УДК 004.77

О.И. Кисляк, магистрант,
С.А. Цололо, канд. техн. наук, доц.,
Донецкий национальный технический университет, г. Покровск, Украина
kissolya13@gmail.com

Особенности разработки анализатора трафика для ОС Android

Анализаторы сетевой активности являются средством решения сетевых проблем, но часто используются и для контроля сетевой активности в корпоративной среде. В связи с большой популярностью мобильных устройств и их использованию в рабочем процессе, существует необходимость контроля персональных и корпоративных данных на этих устройствах. В данной статье будут рассмотрены и проанализированы подходы к разработке анализатора трафика для операционной системы Android.

Ключевые слова: анализатор трафика, нативное приложение, Android, Service, VPN

Введение

Операционная система (ОС) Android продолжает свое развитие, выпуская все новые версии, совершенствуя свое программное обеспечение. Данная система интегрируется в различные устройства, к которым относятся смартфоны, планшеты, телевизоры и другие устройства. В будущем планируется поддержка автомобилей и бытовых роботов [1]. Популярность системы обусловлена открытым исходным кодом, проприетарными драйверы, большим набор библиотек, API и средств разработки.

Развитие системы и ее носителей приводит к появлению новых функций и возможностей, что дает стимул разработчикам создавать новые и интересные приложения. Также возникает необходимость в защите персональных и корпоративных данных на пользовательском устройстве. Данная защита может быть реализована с помощью анализатора трафика. В зависимости от целей его использования он может послужить как злоумышленникам, так и пользователю и/или компании. Приложения данного типа работают в фоновом режиме, пропуская через себя весь трафик и анализируя пакеты данных. Всю подозрительную активность анализатор трафика может блокировать, и отправлять данные в базу. В данной статье будут рассмотрены основные подходы к реализации анализатора трафика для ОС Android.

Архитектура ОС Android

В основе ОС лежит модифицированная и урезанная Linux и Dalvik – собственная реализация виртуальной машины (ВМ) Java от Google. Android позволяет создавать Java-приложения, управляющие устройством через разработанные Google библиотеки. Android Native Development Kit позволяет импортировать библиотеки и компоненты приложений, написанные на C и других языках. Для ВМ Dalvik был разработан формат установочных пакетов –.apk, так как фактически Android-приложения – это программы в нестандартном байт-коде [1].

Архитектура ОС Android является многоуровневой и формируется из компонентов, каждый из которых построен на основе элементов более низкого уровня. Архитектура показана на рис. 1. Кратко выделим особенности некоторых уровней [2]:

1. Linux-ядро обеспечивает корректное функционирование системы и действует как уровень абстракции между аппаратным обеспечением и программным стеком.
2. Уровень библиотек обеспечивает базовую функциональность приложений. Библиотеки реализованы на C/C++ и скомпилированы под конкретное аппаратное обеспечение, вместе с которым они и поставляются производителем.

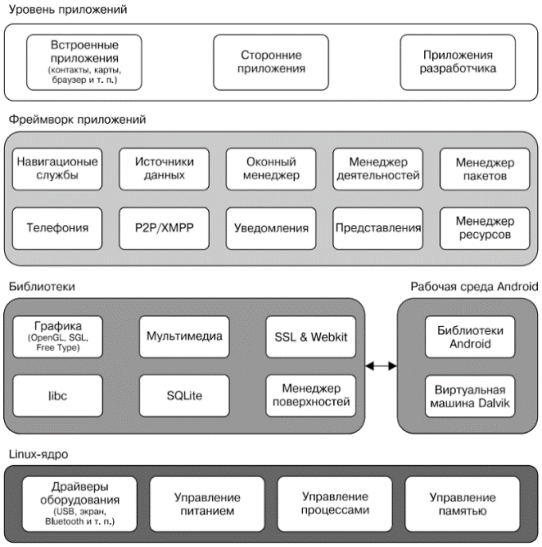

Рисунок 1 – Архитектура ОС Android

3. Android Runtime обеспечивает большую часть низкоуровневой функциональности, доступной библиотекам ядра языка Java.
4. Application Framework позвлдяте получить доступ к API, а к реализованным возможностям других приложений.

Каждое приложение в системе работает в собственном экземпляре виртуальной машины Dalvik и имеет свой уникальный идентификатор. Таким образом система Android реализует принцип предоставления минимальных прав. То есть каждое приложение по умолчанию имеет доступ только к тем компонентам, которые ему необходимы для работы, и ни к каким другим. Однако возможны варианты, когда приложение может предоставить данные другим приложения или обратиться к системным службам, при условии, что приложение получило разрешения на момент его установки [3].

Компоненты Android-приложения

В общем случае стандартное Android-приложение состоит из:

– Java-классов, дочерних от основных классов из Android SDK и Java-классов без родителей в Android SDK;
– манифеста приложений;
– ресурсов;
– файлов.

На рис. 2 приведена иерархия классов, с которыми взаимодействует разработчик при работе с Android SDK [4]. Данная модель отражает не все классы, а лишь основные из них.

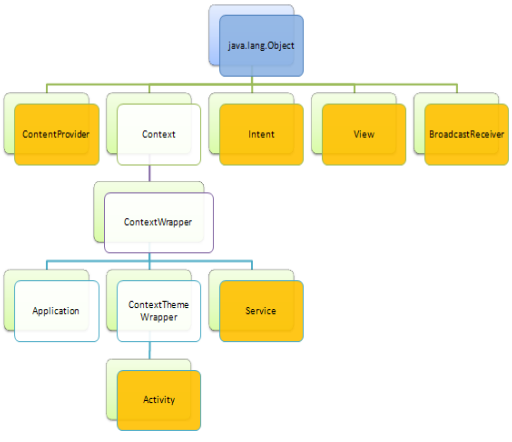

Рисунок 2 – Иерархия Java-классов

Класс Activity и его подклассы отвечает за визуальную активность приложения. Активность имеет жизненный цикл из трех состояний: активна, приостановлена (все еще видна, но потеряла фокус), остановлена [6].

Службы (Service) работают как фоновые процессы [7] и не имеют пользовательского интерфейса. Они предназначены для решения задач, требующих длительного времени, или постоянно находящихся в памяти.

Класс Content provider управляет общим набором данных приложения, позволяет работать с данными в файловой системе, базе данных SQLite и прочими источниками [4].

Манифест Android – это файл, который предоставляет системе всю информацию о приложении [5]. Данный файл содержит:

– разрешения, которые необходимо выдать приложению и от него;
– связанные библиотеки;
– имя Java-пакета приложения;
– описывает компоненты приложения;
– минимальный уровень API Android.

Подходы к реализации приложения

Существует два подхода к реализации приложений – кроссплатформенный и нативный. *Кроссплатформенный подход* позволяет не привязываться к определенной системе, требует меньше ресурсов, использует Android SDK. Это позволяет абстрагироваться от ядра и создавать код на Java. *Нативные приложения* используют всю программно-аппаратную функциональность системы напрямую. Возможно подключение сторонних библиотек и программирование на низком уровне. Скорость работы приложения значительно выше, при этом используется Android NDK (Native Development Kit).

Основная задача работы состоит в разработке анализатора трафика с использованием доступных ресурсов. Поэтому начала рассмотрим возможные подходы к реализации анализатора, их преимущества и недостатки.

Один из подходов – это использование перекомпилированной библиотеки tcpdump. Данная библиотека была написана на языке C/C++, и для ее использования необходимо использовать нативный метод разработки и Android NDK.

В свою очередь NDK использует механизм Java Native Interface (JNI) для взаимодействия Java и C/C++. JNI – это механизм запуска кода на ВМ Java для двух целей: вызывать низкоуровневый код из Java и вызывать Java-методы из низкоуровневого кода. Фактически, это мост между Java и низкоуровневым кодом обеспечивает двусторонние взаимодействие [9].

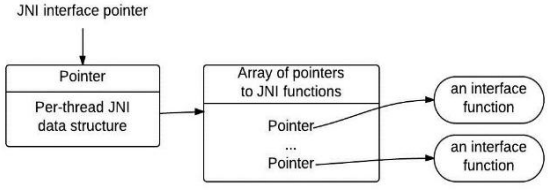

Рисунок 3 – Структура JNI

JNI-таблица по организации похожа на таблицу виртуальных функций в C++. ВМ может работать с несколькими такими таблицами. Например, одна таблица будет применяться для отладки, а вторая – для runtime. Указатель на JNI-интерфейс действителен только в текущем потоке, но нативные методы могут быть вызваны из разных потоков [10]. JNI содержит в себе не только собственный набор данных (примитивные и ссылочные), но и различные функции.

Следует отметить, что JNI не проверяет ошибки такие как NullPointerException или IllegalArgumentException. Причины кроются в снижении производительности и в том, что в большинстве функций C-библиотек трудно защитится от ошибок. Поэтому JNI позволяет использовать Java Exception. Большинство JNI-функций возвращают код ошибок, а не сам *Exception*, и поэтому приходится обрабатывать сам код, а в Java уже выбрасывать *Exception*.

Также важно отметить, что потоки по умолчанию не имеют доступ к *JNIEnv*. Для их присоединения к необходимо вызвать функции *AttachCurrentThread* и *AttachCurrentThreadAsDaemon*. Для отсоединения потока перед завершением необходимо вызвать *DetachCurrentThread*. Android не приостанавливает потоки, которые были были созданы JNI, даже при вызове сборщика мусора.

Структура нативного проекта подобна структуре стандартного приложения, но кроме прочего должна содержать директории jni с нативным кодом и libs (содержит директории под каждую архитектуру процессора, в которой будет лежать нативная библиотека).

Использование перекомпилированной библиотеки *tcpdump* дает возможность использования всего необходимого функционала, но только при условии, что имеются права суперпользователя или root-права. Каждое приложение работает в своей «песочнице» и не имеет доступ к другим компонентам и их данным, если на это не было выдано разрешения. К системным файлам доступ имеется только на чтение. Такие правила позволяют обезопасить систему и пользователя от вредоносного ПО, а также систему от самого пользователя.

Права root предоставляют неограниченный доступ к любому файлу или приложению независимо от того были ли они предоставлены. Такой подход подвергает опасности пользователя и его данные. Реализация такого подхода может быть проблематичной, в связи с тем, что многие производители блокируют данную функцию и ее активация может потребовать выполнения дополнительных действий. Поэтому рассмотрим другие способы, которые реализовываются с помощью стандартных классов набора разработчика Android SDK и не требуют прав супер-пользователя.

Использование прокси-сервера

Первым из таких способов будет установка локального прокси-сервера. Этим способом пользуется большинство антивирусов под Android. Приложению для перехвата трафика достаточно реализовать поддержку HTTP-прокси и установить в настройках *WiFi*-сети прокси-сервер или точку APN в настройках мобильной сети. В этом случае передача данных будет происходить в соответствии со схемой на рис. 4.

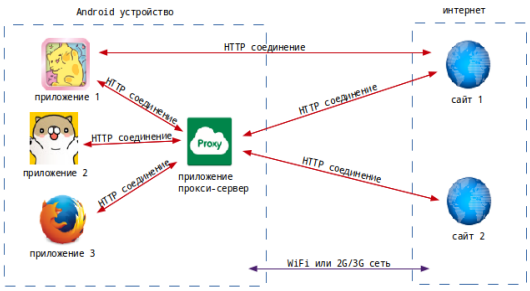

Рисунок 4 – Локальный прокси-сервера

Прокси-сервер – ПО, позволяющее клиентам выполнять косвенные запросы к другим сетевым службам. Сначала клиент подключается к прокси-серверу и запрашивает ресурс, расположенный на другом сервере. Затем прокси-сервер либо подключается к указанному серверу и получает ресурс у него, либо возвращает ресурс из собственного кэша (если он есть). В некоторых случаях запрос клиента или ответ сервера может быть изменён прокси-сервером в определённых целях. Прокси-сервер позволяет защищать компьютер клиента от некоторых сетевых атак и помогает сохранять анонимность клиента [13].

Данный способ имеет ряд недостатков:

– установить WiFi-прокси или APN программно можно только с использование скрытого API и это необходимо выполнять для каждой новой сети;
– работает только для соединений, созданных с помощью класса *Http(s)URLConnection* и то, если явно не будет указано «не использовать прокси»;
– если пользователь удалит или просто не запустит приложение с прокси-сервером, то остальные приложения не смогут соединиться ни с одним сервером – в настройках сети все ещё будет указан прокси и пользователю нужно будет самому удалить его из настроек;
– выстроить цепочку из нескольких прокси нельзя, соответственно и приложение перехватывающее трафик может быть только одно [12].

Однако, все эти проблемы можно решить, если реализовать вместо прокси-сервера VPN-сервер.

Логические сети VPN

VPN (virtual private network) – обобщённое название технологий, позволяющих обеспечить одно или несколько сетевых соединений (логическую сеть) поверх другой сети (например, интернет). Данный подход позволяет реализовать анонимность пользователя в незащищенной сети, а также обезопасить передачу данных, кроме этого благодаря использованию средств криптографии возможно реализовать закрытый канал передачи данных через общедоступные сети.

В зависимости от применяемых протоколов и назначения, VPN может обеспечивать соединения трёх видов: узел-узел, узел-сеть и сеть-сеть. Обычно VPN развёртывают на уровнях не выше сетевого, так как применение криптографии на этих уровнях позволяет использовать в неизменном виде транспортные протоколы (TCP, UDP). Чаще всего для создания виртуальной сети используется инкапсуляция протокола PPP в другой протокол – IP (такой способ использует реализация PPTP – Point-to-Point Tunneling Protocol) или Ethernet (PPPoE)[15]. Классификация VPN представлена на рис. 5.

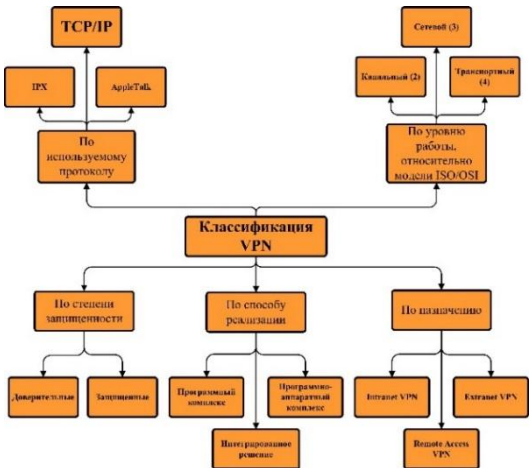

Рисунок 5 – Классификация VPN

VPN состоит из двух частей: «внутренняя» (подконтрольная) сеть, которых может быть несколько, и «внешняя» сеть, по которой проходит инкапсулированное соединение (обычно используется интернет). Возможно также подключение к виртуальной сети отдельного компьютера. Подключение удалённого пользователя к VPN производится посредством сервера доступа, который подключён как к внутренней, так и к внешней (общедоступной) сети. При подключении удалённого пользователя или установке соединения с другой защищённой сетью, сервер доступа требует прохождения процесса идентификации, а затем процесса аутентификации. После успешного прохождения обоих процессов, удалённый пользователь (удаленная сеть) наделяется полномочиями для работы в сети, то есть происходит процесс авторизации.

Принцип работы VPN на практике выглядит следующим образом:

– удаленный пользователь (или маршрутизатор) с использованием клиентского ПО VPN инициирует процедуру соединения с сервером и аутентификацию пользователя – первую фазу установления VPN-соединения;
– в случае подтверждения полномочий наступает вторая фаза – между клиентом и сервером выполняется согласование обеспечения безопасности;
– после этого организуется VPN-соединение, обеспечивающее обмен информацией между клиентом и сервером в форме, когда каждый пакет с данными проходит через процедуры шифрования/дешифрования и проверки целостности – аутентификации данных.

Для реализации анализатора трафика будет использоваться защищенный VPN-туннель между устройством и удаленным сервером. В процессе отправки/приема пакетов они будут просмотрены и проанализированы. Для этого применим один из алгоритмов нечеткого поиска, который будет фиксировать подозрительную активность и отправлять результаты в облачную базу данных, в том случае, если был получен положительный результат.

Такой подход для реализации анализатора трафика был выбран по ряду причин: он кроссплатформенный, не требует рутирования устройства, решает ряд проблем, которые были выявлены при реализации прокси-сервера. Поэтому далее рассмотрим подход с использованием VPN более детально.

Набор разработчика Android SDK предоставляется ряд классов для работы с VPN, которые входят в пакет android.net. К таким классам относятся *VpnService* и *VpnService.Builder*. *VpnService.Builder* является вспомогательным, и необходим для корректной настройки VPN и состоит из методов для построения соединения.

VpnService является базовым классом для приложений, для построения и расширения собственных VPN-решений. Класс создает виртуальный сетевой интерфейс, настраивает адреса и правила маршрутизации и возвращает дескриптор файлу программы. Каждое чтение из дескриптора вытягивает исходный пакет, который был направлен на интерфейс. Каждая операция записи в дескриптор вводит входной пакет в виде, полученном из интерфейса. Интерфейс работает по протоколу IP, поэтому пакеты всегда начинаются с заголовка IP [11].

В классе VpnService есть два основных метода: *prepare (Context)* и *establish ()*. Первый взаимодействует с пользователем и останавливает соединение VPN, созданное другим приложением. Второй создает интерфейс VPN, используя параметры, полученные с помощью класса *VpnService.Builder*. Приложение должно вызвать *prepare (Context)*, чтобы предоставить право на использование других методов в этом классе, и право вызванными в любой момент. Вот общие шаги для создания VPN-соединения:

1. При нажатии на кнопку, чтобы подключения, надо вызвать prepare (Context) и запустить возвращенный интент, если он не нулевой.
2. Когда приложение готово, запустить сервис.
3. Создать тоннель к удаленному серверу и установить сетевые параметры для подключения VPN.
4. Отправить эти параметры к VpnService.Builder и создать VPN-интерфейс, вызвав establish().
5. Обрабатывать обмен пакетов между туннелем и возвращенным дескриптором файла.
6. При вызове onRevoke() закрыть дескриптор файла и тоннель.

Сервисы, которые расширяют этот класс, должны быть объявлены с необходимыми разрешениями. Их доступ должен быть обеспечен разрешением BIND_VPN_SERVICE, и их фильтр должен соответствовать SERVICE_INTERFACE. Более детальную информацию о сервисах и их реализации можно найти в источниках [8, 14].

Далее же остановимся на жизненном цикле сервиса, так как этот вопрос играет важную роль для анализатора трафика. Система Android будет стараться оставить процесс, который выступает хостом сервиса, пока сервис запущен или имеет клиентов. Но если возникнет недостаток в памяти, и система будет убивать существующие процессы для ее освобождения. Чтобы этого избежать необходимо повысить приоритет процесса-хоста сервиса с помощью следующих возможностей [14]:

– если служба выполняет код в методах OnCreate(), onStartCommand() или OnDestroy(), то хост-процесс будет работать в фоновом режиме, чтобы обеспечить завершение выполнения этого кода без сбоев;

– если сервис был запущен, то хост-процесс считается менее важным, чем процессы, которые в данный момент видно на экране, но более важным, чем процесс в фоновом режиме;

– если есть клиенты, привязанные к сервису, то хост-процесс сервиса никогда не является менее важным, чем важнейший клиент. Способ воздействия важности клиента на важность сервиса регулируется с помощью флагов BIND_ABOVE_CLIENT, BIND_ALLOW_OOM_MANAGEMENT, BIND_WAIVE_PRIORITY, BIND_IMPORTANT и BIND_ADJUST_WITH_ACTIVITY;

– запущенный сервис может использовать startForeground (int Notification), чтобы перевести себя в фоновый режим, где система будет считать, что этот сервис активно взаимодействует с пользователем и, таким образом, не является кандидатом в завершение при недостатке памяти.

Исходя из этого, большую часть времени сервис работает, но может быть убит системой, если она находится под сильным давлением памяти. Если произойдет завершение, система позже попытается перезапустить сервис. Важным следствием этого является то, что, если реализуется onStartCommand() для планирования работы, которую необходимо сделать асинхронно или в другом потоке, то можно использовать флаг START_FLAG_REDELIVERY, чтобы иметь возможность системе повторно доставить Intent для пользователя.

Исходя из полученных данных при анализе можно сказать, что рассмотренный подход является наиболее подходящим для реализации анализатора трафика. Проблемы, которые были обнаружены в предыдущих подходах, отсутствуют, а некоторые недостатки, которые могут возникнуть в процессе работы, имеют достаточно эффективное решение. Также немаловажной особенностью данного подхода является то, что он является аппаратно-независимым.

Заключение

В данной работе были рассмотрены различные подходы к реализации анализатора трафика для системы Android. Было рассмотрено три различных подхода, для каждого описаны их недостатки и преимущества. Первый вариант, имеет значительное быстродействие в сравнении с остальными, но не может быть использован из-за того, что требует прав суперпользователя и имеет сложную переносимость на другие устройства. Второй подход имеет ряд недостатков при реализации, которые решаются с помощью третьего подхода. Именно третий подход позволяет организовать необходимый сценарий работы приложения, и постоянно анализировать сетевую активность пользователя в интернете с помощью нечетких алгоритмов, фиксировать полученные данные в облачном хранилище вне системы и мобильного устройства.

Новизна работы заключается в использовании VPN-соединений для перехвата, обработки и контроля содержимого сетевых пакетов, а практическая ценность состоит в разработке программы-сниффера (анализатора трафика) для ОС Android. Предлагаемая система может

быть установлена как на корпоративных мобильных смартфонах, так и на личных устройствах пользователей. Это позволяет реализовать популярную ныне в корпоративной среде модель использования BYOD (Bring Your Own Device). Важнейшей проблемой модели является безопасность и контроль данных, и предлагаемое в работе приложение берет на себя решение части этих проблем. Перспективами разработки мобильного анализатора трафика является создание централизованной структуры контроля с веб-интерфейсом и настройкой прав доступа.

Список литературы

1. Харди Б. , Филлипс Б. Программирование под Android. Для профессионалов. – СПб.: Питер, 2014. – 592 с.: ил.
2. Пол Дейтел, Харви Дейтел Android для разработчиков. – СПб: Питер, 2016. – 512 с.
3. Основы создания приложений [електроннии ресурс]. – Режим доступа: https://developer.-android.com/guide/components/fundamentals.html
4. Архитектура Android-приложений [електроннии ресурс]. – Режим доступа: https://habrahabr.ru/post/141201/
5. МакГрат Майк Создание приложений на Android. – М.: Эксмо, 2016. – 192 с.
6. Activity (Активность, Деятельность) [електроннии ресурс]. – Режим доступа: http://developer.alexanderklimov.ru/android/theory/activity-theory.php
7. Гриффитс Дэвид, Гриффитс Дон. Программирование для Android – СПб: Питер, 2016. – 704 с.
8. Сервис (Service) [електроннии ресурс]. – Режим доступа: http://developer.alexanderklimov.ru-/android/theory/services-theory.php
9. Ретабоуил Сильвен – Android NDK. Разработка приложений под Android на C/C++ – М.: ДМК Пресс, 2012. – 496 с.: ил.
10. Введение в Android NDK [електроннии ресурс]. – Режим доступа: https://habrahabr.ru/post/203014/
11. Описание класса VpnService [електроннии ресурс]. – Режим доступа: https://developer.-android.com/reference/android/net/VpnService.html
12. Разработка VPN-клиента под Android (Часть 1) [електроннии ресурс]. – Режим доступа: https://habrahabr.ru/company/mobisoft/blog/231827/
13. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. – СПб.: Питер, 2001. – 672 с.
14. Service [електроннии ресурс]. – Режим доступа: https://developer.android.com-/reference/android/app/Service.html

Надійшла до редакції 12.09.2016

О.І. КИСЛЯК, С.О. ЦОЛОЛО
Донецький національний технічний університет (Україна)

ОСОБЛИВОСТІ РОЗРОБКИ АНАЛІЗАТОРА ТРАФІКУ ДЛЯ ОС ANDROID

Аналізатори мережевої активності перш за все є засобом вирішення мережевих проблем, але часто використовуються і для контролю мережевої активності в корпоративному середовищі. У зв'язку з великою популярністю мобільних пристроїв і їх використання в робочому процесі, існує необхідність контролю персональних і корпоративних даних на цих пристроях. У даній статті будуть розглянуті і проаналізовані підходи до розробки аналізатора трафіку для операційної системи Android.

Ключові слова: аналізатор трафіку, нативний додаток, Android, Service, VPN.

OLGA KISLYAK, SERGII TSOLOLO
Donetsk National Technical University (Ukraine)

DEVELOPMENT OF THE TRAFFIC ANALYZER FOR ANDROID OS

Analyzers network activity was initially developed as a means of resolving network problems. However, for quite a long time, the company used to monitor network activity, to preserve corporate secrets. Nowadays, mobile devices and tablets gaining popularity, there is a need to protect personal and corporate data on these devices. This article will consider how to develop a traffic analyzer for the Android operating system. First of all it is necessary to understand analyze the platform and its features. Next, you need to consider two fundamentally different approaches to the development of sniffer: development of native and cross-platform applications. It is also necessary to take into account the possible options of implementing the application with using various approaches. two implementations of cross-platform applications were considered: using proxy-server and using VPN-server. In conclusion, we will analyze all advantages and disadvantages of every implementing way and chosen the optimal approach.

Keywords: traffic analyzer, native app, Android, Service, VPN.