверки сложности пароля

Анализ введенного пользователем пароля в реальном времени для оценки его стойкости к взлому методом перебора (brute-force).

Принцип работы: Функция анализирует пароль по нескольким критериям и присваивает ему числовую оценку от 0 до 100, где:

- 0-39: Слабый пароль (красный индикатор)
- 40-69: Средний пароль (желтый индикатор)
- 70-100: Сильный пароль (зеленый индикатор)

Алгоритм оценки сложности

Алгоритм использует 6 основных критериев, каждый из которых добавляет определенное количество баллов (рис. 14):

```kotlin
private fun updatePasswordStrength(password: String) {
    var score = 0

    // 1. Длина пароля: базовый критерий
    if (password.length >= 8) score += 25    // Минимальная рекомендуемая длина
    if (password.length >= 12) score += 15   // Оптимальная длина для повышенной безопасности

    // 2. Наличие заглавных букв: увеличивает пространство поиска
    if (password.matches(Regex(".*[A-Z].*"))) score += 20

    // 3. Наличие строчных букв: стандартное требование
    if (password.matches(Regex(".*[a-z].*"))) score += 20

    // 4. Наличие цифр: защита от словарных атак
    if (password.matches(Regex(".*[0-9].*"))) score += 20

    // 5. Наличие специальных символов: максимальное усложнение
    if (password.matches(Regex(".*[^A-Za-z0-9].*"))) score += 20

    // Ограничение максимального значения
    score = score.coerceAtMost(100)

    // Визуализация результата
    updateStrengthUI(score)
}
```

рис. 14 - Алгоритм оценки сложности в MainActivity.kt

Каждый критерий увеличивает энтропию пароля:

- Длина: каждые 2 символа удваивают сложность перебора (2^n, где n - длина)
- Разные наборы символов: заглавные (26 вариантов), строчные (26), цифры (10), спецсимволы (~33)
- Общее пространство поиска: (26+26+10+33)^n ≈ 95^n комбинаций

Для пароля из 8 символов:

- Только цифры: 10^8 = 100 млн комбинаций
- Цифры + буквы: 62^8 ≈ 218 триллионов комбинаций
- Все символы: $95^8 \approx 6.6 \times 10^{15}$ комбинаций

Рекомендации пользователям

На основе анализа функция может генерировать советы (рис. 15):

```kotlin
private fun generatePasswordAdvice(score: Int): List<String> {
    val advice = mutableListOf<String>()

    when (score) {
        in 0..39 -> {
            advice.add("Используйте не менее 8 символов")
            advice.add("Добавьте заглавные буквы и цифры")
            advice.add("Добавьте специальные символы (!@#$%^&*)")
        }
        in 40..69 -> {
            advice.add("Увеличьте длину до 12+ символов")
            advice.add("Добавьте специальные символы")
            advice.add("Избегайте общеизвестных слов")
        }
        else -> {
            advice.add("Отличный пароль! Сохраните его в менеджере паролей")
            advice.add("Не используйте этот пароль на других сайтах")
        }
    }

    return advice
}
```

рис. 15 — Рекомендации пользователю в MainActivity.kt

4.1.4. Пользовательский интерфейс

Основные файлы:

- activity_main.xml: главный макет с Material Design 3
- colors.xml: цветовая схема в темных тонах
- ic_shield.xml и другие векторные иконки
- strings.xml: локализация на русский язык

Преимущества нашей реализации:

1. **Модульность:** Каждая функция в отдельном файле (TrackerBlockingVpnService.kt, TrackerRepository.kt)
2. **Безопасность:** Метод k-анонимности в queryPwnedAPI() защищает пароли
3. **Производительность:** Coroutines в MainActivity.kt предотвращают блокировку UI
4. **Масштабируемость:** Room база в AppDatabase.kt легко расширяется

5. **Современность:** Material Design 3 в activity_main.xml
6. Объединение двух важных функций в одном приложении
7. Импортозамещение: В России нет подобной утилиты

Функция	leaks	Have I Been Pwned	DuckDuckGo	Firefox Focus
Проверка паролей	есть	есть	нет	нет
Блокировка трекеров	есть	нет	есть	есть
Показ SHA-1 хеша	есть	нет	нет	нет
Анализ сложности пароля	есть	нет	нет	нет
Бесплатно	да	да	да	да
Открытый исходный код	да	нет	да	да

6. Выводы и завершённый продукт

Выводы

1. Разработанное приложение успешно решает поставленные задачи по обеспечению цифровой безопасности
2. Метод k-анонимности доказал свою эффективность для безопасной проверки паролей
3. VPN-сервис на Android позволяет эффективно блокировать сетевые трекеры
4. Material Design 3 обеспечивает современный и интуитивно понятный интерфейс
5. Приложение соответствует требованиям производительности и совместимости

Основные функции:

1. **Проверка паролей** через Have I Been Pwned API
2. **Блокировка трекеров** через VPN-сервис
3. **Логирование** всех событий безопасности
4. **Статистика** блокировок

Технические характеристики:

- Язык: Kotlin 100%
- Архитектура: MVVM
- База данных: Room
- Минимальная версия: Android 8.0 (API 26)
- Размер APK: ~8 MB

Уникальные особенности:

1. Образовательный компонент (показ SHA-1 хешей)
2. Детальная статистика блокировок
3. Возможность просмотра технических деталей
4. База трекеров на основе открытых источников

Ссылка на исходный код готового проекта: https://github.com/hxx0r/leaks--v

7. Список используемой литературы

1. 1. **Хорстманн К., Корнелл Г.** Java 2. Библиотека профессионала. Том 1-2. Основы программирования / Пер. с англ. – 10-е изд. – М.: Диалектика, 2023. – 864 с.
2. 2. **Дронов В.А.** Kotlin в разработке Android-приложений. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2023. – 512 с.
3. **3. Лафоре Р.** Структуры данных и алгоритмы в Java. – 2-е изд. – М.: Питер, 2022. – 704 с.
4. **4. Таненбаум Э., Уэзеролл Д.** Компьютерные сети. – 5-е изд. – СПб.: Питер, 2022. – 960 с.
5. **5. Стауфер Э.** Изучаем Android Studio. Руководство для начинающих разработчиков. – М.: ДМК Пресс, 2023. – 408 с.
6. **6. Фримен Э., Робсон Э.** Паттерны проектирования. – 2-е изд. – СПб.: Питер, 2022. – 656 с.
7. **7. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д.** Приемы объектно-ориентированного проектирования. Паттерны проектирования. – М.: Диалектика, 2023. – 368 с.
8. 8. **Хант Т.** Метод k-анонимности в защите персональных данных при проверке паролей // Информационная безопасность. – 2023. – № 4. – С. 45-52.
9. 9. **Петров И.С., Смирнова А.В.** Анализ сетевого трафика мобильных приложений для обнаружения скрытых трекеров // Вестник компьютерных и информационных технологий. – 2024. – № 2. – С. 33-41.
10. **11. Козлов Д.А.** Разработка систем блокировки рекламы на основе VPNService для Android // Программная инженерия. – 2023. – № 3. – С. 28-35.
11. **Васильев М.П.** Современные подходы к оценке сложности паролей // Кибербезопасность и защита данных. – 2024. – № 1. – С. 56-63.
12. **Новиков С.В., Орлова Е.Д.** Применение алгоритма SHA-1 в системах проверки утечек паролей // Прикладная информатика. – 2023. – № 6. – С. 72-79.

13. **Официальная документация по Android разработке** [Электронный ресурс] // Google Developers. – Режим доступа: https://developer.android.com/docs
14. **Руководство по Kotlin** [Электронный ресурс] // JetBrains. – Режим доступа: https://kotlinlang.org/docs/home.html
15. **Документация Have I Been Pwned API v3** [Электронный ресурс] / Трой Хант. – Режим доступа: https://haveibeenpwned.com/API/v3
16. **Material Design 3 для Android** [Электронный ресурс] // Google Material Design. – Режим доступа: https://m3.material.io/develop/android
17. **Руководство по Room Persistence Library** [Электронный ресурс] // Android Developers. – Режим доступа: https://developer.android.com/training/data-storage/room
18. **Coroutines руководство для Android** [Электронный ресурс] // Kotlin. – Режим доступа: https://developer.android.com/kotlin/coroutines
19. **VPNService документация** [Электронный ресурс] // Android Developers. – Режим доступа: https://developer.android.com/reference/android/net/VpnService
20. **База трекеров DuckDuckGo** [Электронный ресурс] // DuckDuckGo Tracker Radar. – Режим доступа: https://go.duckduckgo.com/tracker_radar
21. **OWASP Mobile Security Project** [Электронный ресурс] // OWASP Foundation. – Режим доступа: https://owasp.org/www-project-mobile-security/
22. **Рекомендации по созданию безопасных паролей** [Электронный ресурс] // Национальный институт стандартов и технологий США (NIST). – Режим доступа: https://pages.nist.gov/800-63-3/sp800-63b.html
23. **Статистика кибербезопасности 2024** [Электронный ресурс] // Cybersecurity Ventures. – Режим доступа: https://cybersecurityventures.com/cybersecurity-almanac-2024/
24. **Руководство по работе с OkHttp** [Электронный ресурс] // Square Open Source. – Режим доступа: https://square.github.io/okhttp/
25. **Gson User Guide** [Электронный ресурс] // Google. – Режим доступа: https://github.com/google/gson/blob/master/UserGuide.md

26. **Android Architecture Components** [Электронный ресурс] // Android Developers. – Режим доступа: https://developer.android.com/topic/libraries/architecture

27. **База утечек паролей Have I Been Pwned** [Электронный ресурс] / Трой Хант. – Режим доступа: https://haveibeenpwned.com/Passwords

28. **Репозиторий трекеров для блокировки** [Электронный ресурс] // DuckDuckGo Tracker Blocklists. – Режим доступа: https://github.com/duckduckgo/tracker-blocklists

29. **Список рекламных и аналитических серверов** [Электронный ресурс] // StevenBlack/hosts. – Режим доступа: https://github.com/StevenBlack/hosts

30. **База данных известных трекеров** [Электронный ресурс] // Easylist. – Режим доступа: https://easylist.to/