

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
РАЗДЕЛ 1. УЯЗВИМОСТИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ.....	5
1.1. Принципы разработки безопасных приложений	9
1.2. Уязвимости операционных систем	12
1.3. Инструменты удалённого доступа	15
1.4. Выводы по разделу 1	21
РАЗДЕЛ 2. СОЗДАНИЕ RAT ДЛЯ ANDROID	22
2.1. Выбор метода связи с сервером	22
2.2. Регистрация в системе FCM	24
2.3. Разработка серверной части.....	26
2.4. Разработка клиентского приложения.....	30
2.5. Тестирование инструмента удалённого доступа	35
2.6. Выводы по разделу 2	37
РАЗДЕЛ 3. ДЕТЕКТИРОВАНИЕ RAT.....	38
3.1. Классификация разрешений	38
3.2. Математическая модель анализа приложений.....	41
3.3. Создание приложения.....	44
3.4. Проведение экспериментов и корректировка коэффициентов ..	49
3.5. Выводы по разделу 3	52
РАЗДЕЛ 4. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ОБМЕНА ДАННЫМИ ПО ЗАЩИЩЕННОМУ КАНАЛУ	53
4.1. Протокол установления связи.....	53
4.2. Функционал приложения	55
4.3. Выводы по разделу 4	60
РАЗДЕЛ 5. ОХРАНА ТРУДА	61
ЗАКЛЮЧЕНИЕ	65
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ.....	66
ПРИЛОЖЕНИЕ 1	68
ПРИЛОЖЕНИЕ 2.....	73
ПРИЛОЖЕНИЕ 3.....	74
ПРИЛОЖЕНИЕ 4.....	75
ПРИЛОЖЕНИЕ 5.....	77

ВВЕДЕНИЕ

В современном мире пользователь все больше времени тратят на приложения, установленные не на персональных компьютерах или ноутбуках, а на смартфонах или планшетах. Это различные мессенджеры, банкинг, бизнес-приложения, личные кабинеты сотовых операторов и т.д. Разработчики уделяют большое внимание дизайну программных продуктов, стараясь сделать их максимально удобными, однако зачастую не очень озабочены безопасностью данных, которые пользователи доверяют производителям этих приложений. По данным международной компании Juniper Research, уязвимости высокого уровня риска обнаружены в 38% мобильных приложений для iOS и в 43% приложений для платформ под управлением Android. Данная работа посвящена анализу уязвимостей как самой операционной системы Android, так и приложений для мобильных устройств, работающих под управлением Android. Исследование проведено на примере приложения для системы ограничения доступа, разработанного для одного из коммерческих предприятий ДНР, и является актуальным для быстро развивающегося сегмента разработки мобильных приложений, представляя интерес как для разработчиков, так и для пользователей.

Цель работы: анализ уязвимостей мобильных приложений под управлением операционной системы Android.

Предмет исследования: уязвимости клиентской и серверной части мобильных приложений под управлением операционной системы Android.

Объект исследования: мобильные приложения для систем ограничения доступа под Android.

Для достижения цели исследования были поставлены следующие задачи:

- Проанализировать известные угрозы и уязвимости мобильных приложений и предлагаемые методы защиты от них;

- Разработать и внедрить мобильное приложение для обмена данными по защищенному каналу в системе ограничения доступа на объект;
- Исследовать вероятные уязвимости разработанного приложения и выделить данные, которые могут быть интересны злоумышленникам и которые необходимо защищать;
- Разработать приложение, имитирующее инструмент удаленного доступа для проведения тестовых экспериментов;
- Провести эксперименты по реализации атак на мобильное приложение и мобильные устройства различных типов;
- Обобщить результаты экспериментов и предложить методику защиты от подобного рода атак;
- Сформулировать рекомендации для разработчиков и пользователей мобильных приложений по их безопасной разработке и эксплуатации.

РАЗДЕЛ 1. УЯЗВИМОСТИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

На данный момент в мире распространено две конкурирующие операционные системы для смартфонов: iOS от компании Apple, и Android, разрабатываемая компанией Google. Обе платформы имеют как сходства, так и различия и развиваются независимо друг от друга, поэтому для каждой операционной системы есть специфические уязвимости, не затрагивающие другую операционную систему. Тем не менее, существует несколько атак на мобильные приложения, осуществимые для обеих платформ:

- 1) обратная разработка приложения, изучение исходного кода и сохранённых приложением файлов в локальном хранилище на устройстве;
- 2) перехват сетевого трафика приложения, осуществление MITM атак;
- 3) повышение привилегий пользователя на устройстве и атака на приложение с помощью отладочных инструментов.

Атака каждого типа на данный момент легко осуществима как для iOS, так и для Android, и защита от этих атак, несмотря на активную работу Apple и Google над безопасностью своих операционных систем, во многом ложится на плечи разработчиков приложений [1, 2].

Рассмотрим основные уязвимости приложений, позволяющие проводить на них вышеизложенные атаки и возможные решения этих проблем:

- 1) Использование незащищенных хранилищ данных. Очень распространённая уязвимость, которая позволяет беспрепятственно просматривать конфиденциальные данные пользователя, такие как: паспортные данные, телефонные номера, адреса электронной почты, физический адрес проживания, пароли и прочее. Устранить эту уязвимость можно с помощью предоставляемых операционной системой защищенных хранилищ. Однако, пока что они не используются по умолчанию и разработчики должны самостоятельно заботиться о сохранности данных

пользователей [3].

2) Хранение приватных данных в коде или ресурсах приложения. К таким приватным данным могут относиться пароли к удалённым серверам или базам данных, ключи шифрования трафика, ключи шифрования локально сохраняемых файлов и т.д. Очевидно, для решения проблемы следует не хранить никаких важных данных в коде или ресурсах приложения. Это достигается за счёт правильной архитектуры системы и использования современных криптографических алгоритмов [2, 3].

3) Использование ассиметричного шифрования с приватным ключом, известным серверу. Это не будет являться уязвимостью, если пользователи дают своё явное согласие на предоставление своих конфиденциальных данных серверу и сервер при этом хорошо защищен. Однако, во-первых, иногда разработчики не оповещают пользователя о возможности расшифровки их данных на сервере, заявляя, что данные хранятся в зашифрованном виде. А во-вторых, разработчики хранят больше данных, чем им необходимо. При разработке подобных систем, следует запрашивать у пользователя только действительно необходимые данные (а по возможности вовсе отказаться от хранения конфиденциальной информации на сервере с возможностью её расшифровки) и требовать явного согласия пользователя на их хранение [4].

4) Использование собственных, не стандартизированных алгоритмов шифрования. Разработка таких систем нарушает принцип Керкгоффса, согласно которому только некоторые параметры криптографической системы должны быть засекречены (например, ключ шифрования). Сам же алгоритм должен быть открыт для изучения и обсуждения. Скрытие алгоритма обычно не является большим препятствием к его криптоанализу, а разработка открытого алгоритма с хорошими криптографическими свойствами требует годы работы учёных. Соответственно, следует использовать современные открытые стандартизированные алгоритмы шифрования с доказанной криптостойкостью [5, 6].

5) Передача критически важной информации в открытом виде. Очень

распространённая уязвимость. Несмотря на повсеместный переход на защищенные протоколы передачи данных, всё ещё можно обнаружить приложения, передающие некоторые пользовательские данные в открытом виде. Это случается, если разработчики не считают некоторые данные критически важными, например, адреса электронной почты, или паспортные данные (имя, год рождения). А также можно встретить передачу ключа для дальнейшего шифрования данных в открытом виде с расчетом на то, что его перехват крайне маловероятное событие. Устранение этой уязвимости сводится, во-первых, к тому, чтобы считать любые пользовательские данными критически важными и требующими шифрования, а во-вторых, к использованию безопасных алгоритмов обмена ключом шифрования [2, 3].

б) Игнорирование рутованных устройств (т.е. устройств с повышенными пользовательскими привилегиями). Обычно на таких устройствах отключены стандартные методы защиты данных приложений, предоставляемые операционной системой [4]. Рутование устройства может происходить как по согласию пользователя, так и без него (при заражении вирусом, например). Поэтому подход к устранению такой уязвимости может быть разный:

- Первый подход заключается в том, чтобы попросту запретить работу приложения, если удалось обнаружить, что устройство рутовано. Но, во-первых, приложению не всегда удастся успешно определить рутованное устройство. Во-вторых, пользователи могут осознанно повышать свои привилегии на устройстве, и запрет использования приложения может плохо сказаться на пользовательском опыте.

- Второй подход - предупреждать пользователя о возможной незащищенности его данных. В этой ситуации пользователь будет явно знать, что его данные могут находиться под угрозой. В таком случае, опыт использования приложения не ухудшится, но это оставит брешь в безопасности.

- В конце концов, можно отказаться от стандартных методов

защиты данных операционной системы и разрабатывать собственные системы защиты. Это обеспечит сохранность данных независимо от того, является ли устройство рутованным или нет. С точки зрения безопасности, это лучший вариант, но на его осуществление придётся потратить большое количество времени, и даже тогда может не удастся достичь того же результата, который предоставляют современные операционные системы.

7) Хранение данных в защищенных хранилищах, но в открытом виде. Несмотря на хорошую безопасность системных защищенных хранилищ, невозможно дать гарантию того, что в них нет уязвимостей. К тому же, как было сказано выше, на рутованных устройствах системные хранилища могут быть скомпрометированы. Не стоит целиком и полностью полагаться на системные средства, и по возможности шифровать любые хранимые на устройстве данные [7].

8) Использование функционала других приложений. Иногда, выполнение некоторых действий разработчики перекладывают на другие, уже установленные на устройстве приложения. Это могут быть программы для чтения документов, проигрывания музыки или видео, просмотра фотографий, веб-браузеры и прочее. В таких ситуациях приложение может только частично контролировать работу стороннего приложения с данными. Устранить такие уязвимости разработчики могут либо добавив весь необходимый функционал непосредственно в своё приложение, либо зашифровав все передаваемые в сторонние приложения данные. В лучшем случае следует полностью отказаться от передачи конфиденциальной информации какому-либо стороннему приложению, включая компоненты операционной системы [8].

Таким образом, сохранность пользовательских данных по большей части зависит от разработчиков. Тем более, что в некоторых ситуациях системы защиты операционной системы могут быть отключены либо осознанно пользователем, либо путём заражения вредоносным программным обеспечением.

1.1. Принципы разработки безопасных приложений

Устранение вышеизложенных уязвимостей в приложении хоть и значительно повышает сохранность данных, всё же не является абсолютной защитой. Помимо этого, следует также придерживаться нескольких принципов, при разработке приложений, несоблюдение которых не всегда является уязвимостью, но может привести к утечке данных или взлому:

В приложениях, защищенных аутентификационными механизмами (пин-код, пароль, отпечаток пальца и т.п.), следует отслеживать переходы пользователя к другим приложениям и требовать аутентификацию при каждом возврате пользователя к приложению. В противном случае это может создать угрозу данным. Например, если телефон был украден или потерян, а приложение в этот момент было запущено [9].

Количество попыток ввода аутентификационных данных должно быть ограничено для противодействия простому перебору. Например, после 3 неудачных попыток, следует запретить вводить пароль на 10 секунд, и увеличивать время с каждой новой неудачной попыткой. Это станет эффективной защитой от перебора даже для коротких паролей [10].

Длина пароля или пин-кода должна быть ограничена снизу. На данный момент рекомендуется использовать коды длиной не менее 6 символов, при учёте, что принцип 2 также соблюдается, в противном случае стоит использовать пароли длиной не менее 8 символов.

В клиент-серверных приложениях время сессии должно быть ограничено, на случай если пользователь забыл закрыть приложение. В таком случае после долгого простоя сессия должна закрываться и сервер должен требовать аутентификации от пользователя. Время жизни сессии и механизм её обновления должен разрабатываться с учетом особенностей приложения и соблюдать баланс между удобством использования приложения и безопасностью. Время окончания сессии должно генерироваться на стороне сервера, также сервер, а не приложение, должен следить за окончанием сессии.

Работа с датой и временем должна осуществляться в абсолютном формате, не зависящем от часового пояса. Так, нельзя определять время жизни сессии как «N минут от данного момента» [11, 12]. Сервер должен присылать абсолютное значение, в какой момент времени сессия будет считаться закрытой. Оптимальным вариантом для работы со временем является UNIX timestamp - количество секунд, прошедших с 00:00:00 UTC 1 января 1970 года.

Приватная пользовательская информация не должна отображаться на экране без явного запроса пользователя, шрифт и цвет текста должны быть подобраны таким образом, чтобы их было достаточно для удобного чтения, но сложно было рассмотреть издали. Это исключит риск «подглядывания из-за плеча».

Стоит с осторожностью относиться к любым сторонним библиотекам с открытым исходным кодом, которые имеют доступ к конфиденциальной информации (например, библиотеки для работы с базами данных). Следует использовать только проверенные временем и большим количеством пользователей библиотеки. Библиотеки с закрытым исходным кодом вовсе не должны иметь доступ к приватным данным.

Нельзя использовать криптографические библиотеки с закрытым исходным кодом. Это не позволит проверить корректность реализации алгоритмов и не дает гарантии того, что в них нет программных закладок. В данном случае следует также доверять проверенным библиотекам с открытым исходным кодом или реализовать алгоритмы самостоятельно. Однако для реализации вычислительно эффективной и при этом криптографически стойкой реализации современных алгоритмов требуется большой багаж знаний как в области разработки, так и в области криптографии. Большинству разработчиков рекомендуется использовать библиотеки предоставляемые вместе с языком программирования или проверенные библиотеки с открытым кодом.

При использовании любых сторонних библиотек нужно помнить, что в этих библиотеках тоже могут быть уязвимости. Необходимо следить за

сообщениями об этих уязвимостях, обновлять библиотеки до версий с устраненными уязвимостями и максимально быстро публиковать новые версии приложения [18].

Перед публикацией приложения необходимо убрать из кода логгирование любой отладочной и приватной информации. Следует рассчитывать на то, что лог может прочитать кто-угодно. При необходимости сбора логов следует озаботиться шифрованием всех собираемых данных.

Устранение изложенных в предыдущей главе уязвимостей вместе с соблюдением вышеописанных принципов позволяет создавать приложения, устойчивые ко взлому и краже персональных данных пользователей. В то же время, на данном этапе развития мобильных операционных систем большинство этих задач ложится на плечи разработчиков приложений. В текущие задачи специалистов по информационной безопасности входит разработка механизмов устранения уязвимостей и упрощение соблюдения этих принципов на уровне операционных систем или разработка открытых библиотек для решения этих задач.

1.2. Уязвимости операционных систем

Помимо уязвимостей приложений, которые возникают при написании кода, разработчикам также стоит помнить, что их приложения выполняются на устройствах, управляемых какой-то конкретной операционной системой, и созданных на основе каких-то компонентов. А операционную систему и прошивки компонентов также создают люди, которые могут допускать ошибки или оставлять уязвимости. Особенно это касается системы Android, так как производители устройств с этой операционной системой могут использовать самые различные компоненты для сборки своей техники. Кроме того, поддержка малобюджетных устройств либо вовсе не производится, либо производится из рук вон плохо, поэтому важные обновления безопасности систем могут либо не доходить до пользователей, либо доходить с огромной задержкой. В случае с iOS факт того, что разработка операционной системы и всех устройств, её поддерживающих, полностью находится в руках компании Apple позволяет последней очень быстро закрывать новые уязвимости и выпускать обновления безопасности. Таким образом, злоумышленникам часто приходится атаковать iOS либо через уязвимости приложений, либо через 0-day (т.е. неизвестные широкой публике либо ещё неисправленные) уязвимости [19, 20].

Перекрытие уязвимостей в операционной системе или прошивке компонентов устройства далеко не всегда возможна на уровне приложения, однако разработчикам в любом случае следует следить за этими уязвимостями. Это позволит по возможности устранять их, оповещать пользователей, или в конце концов не быть застигнутыми врасплох при выявлении утечки данных.

Ниже приведены самые опасные уязвимости операционной системы Android, существующие в наиболее распространённых на данный момент её версиях.

BlueBorne - уязвимость созданная набором ошибок в работе Bluetooth современных операционных систем, в том числе и Android. Ошибки находятся в коде функции `l2cap_parse_conf_rsp` ядра Linux, начиная с версии 3.3. Уязвимость может быть эксплуатирована, если на устройстве включена функция Bluetooth, и для её эксплуатации атакующему достаточно находится в радиусе действия передатчика Bluetooth. Отправляя на устройство специальным образом отформатированные пакеты, можно выполнять вредоносный код с привилегиями ядра Linux, то есть теоретически можно получить полный контроль над атакуемым устройством. Уязвимость изначально существовала в версиях Android от 4.4 до 8.0. Позже компания Google выпустила обновления безопасности, однако всё ещё велика вероятность встретить устройство с рабочей уязвимостью. Также стоит отметить, что эта уязвимость не страшна для устройств, использующим технологию BLE (Bluetooth Low Energy) [15].

Extra Field - эта уязвимость позволяет подделывать существующие приложения. Существует в версии Android 4.4. Эта версия, несмотря на свой преклонный возраст, всё ещё популярна среди производителей телевизоров или какой-либо специальной техники. Заключается она в том, что злоумышленник может подделать некоторые поля установочного файла приложения при этом не изменив его цифровую подпись. Таким образом в приложение можно встроить вредоносный код, например, для кражи конфиденциальных данных пользователей. В последующих версиях операционной системы эта уязвимость была устранена.

Janus - уязвимость аналогична предыдущей, но для более свежих версий операционной системы. Принцип эксплуатации уязвимости отличается от Extra Field, однако результат тот же - можно внедрить вредоносный код в APK файл при этом не изменяя его цифровую подпись. Начиная с версии 7.0 в Android стала применяться новая схема подписи приложения - Signature Scheme v2, которая исключает возможность эксплуатации этой уязвимости. Но выбор схемы подписи приложения - это задача разработчика, поэтому

некоторые приложения, подписанные без использования новой схемы, всё ещё могут быть уязвимы для этой уязвимости на версиях Android до 8.0 [16].

ObjectInputStream Serialization - уязвимость в версиях Android до 5.0. Уязвимость скрывается в системе сериализации и десериализации объектов. Сериализация - это возможность некоторых объектов в коде программы трансформироваться в набор байт, для сохранения на диске например. Обратный процесс называется десериализацией. Эксплуатация данной функции позволяет аварийно завершать системные процессы в Android, в том числе критически важные, создавая таким образом полигон для дальнейших атак на систему.

OpenSSLX509Certificate - ошибка в системном компоненте Android версий 4.3 - 5.1, позволяющая приложению выполнять код с системными привилегиями. С её помощью можно, например, заменить существующее приложение на другое, с внедрённым в него вредоносным кодом.

StageFright - уязвимость в библиотеке обработки файлов MP4 в Android версий 2.2 - 5.1.1. Она позволяет выполнить любой код с системными привилегиями, при попытке открыть файл MP4 или даже папку, в которой этот файл находится [17].

Cloak and Dagger - позволяет приложению получить полный доступ к операционной системе, создав прозрачные окна поверх окон других приложений. В результате этого приложение сможет перехватывать нажатия на экран, нажатия на клавиши, а также доступ к любой конфиденциальной информации. Уязвимость актуальна для версий Android 4.4 - 7.1.2.

Таким образом, помимо уязвимостей, появляющихся при ошибках разработки конкретных мобильных приложений, разработчикам и простым пользователям стоит помнить о возможности существования ошибок в операционных системах на устройствах, или в прошивках компонентов этих устройств. Чаще всего, устранение подобных уязвимостей возможно только со стороны разработчиков операционных систем, однако информация о них будет полезна и разработчикам приложений, и пользователям устройств.

1.3. Инструменты удалённого доступа

Инструментами удалённого доступа (Remote Access Tool, или сокращённо - RAT) называют приложения, дающие полный или ограниченный доступ к устройству через интернет, или другое соединение. Такие приложения существуют для всех операционных систем. Особенностью RAT является то, что они могут разрабатываться и использоваться как для легитимного использования устройства, так и для слежки за пользователями. В первом случае это зачастую приложения, позволяющие нескольким пользователям одновременно работать с одним устройством или настраивать устройства не имея к ним физического доступа. Например, системный администратор может использовать подобные программы для настройки компьютеров сотрудников без необходимости личного присутствия в офисе. Во втором же случае, RAT встраиваются в компоненты приложений, выполняющих другую функцию. К примеру, приложение-чат может на самом деле считывать данные с микрофона или видеокамеры устройства, или исследовать файловую систему в поисках конфиденциальной информации. Более того, такие приложения могут выполнять свою вредоносную функцию даже если пользователь на данный момент им не пользуется.

Такой дуализм инструментов удалённого доступа не позволяет однозначно отнести их к вирусам сильно усложняет борьбу с вредоносными приложениями. Это особенно важно для мобильных устройств, так как современные смартфоны имеют возможность собирать беспрецедентно много информации: текущее местоположение и история передвижений, контакты, сообщения, информация о вызовах, данные с микрофона и видеокамеры, данные с экрана, загрузки, фотографии, видео, данные других приложений и другую конфиденциальную информацию.

В связи с этим, разработчики Android и iOS с каждой новой версией активно внедряют новые функции, усложняющее (но не устраняющие) возможность создания RAT-приложений для своих систем, а также

позволяющие дать пользователю максимально полную информацию о действиях приложений. Так, в последних версиях iOS система создаёт особое оповещение в строке состояния о том, что приложение использует микрофон или видеокамеру. Как в Android, так и в iOS существует и с каждой версией дополняется расширенная система разрешений для приложений. Никакое приложение не сможет получить доступ к той или иной функции устройства пока пользователь в явном виде не предоставит ему разрешение. Пользователи могут просматривать и изменять разрешения приложений в настройках устройства.

Также, обе операционные системы очень строго контролируют работу приложения в фоновом режиме, то есть, когда пользователь не использует устройство, или работает с другим приложением. Например, в iOS разработчик приложения может выбрать один или несколько заранее определённых сценариев работы в фоновом режиме, таких как: периодическое получение информации о местоположении устройства, синхронизация данных, работа VoIP приложения, связь с подключаемыми устройствами (например, Bluetooth наушники) и несколько других. В каждом из сценариев приложение получает доступ только к необходимым для исполнения этого сценария функциям и не более того [19, 20].

В Android для работы в фоновом режиме у приложений есть больше возможностей, однако от версии к версии компания Google добавляет новые функции, призванные обезопасить эту работу. Так, приложение, совершающее постоянную работу в фоновом режиме, обязано создать особое уведомление в строке состояния, которое даёт пользователю понять, что приложение что-то делает.

Помимо этого, обе системы работают над максимальной изоляцией приложений друг от друга. Например, у каждого приложения есть собственная область в файловой системе, в которую не может попасть ни приложения, ни пользователь устройства. Чтение и запись файлов вне этой области осуществляется только с явным участием пользователя.

В конце концов, Apple и Google поддерживают собственные библиотеки приложений, из которых пользователи могут скачать приложения и установить на своё устройство. Все, добавляемые в эти библиотеки приложения, проходят строгую проверку по безопасности, что сильно усложняет добавление вредоносных приложений в библиотеку и установку на устройство пользователей. Тем не менее, пользователи могут скачивать и устанавливать приложения из других источников.

Все эти мероприятия, предпринимаемые компаниями Apple и Google, позволяют повысить безопасность использования приложений, а также осведомлённость пользователей о происходящих в операционной системе процессах. Однако, они не исключают возможность создания инструментов удалённого доступа, сохраняя максимально широкий функционал операционной системы, поэтому для того, чтобы не стать жертвой вредоносных программ с функцией удалённого доступа, пользователи сами должны предельно внимательно следить за тем, какие приложения они устанавливают на свои устройства, какие функции эти приложения используют и соответствуют ли используемые функции заявленному функционалу приложений.

Под данным же исследований, пользователи обращают очень мало внимания на запрашиваемые приложением разрешения при установке приложения. Так, только 17% пользователей вообще обращают внимание на то какие разрешения требуются приложению и только 3% пользователей могут сказать зачем приложению необходимо получить то или иное разрешение и что оно будет способно сделать, получив его [21].

При этом, даже если бы пользователи уделяли больше внимания разрешениям, им бы пришлось самостоятельно искать и информацию о разрешениях, так как сама система всё ещё не предоставляет удовлетворительного описания и разъяснения к запрашиваемым разрешениям [22]. На рисунке 1.1 представлен пример запроса разрешения от приложения.

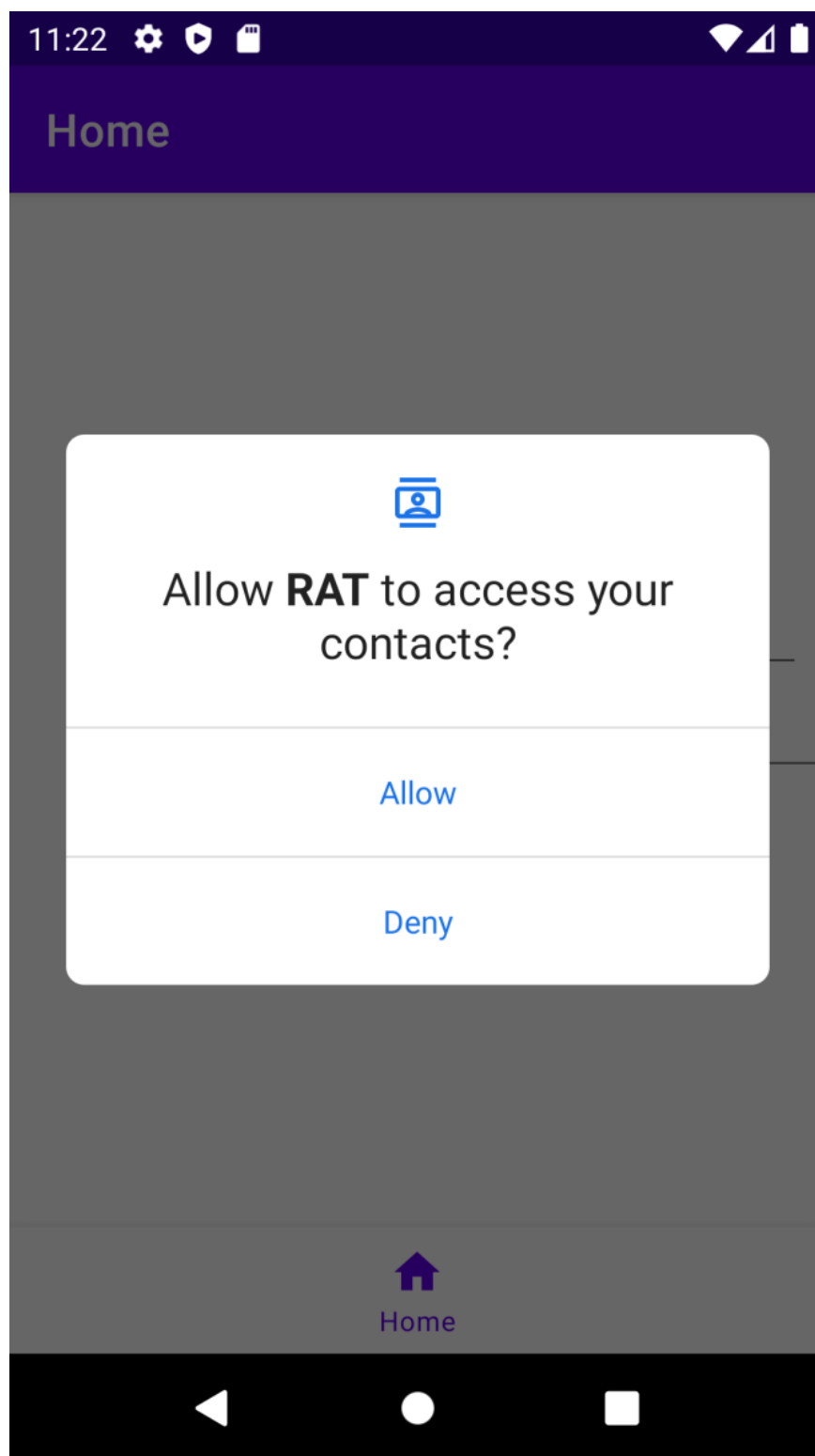


Рисунок 1.1. Запрос разрешения на доступ к контактам

В запросе указан доступ к контактам, без должного разъяснения о том зачем нужно это разрешение и какие возможности оно фактически предоставляет. Также пользователи могут посмотреть разрешения

приложения после установки в настройках системы. И там также нет никакой дополнительной информации об этих разрешениях (Рис. 1.2).

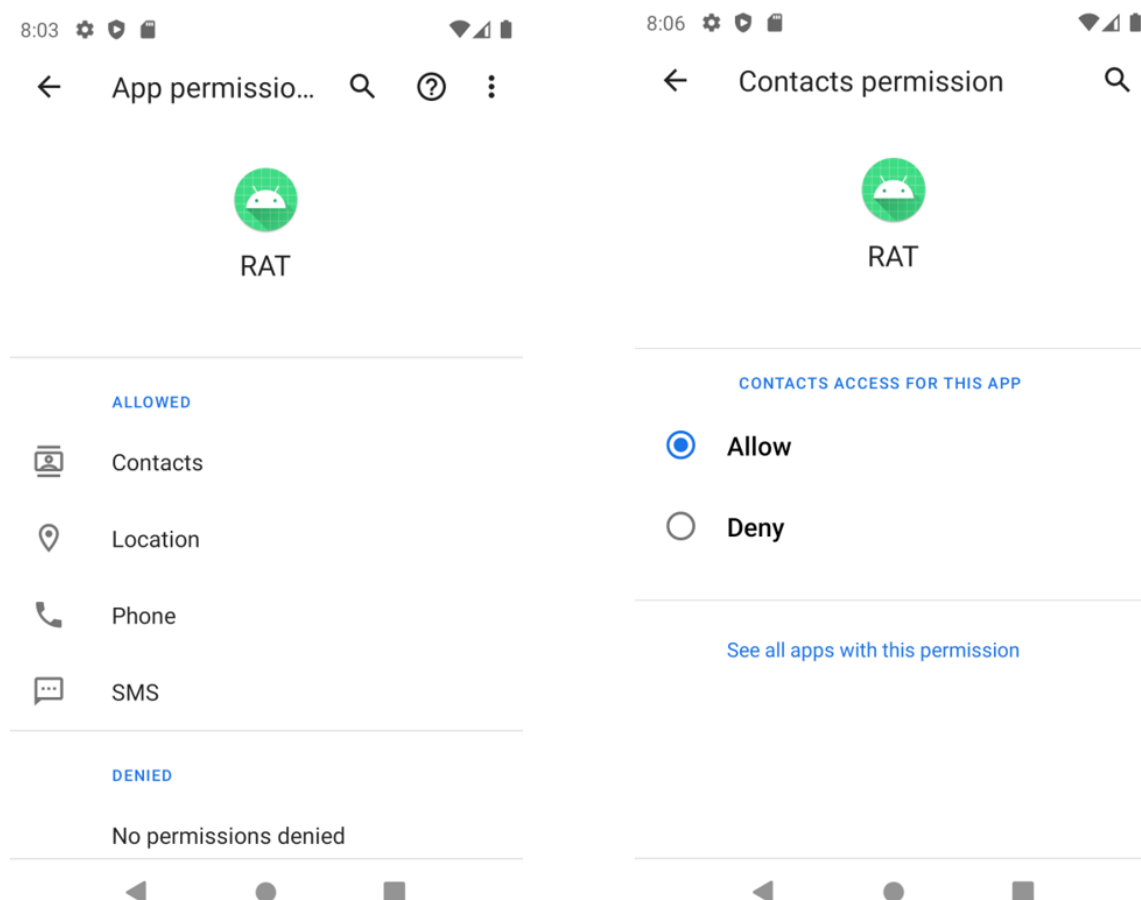


Рисунок 1.2. Просмотр разрешений в настройках системы

Разрешениям приложений также уделяют большое внимание разработчики антивирусных программ. Так, мобильный антивирус от компании Kaspersky даёт более полное описание разрешения, а также описание потенциальных рисков, связанных с этим разрешением (Рис. 1.3). Похожим функционалом обладают и многие другие мобильные антивирусы.

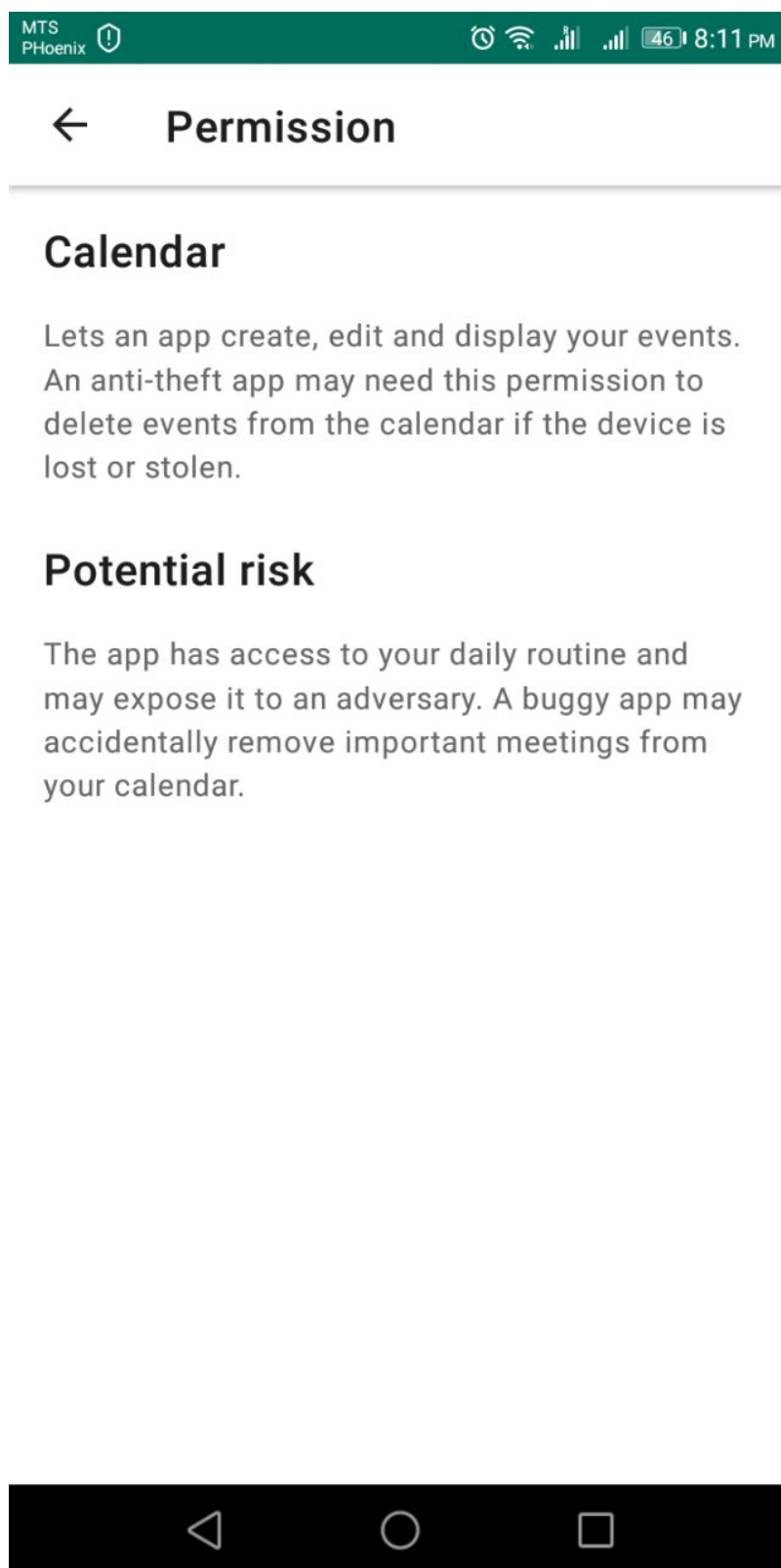


Рисунок 1.3. Описание разрешения на доступ календарю в антивирусе
Kaspersky

1.4. Выводы по разделу 1

Таким образом, стоит отметить, что, несмотря на повышенный интерес к безопасности мобильных приложений и мобильных операционных систем в последние годы, а также на большую проведенную работу, в данной сфере остаётся широкое поле для деятельности.

Эта деятельность заключается в усилении безопасности самих операционных систем, в повышении безопасности мобильных приложений и продвижении принципов безопасной разработки. Немаловажными вопросами являются также привлечение внимания пользователей к вопросам конфиденциальности их данных и разработка методов борьбы со шпионским программным обеспечением без потери функционального богатства операционной системы.

РАЗДЕЛ 2. СОЗДАНИЕ RAT ДЛЯ ANDROID

2.1. Выбор метода связи с сервером

Операционная система Android обладает широким функционалом, который позволяет разработчикам легко реализовывать их наиболее смелые задумки. Но в то же время этот функционал создаёт большое поле деятельности для злоумышленников. Так, в Android существует достаточно много различных способов создать эффективный инструмент удалённого доступа. В основном эти способы разнятся методом установления и поддержания связи с сервером, а также методом обеспечения работы приложения незаметно для пользователя.

Для достижения целей данной работы был выбран способ связи с сервером через Firebase Cloud Messaging. Firebase Cloud Messaging - это система, разработанная компанией Google, для эффективной передачи сообщений от сервера клиентам (одному или многим). Данный выбор мотивирован следующими преимуществами, в сравнении с другими методами:

1) **Простота в использовании.** От разработчика требуется минимум усилий для добавления и настройки FCM в своём приложении. Нет необходимости заботиться о поддержании связи с сервером. Система Firebase Cloud Messaging сама заботится об установке и поддержании связи с сервером.

2) **Отказоустойчивость.** FCM обладает низким процентом отказов и малой задержкой при передаче сообщений. Если сообщение не было передано от сервера к клиенту, чаще всего это значит, что клиент находится вне сети.

3) **Гарантия выполнения работы.** FCM является частью операционной системы, поэтому если сообщение дошло до устройства, то приложение гарантировано исполнит предписанный код (при условии, что время исполнения ограничено несколькими секундами).

Помимо плюсов, безусловно есть и несколько минусов:

1) **Не все устройства имеют доступ к системе FCM.** Сервис Firebase Cloud Messaging является частью набора проприетарных сервисов компании Google и только лицензированные производители смартфонов имеют право включать их в свою операционную систему.

2) **Централизованный контроль Google.** Связь между сервером управления и приложением осуществляется через серверы компании Google (Рис. 2.1). Это значит, что Google может централизованно ликвидировать сеть управляемых устройств.

В целом для злоумышленников минусы данного подхода являются критическими. Они не позволяют построить большую, стабильную сеть, которую сложно вывести из строя. Поэтому они предпочитают использовать другие методы. Тем не менее, данное исследование посвящено не построению стабильной большой сети устройств, а возможности сбора конфиденциальной информации без ведома пользователя.

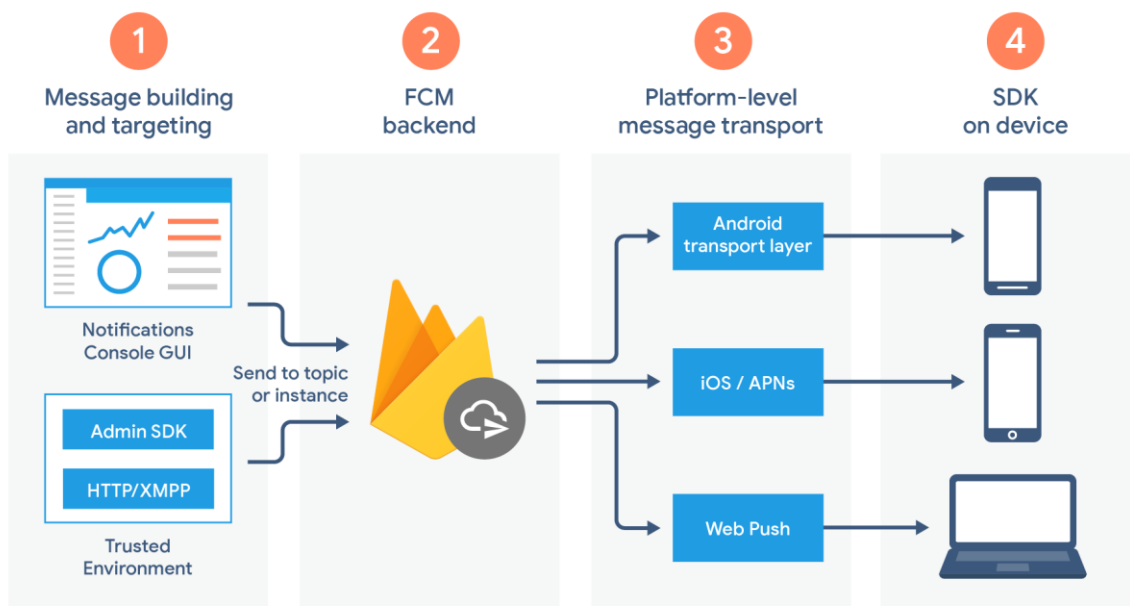


Рисунок 2.1. Архитектура FCM

2.2. Регистрация в системе FCM

Система Firebase Cloud Messaging плотно интегрирована с сервисами Google, поэтому для создания проекта достаточно зарегистрировать почту в сервисе GMail, перейти на сайт `console.firebase.google.com` и следовать инструкциям. Далее нужно перейти в настройки проекта и на вкладке “Service accounts” нажать кнопку “Generate new private key” (Рис. 2.2).

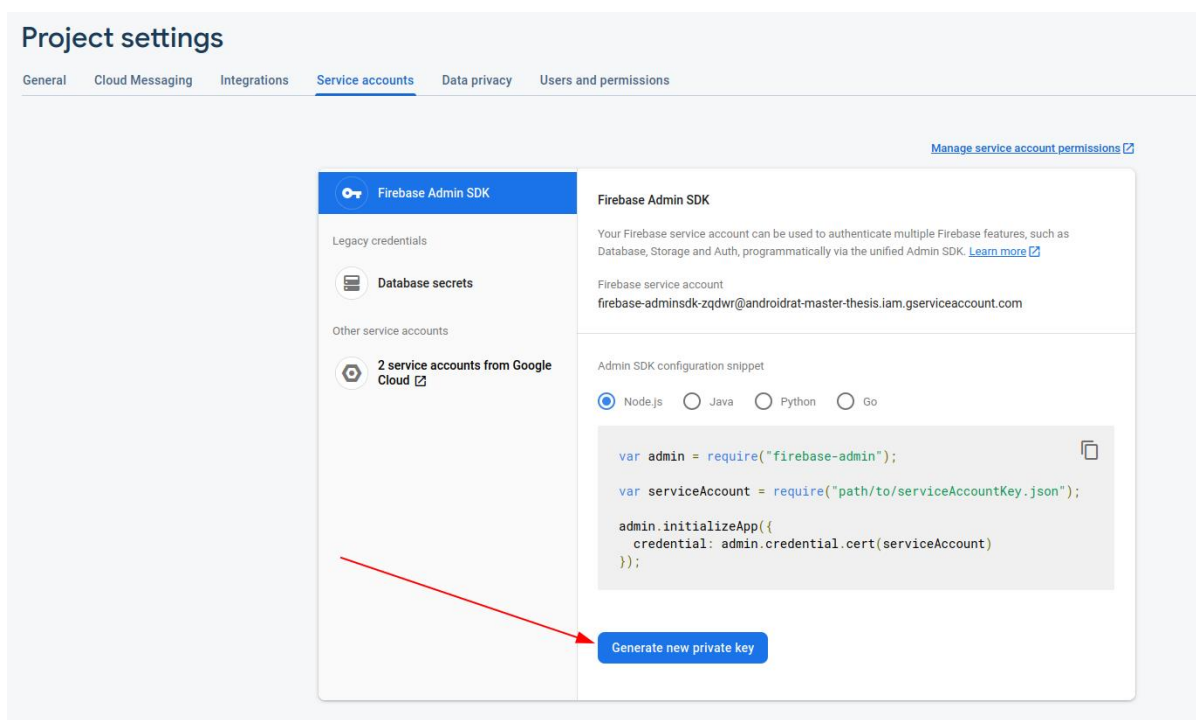


Рисунок 2.2. Создание ключа администратора

После нажатия на кнопку FCM сгенерирует новый ключ администратора и предложит скачать файл в формате JSON. Этот файл в дальнейшем будет использоваться серверной частью инструмента удалённого доступа для подключения к серверу Google.

Далее во вкладке “General” нужно нажать на иконку с логотипом Android для добавления нового приложения в проект (Рис. 2.3).

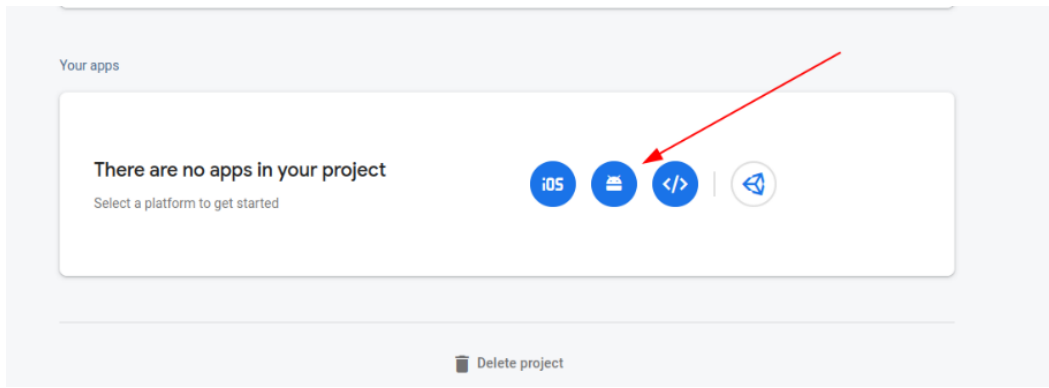


Рисунок 2.3. Добавление Android приложения в проект FCM

Далее, следуя инструкциям нужно скачать файл `google-services.json`, который содержит уникальный идентификатор проекта для FCM, и настроить проект приложения в среде разработки Android Studio (Рис. 2.4). После этого настройку проекта можно считать завершенной и приступать к разработке серверной части.

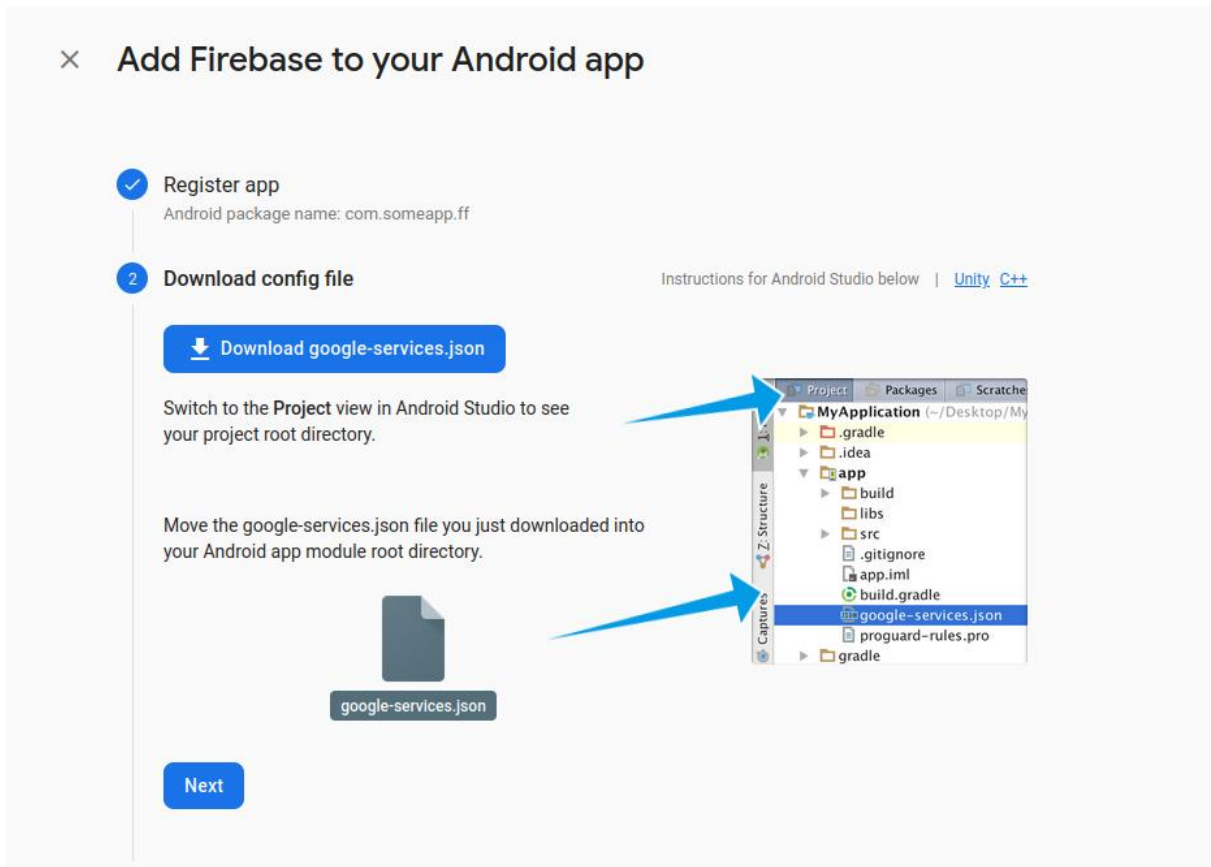


Рисунок 2.4. Инструкция по настройке проекта в Android Studio

2.3. Разработка серверной части

Серверную часть, как и клиентскую, было решено реализовать с помощью языка программирования Kotlin. Таким образом, при необходимости, можно будет использовать код серверной части в клиентском приложении и наоборот без дополнительной конвертации. Кроме того, так как программы на языке Kotlin компилируются в байт-код для виртуальной машины Java, программу можно будет запустить в любой операционной системе с установленной JVM.

В первую очередь после создания проекта необходимо сконфигурировать Firebase Admin SDK для возможности использования функций FCM. Это можно сделать, следуя инструкциям из официальной документации Firebase: <https://firebase.google.com/docs/admin/setup?hl=en>

Сама схема работы серверной части делится на две части. Первая часть будет взаимодействовать с FCM для отправки команд клиентам. Вторая часть достаточно классическая, будет представлять из себя обычный TCP сервер:

- 1) Один поток для ожидания новых подключений.
- 2) Пул потоков, в котором будет осуществляться работа с клиентами.

Таким образом, после получения какой-то команды, клиентское приложение будет открывать TCP соединение, отправлять серверу необходимые данные и закрывать соединение.

Первая часть реализована с помощью одной функции, представленной на рисунке 2.5.

```
fun sendCommand(client: Client, command: String, additionalData: String = "") {
    val msg = Message.builder()
        .putData("command", command)
        .putData("additionalData", additionalData)
        .setToken(client.token)
        .build()
    FirebaseMessaging.getInstance().send(msg)
}
```

Рисунок 2.5. Функция отправки команд клиенту

Реализация второй части также не требует от разработчика больших усилий, благодаря богатому функционалу языка Java. Поток, ожидающий подключения от клиентов, реализован с помощью класса Thread. Основной цикл этого потока представлен на рисунке 2.6.

```

Logger.log(tag, message: "Server started")
running = true
while (running) {
    val client = socket.accept()
    threadPool.submit(ClientThread(WeakReference(referent: this), client))
    sleep(millis: 10)
}

Logger.log(tag, message: "Server stopped")

```

Рисунок 2.6. Поток, ожидающий новые подключения

В качестве пула потоков используется динамический пул из пакета `java.util.concurrent.Executors`. Этот пул будет создавать новые потоки при подключении клиентов, и удалять их, когда работа с клиентом окончена. Класс `ClientThread`, аналогично слушающему потоку, реализован с помощью класса `Thread`. Основной цикл этого потока представлен на рисунке 2.7.

```

Logger.log(tag, message: "Client connected: ${client.remoteSocketAddress}")
val output = client.getOutputStream()
val input = client.getInputStream().bufferedReader()

while (!client.isClosed) {
    val line = input.readLine().split(dataDivider)

    when (line[0]) {
        "connect" -> {...}

        "contacts" -> {...}

        "phone_info" -> {...}

        "sms" -> {...}

        "location" -> {...}

        "finish" -> break
    }

    if (line.isEmpty()) break
    Thread.sleep(millis: 25)
}

Logger.log(tag, message: "Client disconnected: ${client.remoteSocketAddress}")

```

Рисунок 2.7. Поток, взаимодействующий с клиентом

Далее остаётся только построить пользовательский интерфейс и связать его с вышеприведённым кодом. Принципиальной разницы между инструментами для построения интерфейсов нет, данная программа может использовать как консольный командный интерфейс, так и расширенный графический интерфейс. В данной работе для создания графического интерфейса пользователя был использован декларативный набор инструментов для языка Kotlin - TorandoFX, основанный на JavaFX. Основное окно программы приведено на рисунке 2.8.

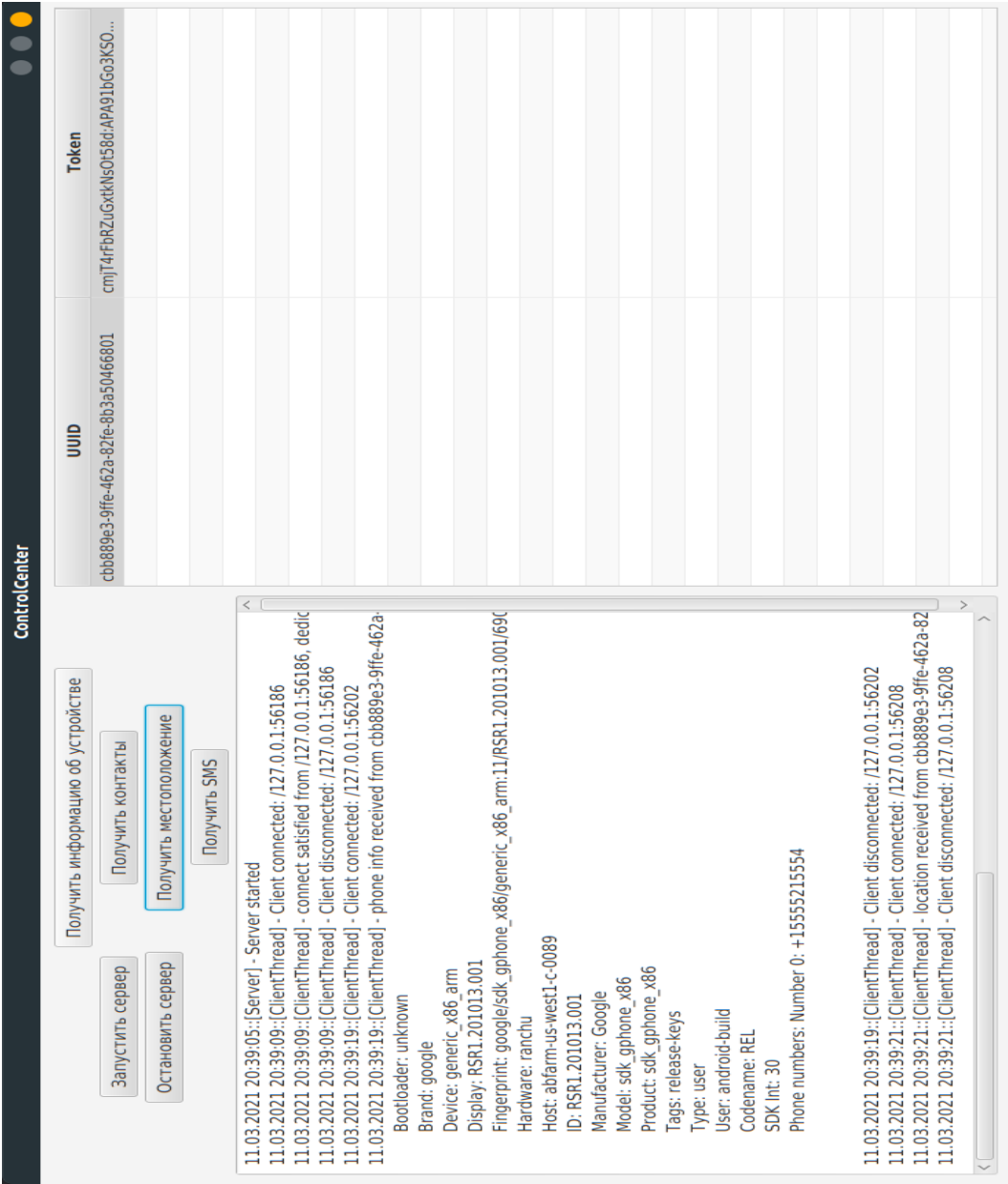


Рисунок 2.8. Основное окно программы сервера

Кнопки “Запустить сервер” и “Остановить сервер” запускают и останавливают основной поток, ожидающий подключение клиентов соответственно.

Таблица в правой стороне окна отображает список доступных клиентов. В этой таблице указан уникальный идентификатор клиента, который присваивается каждому клиенту при подключении. Клиентское приложение будет сохранять присвоенный ему идентификатор и в дальнейшем добавлять ко всем сообщениям, отправляемым серверу.

Текстовое поле под кнопками предназначено для вывода сообщений о запуске и остановке сервера, о подключаемых клиентах и о принятых от клиентов данных.

В правом столбце кнопок каждая кнопка используется для отправки клиенту определённой команды:

1) “Получить информацию об устройстве” - запрашивает у клиента основные данные об устройстве, такие как: производитель, версия операционной системы, версия аппаратного обеспечения, номер сборки, номер телефона и т.д.

2) “Получить контакты” - запрашивает у клиента список контактов в виде таблицы “имя - номер телефона”.

3) “Получить местоположение” - получает от клиента координаты последнего известного местоположения пользователя в виде долготы и широты, а также точность определения координат и время получения данных.

4) “Получить SMS” - получает от клиента все полученные и отправленные сообщения, а также сообщения, сохранённые в черновиках.

2.4. Разработка клиентского приложения

Обычно злоумышленники создают RAT приложения под видом других приложений. Например, под видом чата или веб-браузера. Популярным также является вариант приложений анти-вор. Эти приложения помогают пользователям искать потерянный или украденный телефон, блокировать его работу и прочее. Несмотря на заявленные добрые намерения, эти приложения также могут использоваться мошенниками для сбора пользовательских данных. Для целей данной работы создавать надёжное прикрытие приложения нет необходимости, поэтому интерфейс приложения включает в себя только поля ввода IP-адреса и порта серверной программы и кнопку подключения (Рис. 2.9).

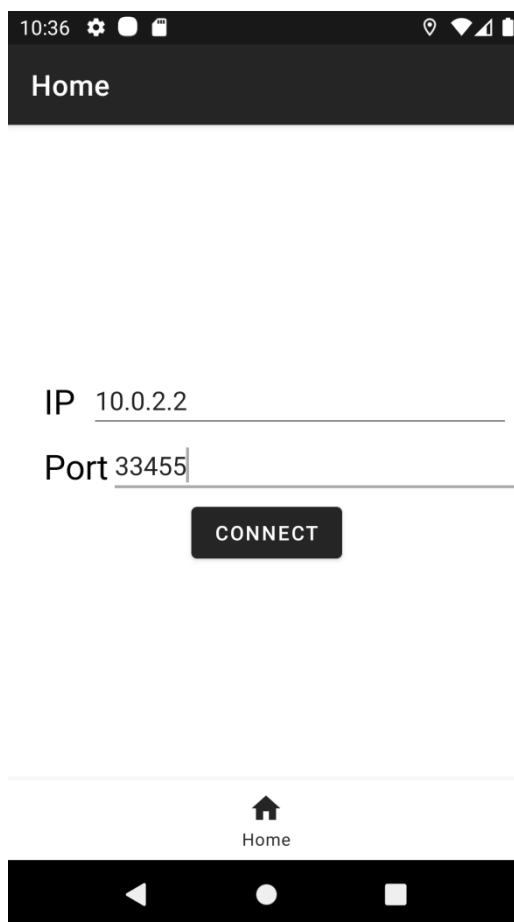


Рисунок 2.9. Интерфейс клиентского приложения

Далее для того, чтобы приложение имело доступ к пользовательским данным, оно должно получить от пользователя разрешения. В операционной системе Android разрешения делятся на несколько категорий:

- обычные разрешения, указываются в файле AndroidManifest.xml приложения (Рис. 2.10). Эти разрешения пользователь предоставляет при установке. Во время установки приложения пользователь может просмотреть список разрешений и прервать установку, если он не согласен предоставлять какие-либо из этих разрешений. К обычным разрешениям относятся, например, доступ к интернету.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.READ_PHONE_NUMBERS"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_BACKGROUND_LOCATION"/>
<uses-permission android:name="android.permission.SET_ALARM"/>
<uses-permission android:name="android.permission.READ_SMS"/>
```

Рисунок 2.10. Указание требуемых разрешений в AndroidManifest.xml

- опасные разрешения - указываются в файле AndroidManifest.xml, а также дополнительно должны быть запрошены во время работы приложения. Зачастую все подобные разрешения запрашиваются при первом запуске приложения (Рис. 2.11), хотя Google рекомендует запрашивать их только при непосредственной необходимости (например, доступ к камере должен запрашиваться только если приложение собирается сделать фото прямо сейчас, а не сразу после запуска приложения). Каждое такое разрешение запрашивается отдельно, и пользователь может либо предоставить его, либо отказаться (в таком случае приложение может потерять часть функций, связанных с этим разрешением). К опасным разрешениям относится доступ к камере, доступ к списку контактов, доступ к сообщениям и т.д.

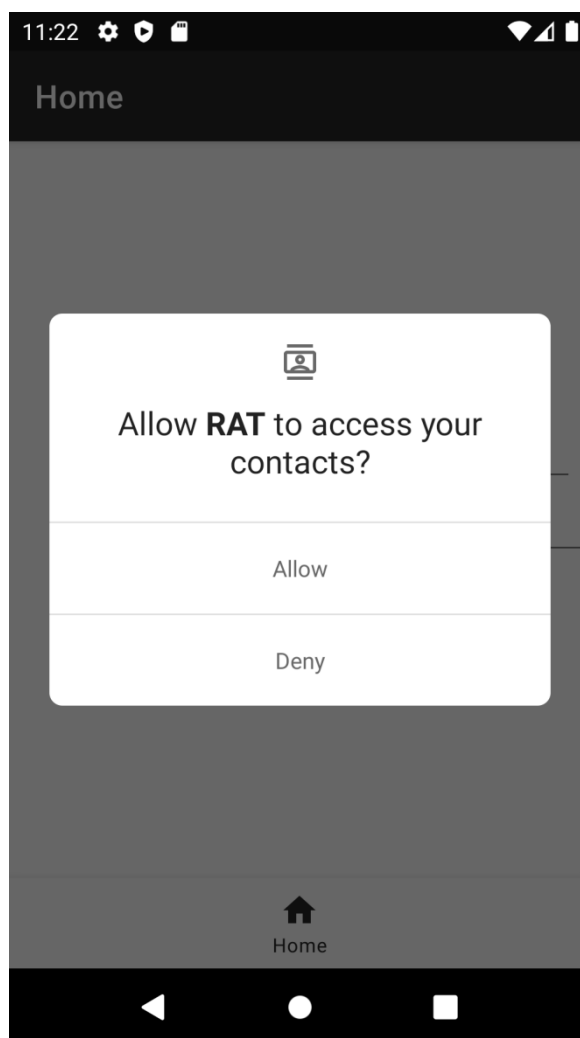


Рисунок 2.11. Запрос опасного разрешения при запуске приложения

- специальные разрешения - подкласс опасных разрешений, которые требуют, чтобы пользователь включил их в настройках приложения. Само приложение не может запросить эти разрешения, а только направить пользователя в настройки. К таким разрешениям относится игнорирование оптимизации батареи, возможность перекрывать окна других приложений и др.
- разрешения по подписи - класс разрешений, который может быть добавлен устанавливаемыми приложениями. Некоторые приложения при установке могут добавить собственные разрешения в этот класс. Далее другое приложение, которое устанавливается на это устройство, может получить новые разрешения от этого приложения, но только если другое приложение

подписано той же подписью.

- системные разрешения - могут быть получены только системными приложениями, т.е. приложениями, установленными производителем устройства.

После получения всех необходимых разрешений нужно настроить работу с FCM для получения команд от сервера. Это реализуется в два шага. В первую очередь нужно получить токен - уникальный идентификатор устройства в среде Firebase Cloud Messaging. Для этого при инициализации приложения требуется отправить запрос на серверы Google (Рис. 2.12). Полученный токен сохраняется во внутренние настройки приложения в ПЗУ.

```

override fun onCreate() {
    super.onCreate()

    context = applicationContext
    Preferences.init(context)

    FirebaseMessaging.getInstance().token.addOnCompleteListener { it: Task<String!>
        | Preferences.set(Preferences.PreferencesField.TOKEN, it.result ?: "")
    }
}

```

Рисунок 2.12. Получение токена FCM

Второй шаг - создание специального сервиса (Рис. 2.13) для получения команд от сервера и новых токенов (выданный токен может меняться время от времени).

При получении команды от сервера выполняется функция `onMessageReceived`, которая в свою очередь вызывает соответствующий каждой команде метод из класса `Interactor`. Каждый из этих методов просто открывает новое TCP соединение с сервером, собирает необходимые данные и отправляет их серверу, далее соединение закрывается. Код функций, собирающих данные с устройства приведён в приложении 1.

Функция `onNewToken` вызывается, когда токен, ранее выданный приложению, изменился. В этом случае приложение записывает новый токен в настройки и отправляет его на сервер.

```
class CommandsService: FirebaseMessagingService() {

    private val tag = "CommandsService"

    override fun onMessageReceived(message: RemoteMessage) {
        super.onMessageReceived(message)

        when (message.data["command"]) {
            "contacts" -> Interactor.sendContacts()
            "phone_info" -> Interactor.sendPhoneInfo()
            "sms" -> Interactor.sendSms()
            "location" -> Interactor.sendLocation()
        }

        Log.w(tag, msg: "Received command: ${message.data["command"]}")
    }

    override fun onNewToken(newToken: String) {
        super.onNewToken(newToken)

        Preferences.set(Preferences.PreferencesField.TOKEN, newToken)
        Interactor.sendToken()
    }
}
```

Рисунок 2.13. Код сервиса для обработки событий FCM

2.5. Тестирование инструмента удалённого доступа

Для проверки того, что приложение устанавливается, беспрепятственно, а также незаметно для пользователя выполняет свои функции были проведены эксперименты с применением устройств, управляемых различными версиями операционной системы Android. Эксперименты проводились по следующему алгоритму:

- 1) Приложение устанавливается и запускается на устройстве.
- 2) Приложению предоставляются необходимые разрешения.
- 3) Производится подключение к серверной программе.
- 4) Приложение закрывается, убирается из недавних приложений и экран устройства гасится. Таким образом устройство переходит в неактивный режим, в котором пользователь непосредственно с ним не взаимодействует.
- 5) С помощью серверной программы на устройство последовательно отправляется каждая команда.

Выполнение команды считается успешным, если устройство отправляет серверу верные данные. Например, при запросе контактов проверяется, что полученные имена и номера телефонов действительно существуют на устройстве. Если устройство не отвечает, присылает пустые или неверные данные, выполнение команды считается проваленным.

В таблице 2.1 указаны результаты экспериментов. Цифры сверху указывают номер версии операционной системы, на которой проводился эксперимент, слева указаны отправляемые команды. На пересечении числа и команды символом “+” указано успешное выполнение команды. Символ “+/-” указывает на то, что команда была выполнена успешно, полученные данные были верными, но неполными. Так, некоторые устройства присылали неполный список контактов и иногда не позволяли получить номер телефона. При этом устройства никак не оповещали пользователя об утечке данных и не блокировали этой утечки в полной мере.

Таблица 2.1

Результаты тестирования

		Версия Android						
		5	6	7	8	9	10	11
Команда	Информация об устройстве	+	+	+	+/-	+/-	+/-	+
	Контакты	+	+	+	+	+	+	+/-
	Местоположение	+	+	+	+	+	+	+
	Сообщения	+	+	+	+	+	+	+

2.6. Выводы по разделу 2

Таким образом, по результатам проведённых исследований можно сделать несколько выводов:

1) Современные инструменты позволяют злоумышленникам создавать приложения для удалённого доступа к устройству с широким функционалом без значительных усилий и навыков, перекладывая основную работу на уже существующие сервисы. Обе - серверная и клиентская часть приложения были записаны менее чем в 2000 строк программного кода.

2) Операционная система не блокирует и не уведомляет пользователя о том, что какое-либо из приложений получает доступ к критически важным данным. Так приложение с соответствующим разрешением может считать и отправить на удалённую машину все полученные и отправленные пользователем SMS даже если пользователь не пользуется ни приложением, ни самим телефоном в данный момент.

3) Несмотря на активную работу разработчиков операционной системы Android над усилением контроля за приватностью пользователей, получение разрешений и доступ к критически важным пользовательским данным не был существенно затруднён ни на старой, пятой, версии системы, ни на самой новой - одиннадцатой.

РАЗДЕЛ 3. ДЕТЕКТИРОВАНИЕ RAT

3.1. Классификация разрешений

Как уже было замечено выше, детектирование и блокировка шпионского программного обеспечения, такого как инструменты удалённого доступа, без вмешательства человека практически невозможна ввиду специфики работы таких программ. Конечное решение по тому является ли приложение вредоносным или нет должен выносить пользователь. И, несмотря на большие усилия разработчиков операционной системы Android и мобильных антивирусов направленные на привлечения внимания пользователей к проблеме и повышение их осведомлённости о процессах, происходящих в операционной системе, ни операционная система, ни антивирусы не предлагают пользователям функционал по автоматическому анализу приложений на предмет сбора этими приложениями данных. В ходе данного исследования был разработан метод автоматического анализа приложений для поиска потенциальных инструментов удалённого доступа и написано приложение, реализующее этот метод.

Для оценки приложений был выбран единственный критерий - набор, предоставленных им разрешений. В ходе работы были проанализированы все разрешения, которые может получить приложение, и каждому из них был присвоен коэффициент опасности. Коэффициент рассчитывается на основании двух факторов. Во-первых, учитывается степень опасности разрешения по классификации самой компании-разработчика операционной системы - Google (Рис. 3.1). Во-вторых, были проанализированы публикации исследователей безопасности (в том числе крупных антивирусных компаний) для определения того, какие данные может собирать приложение имея то или иное разрешение, и какой ущерб могут нанести злоумышленники, заполучив эти данные.

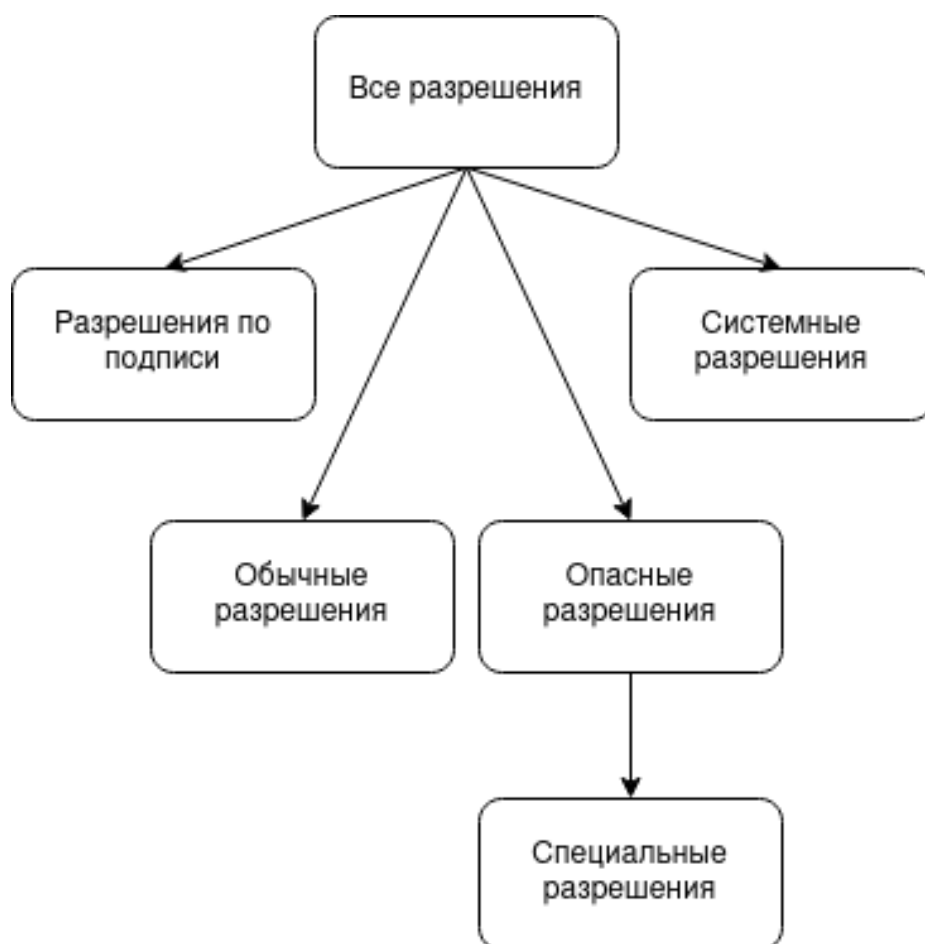


Рисунок 3.1. Классификация разрешений в Android по версии Google

По собранным данным в ходе данной работы была создана новая система классификации разрешений. Все разрешения были разбиты на три класса: неопасные, опасные и критические. Свойства этих классов описаны в таблице 3.1. В первую очередь все разрешения строго разбиваются на эти три класса, коэффициент опасности приложений внутри класса определяется уже относительно других членов этого класса. Кроме того, коэффициенты могут корректироваться по результатам эксперимента для оптимизации анализа. Так, в класс неопасных разрешений попали разрешения на доступ к интернету, доступ к Bluetooth и NFC и подобные им. К классу опасных разрешений относятся доступ к камере, получение местоположения с маленькой точностью и др. Критически опасными считаются получение местоположения

с большой точностью, доступ к файловой системе устройства, возможность чтения SMS/MMS и контактов и прочие.

Таблица 3.1

Предлагаемая новая классификация приложений

Класс	Диапазон коэффициентов	Описание
Неопасные	0 - 10	Разрешения, позволяющие приложению собирать минимальное количество информации, или не позволяющее собирать информацию вовсе.
Опасные	11 - 14	Разрешения, позволяющие собирать конфиденциальную информацию о пользователе, сбор которой даёт злоумышленникам ограниченный набор возможностей.
Критические	15 - 20	Разрешения, позволяющие собирать критически важную информацию, дающую злоумышленникам большие возможности для проведения последующих атак.

3.2. Математическая модель анализа приложений

Теперь, когда для каждого разрешения определён некий коэффициент, мы можем строить математические модели для анализа приложений на основе предоставленных им разрешений.

Безусловно, выбор правильной модели является критически важной составляющей успеха автоматизированного анализатора. На основе вышеприведённой классификации можно предложить два различных подхода к анализу:

1) имея численный коэффициент опасности каждого разрешения, можно просто просуммировать коэффициенты всех разрешений, получив таким образом общий индекс опасности приложения (см. формулу 3.1);

$$K_1 = \sum_i p_i, \quad (3.1)$$

где p_i - коэффициент i -го разрешения.

2) вычислить коэффициент опасности на основе количества критических, опасных и неопасных разрешений (см. формулу 3.2);

$$K_2 = \alpha C + \beta D + \gamma N, \quad (3.2)$$

где C - количество критических разрешений;

D - количество опасных разрешений;

N - количество неопасных разрешений;

α , β и γ - некоторые постоянные коэффициенты.

Оба подхода обладают преимуществами и недостатками. Так, первый подход, очевидно, показывает сколько всего данных может собирать приложение. Но, с другой стороны, он совершенно не принимает во внимание важность этих данных. Приложения с большим количеством неопасных разрешений смогут перевесить приложения с малым количеством, но критических разрешений. Второй подход, с другой стороны, теряет количественную оценку, сосредотачивая внимание на качественной.

Становится очевидно, что использование исключительно одного из этих подходов делает анализ односторонним, чего никак нельзя допускать. Поэтому в данной работе оба подхода были объединены. Используя два индекса, созданный автоматический анализатор сможет предоставлять пользователю наиболее полную информацию о приложении и давать делать выбор о том, стоит ли ему доверять этому приложению.

При этом нужно обратить внимание на то, что использование двух численных индексов может запутать пользователя, что плохо отразится на опыте использования анализатора. К тому же индекс по второму методу, являющийся численной интерпретацией качественной оценки, даёт весьма размытое понимание того, что он значит. На этом основании было принято решение разбить приложения на три класса аналогично тому, как разделены сами разрешения. Определение класса приложения можно производить по формуле 3.3 на основе второго индекса и предоставлять пользователю только информацию о том, к какому классу относится приложения, не запутывая его дополнительным индексом.

$$S = \begin{cases} 0, & K_2 < p_1 \\ 1, & p_1 \leq K_2 < p_2, \\ 2, & p_2 \leq K_2 \end{cases} \quad (3.3)$$

где p_1, p_2 - выбранные пороги, причём $p_1 < p_2$.

Аналогично классификации разрешений, если приложение имеет класс 0, значит оно не оперирует важными данными вовсе, или оперирует небольшим количеством важных данных. Класс 1 подразумевает умеренное использование конфиденциальных данных и класс 2 - большое количество важных данных.

Точно рассчитать пороги p_1, p_2 , а также коэффициенты α, β и γ не представляется возможным, поэтому эти числа будут необходимо экспериментально выбирать и корректировать для наилучшего результата.

Тем не менее, существуют некоторые факты о программах удалённого доступа и их различии от обычных приложений:

1) Шпионские программы зачастую требуют очень много именно критических разрешений. В тоже время разработчики обычных приложений стараются использовать минимальное количество разрешений действительно необходимых для корректной работы приложения;

2) Обычные программы могут использовать достаточно много опасных разрешений, но редко используют большое количество критических. Обычно для обеспечения даже самого широкого функционала приложения достаточно получить 2 - 3 критических разрешения.

На их основе можно выбрать некое первое приближение коэффициентов. Очевидно, что между коэффициентами α и β должен существовать большой разрыв, чтобы большое количество опасных разрешений не перетягивало приложение в класс 2 вне зависимости от количества критических разрешений. Также разрыв между β и γ должен быть, вероятно, ещё больше, чтобы приложение, использующее неопасные разрешения без доступа к опасным или критическим, не попадало в класс 1. В качестве начальных значений коэффициентов были выбраны следующие:

$$\alpha = 3, \beta = 0.75, \gamma = 0.05$$

Пороги p_1 и p_2 выбраны следующие:

$$p_1 = 6 \quad p_2 = 4$$

3.3. Создание приложения

Создание приложения-анализатора во многом схоже с созданием RAT. Пользовательский интерфейс приложения состоит из двух экранов:

1) Общий список приложений. Здесь указаны только названия приложений, их количественный индекс опасности K1, вычисленный по формуле 3.1. Дополнительно число индекса имеет определённый цвет, указывающий на класс приложения в соответствии с индексом K2 (см. формулу 3.2). Красный соответствует классу 2, желтый - классу 1 и зелёный - классу 0. Для удобства приложения отсортированы по индексу K1.

2) Детальный обзор приложения. При нажатии на любое приложение из списка на первом экране пользователь попадает на следующий экран, на котором также указаны имя пакета и индекс опасности. Дополнительно здесь указано имя пакета приложения и список разрешений, предоставленных приложению, отсортированных по коэффициенту опасности. Для каждого разрешения также указано описание, объясняющее зачем приложение может использовать это разрешение и какие данные оно может собирать.

Начинать следует, конечно, с создания первого экрана. Разметка интерфейса первого экрана имеет крайне лаконичный вид (Рис. 3.1) и состоит из корневого контейнера ConstraintLayout и специального элемента для отображения списков RecyclerView.

```

1  <?xml version="1.0" encoding="utf-8" ?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      tools:context=".main.MainActivity">
8
9      <androidx.recyclerview.widget.RecyclerView
10         android:id="@+id/rvApps"
11         android:layout_width="match_parent"
12         android:layout_height="match_parent"
13         android:overScrollMode="ifContentScrolls"
14         android:padding="5dp"/>
15
16  </androidx.constraintlayout.widget.ConstraintLayout>

```

Рисунок 3.1 Разметка интерфейса первого экрана

Каждый элемент списка также лучше всего вынести в отдельный элемент интерфейса `AppView`, включающий в себя все необходимые элементы, вместо создания каждого из них в коде приложения. Каждая строка в списке состоит из двух текстовых полей для отображения названия приложения и его индекса опасности (Рис. 3.2). Также для этого элемента интерфейса требуется создать соответствующий Kotlin класс, код которого приведён в приложении 2.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      android:layout_width="match_parent"
6      android:layout_height="wrap_content"
7      android:padding="10dp"
8      android:background="@drawable/background_app_view">
9
10     <TextView
11         android:id="@+id/tvAppName"
12         android:layout_width="wrap_content"
13         android:layout_height="wrap_content"
14         android:textSize="14sp"
15         android:textColor="@color/black"
16         android:text="Application name"
17         app:layout_constraintLeft_toLeftOf="parent"
18         app:layout_constraintTop_toTopOf="parent"
19         app:layout_constraintBottom_toBottomOf="parent"
20         app:layout_constraintRight_toLeftOf="@id/tvIndex"/>
21
22     <TextView
23         android:id="@+id/tvIndex"
24         android:layout_width="wrap_content"
25         android:layout_height="wrap_content"
26         android:textSize="16sp"
27         android:textColor="@color/black"
28         android:text="100"
29         app:layout_constraintRight_toRightOf="parent"
30         app:layout_constraintTop_toTopOf="parent"
31         app:layout_constraintBottom_toBottomOf="parent"
32         app:layout_constraintLeft_toRightOf="@id/tvAppName"/>
33
34 </androidx.constraintlayout.widget.ConstraintLayout>

```

Рисунок 3.2. Разметка отдельного элемента списка приложений

Для удобной инкапсуляции необходимых данных о каждом приложении также был создан класс `AppModel` (Рис. 3.3), в котором хранятся имя приложения, имя пакета, список полученных им разрешений и индексы опасности приложения K_1 и K_2 . Для каждого приложения теперь можно создавать объект этого класса и использовать этот объект для инициализации

AppView. Так, во время инициализации AppView цвет текста индекса K_1 устанавливается в соответствии с индексом K_2 и порогами p_1 и p_2 (Рис. 3.4).

```
data class AppModel(
    val id: Int,
    val name: String,
    val packageName: String,
    val permissions: List<String>,
    val index1: Int,
    val index2: Float
)
```

Рисунок 3.3. Класс AppModel

```
when {
    value.index2 > 6 -> tvIndex.setTextColor(ContextCompat.getColor(context, R.color.red_secret_data))
    value.index2 > 4 -> tvIndex.setTextColor(ContextCompat.getColor(context, R.color.orange_sensitive_data))
    else -> tvIndex.setTextColor(ContextCompat.getColor(context, R.color.green_not_secret_data))
}
```

Рисунок 3.4. Установка цвета текста в соответствии с индексом K_2

Далее необходимо описать инициализацию первого экрана в коде приложения. Эта инициализация включает в себя сбор списка всех установленных приложений, сбор их разрешений, вычисление индексов опасности для каждого приложения, создание моделей всех приложений и создание списка приложений на основе этих моделей. Вычисление индексов K_1 и K_2 приведено на рисунке 3.5. Код сбора списка установленных приложений и их разрешений находятся в приложении 3.

```
private fun calculateIndexForPermissions(permissions: List<String>): Pair<Int, Float> {
    var dangerousCount = 0
    var sensitiveCount = 0
    var regularCount = 0

    var index1 = 0
    for (p in permissions) {
        val weight = PermissionsHelper.getPermissionWeight(p)
        when {
            weight > 15 -> dangerousCount++
            weight > 10 -> sensitiveCount++
            weight > 0 -> regularCount++
        }
        index1 += weight
    }

    val index2 = 3f * dangerousCount + 0.75f * sensitiveCount + 0.05f * regularCount
    return Pair(index1, index2)
}
```

Рисунок 3.5. Код вычисления индексов K_1 и K_2

На рисунке 3.6 представлен итоговый вид первого экрана.



Viber	180
Telegram	133
Альфа-Банк	107
VK	107
Duolingo	64
Домофон	59
Yandex.Maps	56
Office	54
Firefox	53
ЦРБ Онлайн	44
Yandex.Mail	43
Robot36	43
weawow	40
Firefox Focus	35

Рисунок 3.6. Вид первого экрана приложения

Здесь можно видеть большое количество ложных срабатываний анализатора. Так, приложения Viber, Telegram, Альфа-Банк, VK и ЦРБ Онлайн были отнесены к приложениям класса 2, то есть к приложениям с наибольшей опасностью, являющимся шпионскими программами с большой вероятностью.

Создание второго экрана производится по аналогичному сценарию, что и создание первого. В частности, для отображения списка разрешений для приложения используется тот же контейнер, что и для отображения списка

приложений на первом экране. Каждый элемент списка разрешений выглядит аналогично элементу списка приложений за одним исключением - дополнительное текстовое поле, содержащее краткое описание для каждого разрешения. Разметка элемента списка разрешений и код, соответствующего Kotlin класса приведены в приложении 4. Данные о разрешениях инкапсулированы в классе `PermissionModel` (Рис. 3.7) и составляют: строковый идентификатор разрешения, вес разрешения согласно классификации, предложенной в пункте 3.1, описание разрешения.

```
data class PermissionModel(
    val constant: String,
    val weight: Int,
    val description: String
)
```

Рисунок 3.7. Модель разрешения

Вид второго экрана приведён на рисунке 3.8.

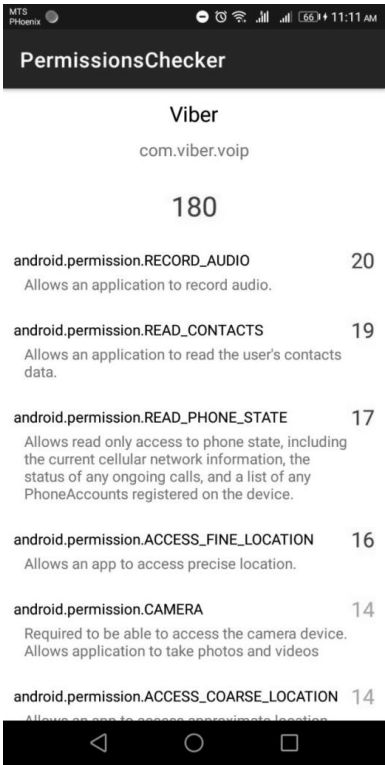
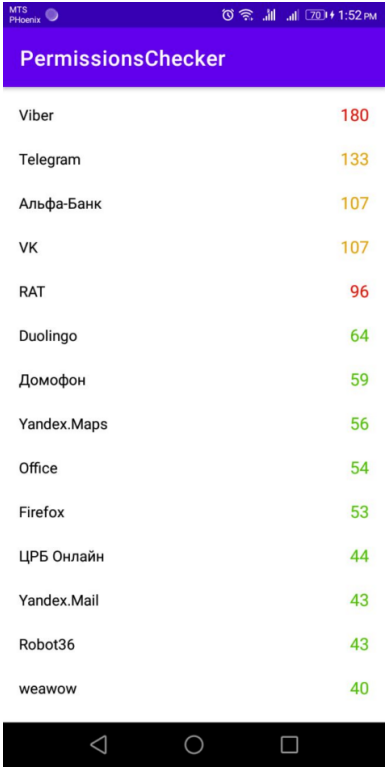


Рисунок 3.8. Вид второго экрана приложения

3.4. Проведение экспериментов и корректировка коэффициентов

Перед приложением-анализатором ставились две задачи. В первую очередь оно должно отмечать шпионские программы как критически опасные, то есть относящиеся к классу 2. Однако в то же время обычные приложения не должны попадать в этот класс вовсе. То есть необходимо минимизировать шанс ложного срабатывания, сохраняя при этом достаточную чувствительность. Так, на рисунке 3.6 мы видим большое количество ложных срабатываний на приложениях, которые на самом деле шпионскими не являются.

Для корректировки значений коэффициентов на устройство было установлено приложение, разработанное в разделе 2, и коэффициенты изменялись до тех пор, пока не удалось достичь удовлетворительных результатов анализа. На рисунке 3.9 приведены результаты анализа с оптимальными коэффициентами.



PermissionsChecker	
Viber	180
Telegram	133
Альфа-Банк	107
VK	107
RAT	96
Duolingo	64
Домофон	59
Yandex.Maps	56
Office	54
Firefox	53
ЦРБ Онлайн	44
Yandex.Mail	43
Robot36	43
weawow	40

Рисунок 3.9. Результаты анализа с оптимальными коэффициентами

Здесь видно, что приложения Telegram, VK, Альфа-Банк и ЦРБ Онлайн больше не относятся ко второму классу опасности как ранее. Они перешли в класс 1, а приложение ЦРБ Онлайн и вовсе приобрело класс опасности 0. В то же время приложение RAT, являющееся созданной ранее в данной работе шпионской программой, попадает в класс критически опасных приложений. При этом его индекс K_1 меньше, чем у вышестоящих приложений. Таким образом анализатор успешно разделяет количественный показатель от качественного.

Таким образом коэффициенты α , β и γ для формулы (3.2) равны 1.35, 0.25 и 0.075 соответственно, а пороги p_1 и p_2 для формулы (3.3) равны 5.5 и 3.

Также можно заметить, что приложение Viber, которое не является шпионской программой, всё ещё ложно определяется как критически опасное. Это связано с тем, что Viber запрашивает слишком много критических разрешений (Рис. 3.10). Корректировка коэффициентов для того, чтобы приложение Viber перестало определяться как критическое привело бы к очень слабой чувствительности анализатора.

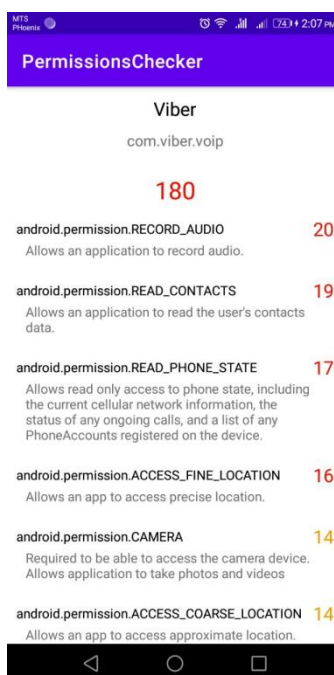


Рисунок 3.10. Список критических разрешений приложения Viber

Анализатор также был проверен на других шпионских программах. На рисунке 3.11 можно видеть, что анализатор успешно относит программы System Framework, Backup и Device Health к критически опасным программам. Эти программы, как и прочие приложения удалённого доступа, предлагают широкий функционал, которым объясняют такой обширный доступ к пользовательским данным. В то же время пользователи не имеют гарантии того, что их данные не будут отправляться злоумышленникам.

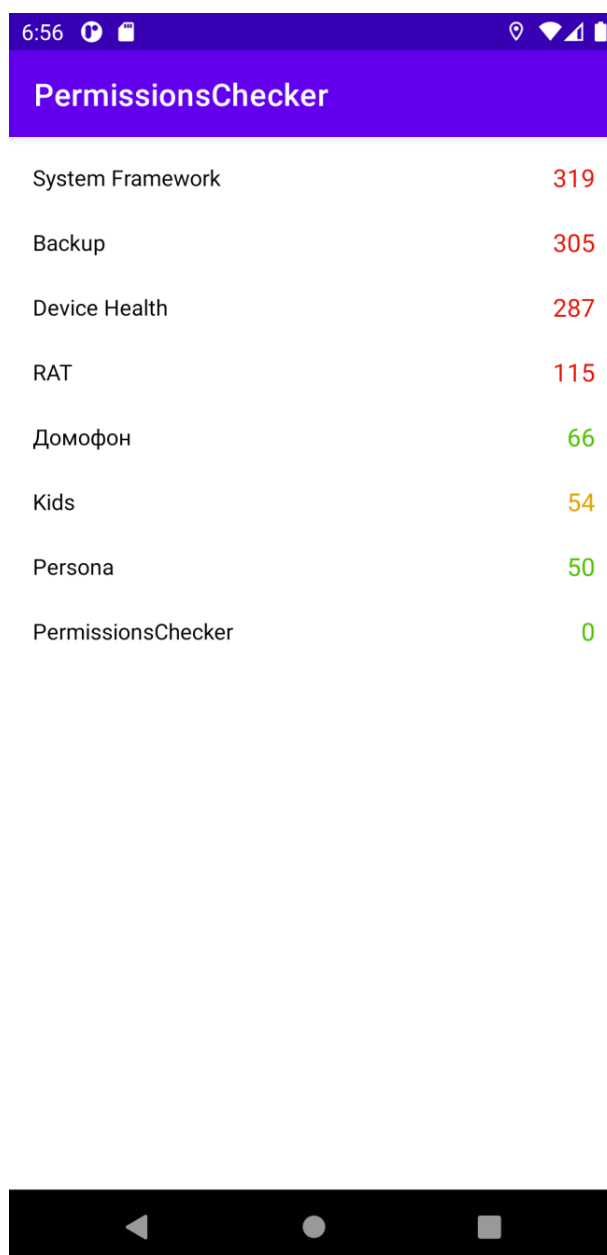


Рисунок 3.11. Анализ других шпионских программ

3.5. Выводы по разделу 3

В ходе данной работы было разработано приложение-анализатор, которое способно на основе анализа полученных другими приложениями разрешений выявить потенциально опасные программы. Приложение успешно определяет шпионские программы и обладает низким процентом ложных срабатываний.

На основе исследования видно, что анализируя исключительно список разрешений, предоставленных приложению можно с большим успехом находить инструменты удалённого доступа и потенциальные утечки пользовательских данных. При этом сам приложению вовсе не требуется никаких разрешений, имеет малый вес и высокую производительность, что даёт ему преимущество относительно больших антивирусных комплексов.

Анализатор предоставляет пользователям информацию о приложениях в удобном и понятном виде. Используя приложения, пользователи могут оценить количество и качество данных, собираемых каждым приложением и сделать вывод о том, стоит ли им доверять данные данному приложению или им необходимо отказаться от его использования.

РАЗДЕЛ 4. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ОБМЕНА ДАННЫМИ ПО ЗАЩИЩЕННОМУ КАНАЛУ

4.1. Протокол установления связи

Выводы вышеизложенных исследований были учтены при разработке безопасного мобильного приложения для обмена данными по зашифрованному каналу. Приложение стало частью большой системы контроля доступа. Наиболее важным элементом данной системы является протокол установления безопасного канала передачи данных.

Общая схема подключения установления защищенного соединения приведена на рисунке 4.1.



Рисунок 4.1. Схема установления соединения приложения с сервером

Подробный алгоритм выглядит следующим образом:

- 1) Приложение отправляет серверу имя пользователя, под которым оно намерено авторизоваться.
- 2) Сервер, проверив, что такой пользователь у него зарегистрирован

отвечает либо ошибкой (если такого пользователя не существует), либо отправляет обратно случайное число.

3) Получив случайное число, приложение использует его в качестве соли для вычисления SHA-256 хеша от пароля. Вычислив хеш оно отправляет его серверу.

4) Сервер в свою очередь также вычисляет хеш и проверяет его с полученным от клиента. Если хеши совпали, пользователь считается авторизованным.

5) Всё последующее общение сервера с приложением шифруется с помощью пароля пользователя алгоритмом AES-256 в режиме CBC.

Такая схема обладает как преимуществами, так и недостатками. Преимуществом является очень быстрая установка соединения в сравнении с тяжеловесными асимметричными алгоритмами. Для создания канала связи достаточно отправить 4 небольших пакета с данными, а вычисление SHA-256 от пароля в 16 символов занимает очень мало времени. В дополнение такая схема защищена от атаки “человек посередине”, потому что ни чтение, ни изменение данных не предоставит злоумышленнику никаких преимуществ.

С другой стороны, при такой схеме существует необходимость хранить пароль пользователя на сервере в открытом виде. Этот недостаток перекрывается с помощью других компонентов системы.

После установления соединения приложение позволяет:

- принимать вызовы от посетителей (для вызова используется специальный блок вызова, установленный на периметре объекта, к которому ограничен доступ) и обмениваться с ними аудиоданными;
- удалённо открывать двери во время вызова или во время разговора с посетителем;
- записывать диалоги с посетителями;
- получать текстовые сообщения от администратора системы;
- вести учёт посетителей.

4.2. Функционал приложения

Главный экран приложения представлен на рисунке 4.2. В верхней его части находятся три переключателя для подключения к серверам. Под переключателями расположен список всех полученных от администратора сообщений. Внизу экрана находится панель навигации, позволяющая переключаться между экранами.

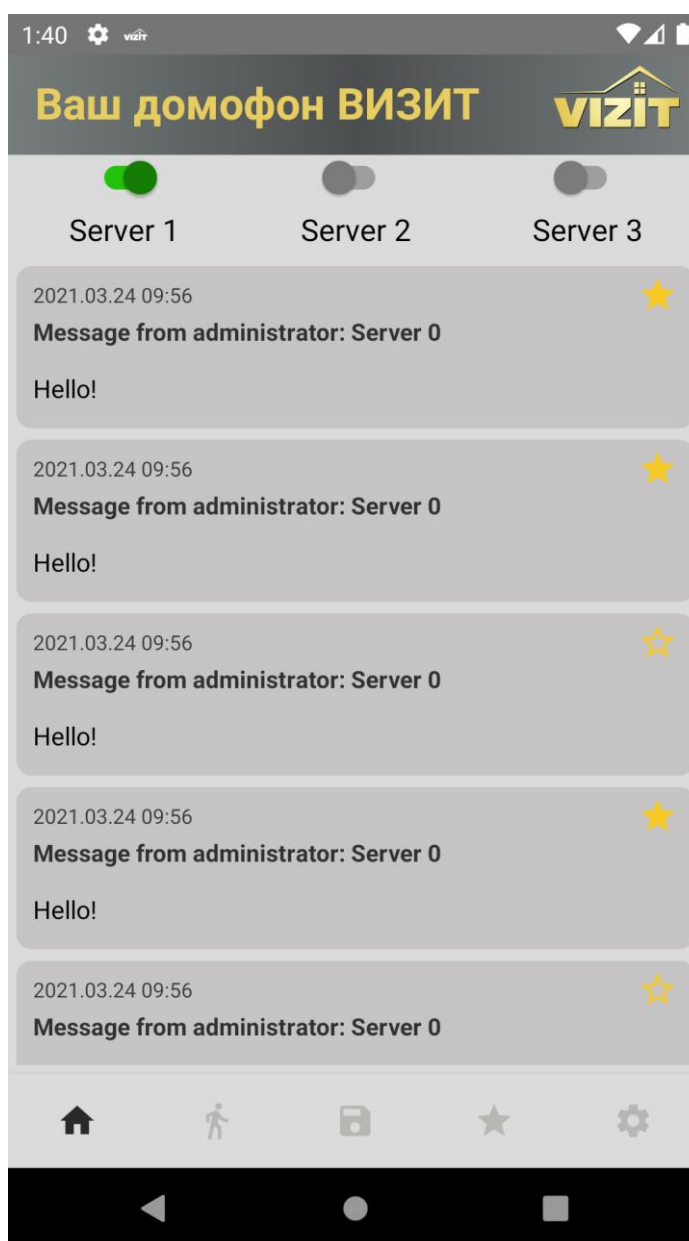


Рисунок 4.2. Главный экран приложения

Список сообщений позволяет удалять сообщения, а также добавлять их в список избранных. При нажатии на кнопку в виде контура звёзды сообщение будет добавлено в закладки, повторное нажатие удаляет сообщение из закладок.

Вторая вкладка на панели навигации открывает экран посетителей (Рис. 4.3). Здесь находится список всех посетителей в базе данных. Посетители делятся на три типа, каждый тип выделяется цветом и иконкой. Посетители, вызов от которых не был принят, имеют красную рамку и иконку уложенной трубки. Посетители, вызов от которых был принят, но дверь не была открыта имеют зелёную рамку и иконку поднятой телефонной трубки. И наконец посетители, для которых была открыта дверь имеют желтую рамку и иконку ключа.

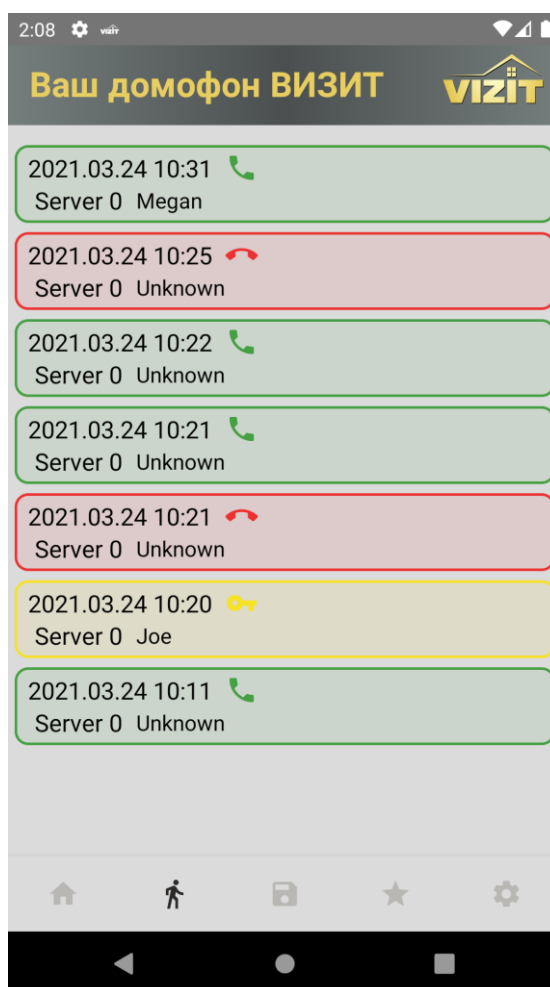


Рисунок 4.3. Экран посетителей

Для каждого посетителя указано время и имя сервера, от которого пришёл вызов. Также существует возможность установить имя посетителя.

Третья вкладка открывает список записанных диалогов (Рис. 4.4). Здесь можно прослушать записанные разговоры, удалить или выгрузить их в открытое хранилище (все диалоги изначально записываются во внутреннее хранилище приложения и не могут быть просмотрены никаким другим приложением). Имя файла диалога включает в себя имя сервера, дату и время, когда разговор имел место быть.

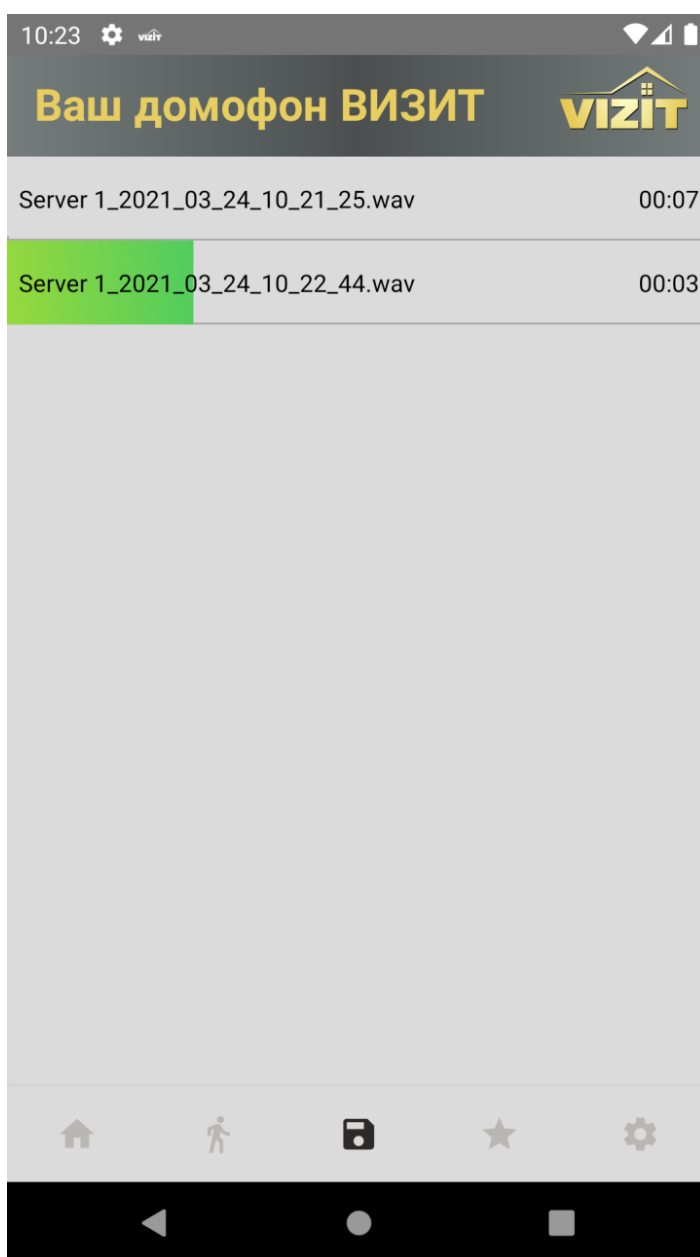


Рисунок 4.4. Экран записанных диалогов

Последние две вкладки ведут к списку закладок и к настройкам (Рис. 4.5). Список закладок полностью повторяет главный экран, за исключением того, что он не содержит переключателей серверов и здесь отображаются только сообщения отмеченные как закладки.

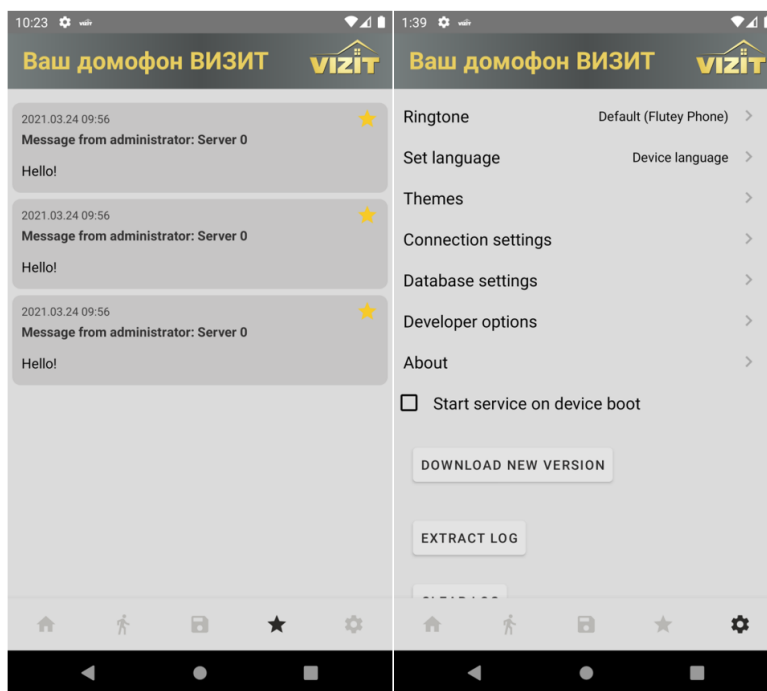


Рисунок 4.5. Экран закладок и экран настроек.

Приложение обладает большим списком настроек. Здесь можно установить мелодию вызова, язык приложения (поддерживаются английский и русский), цветовую тему приложения (на данный момент поддерживаются 4 темы, а также опция автоматического переключения тёмной и светлой тем в зависимости от настроек устройства). В настройках также настраиваются данные для подключения к серверам. Кроме того, есть возможность очистить базу данных, а именно можно:

- удалить все сообщения;
- удалить всех посетителей;
- удалить все записанные диалоги.

Также настройки позволяют выгрузить или очистить лог приложения. Лог используется для отслеживания состояния приложения и обнаружения ошибок. В лог не пишутся никакие конфиденциальные данные, в чём пользователи могут самостоятельно убедиться, выгрузив и прочитав его.

Приложение сохраняет данные лога во внутреннее хранилище устройства. Разработчики не имеют доступа к логу и единственным вариантом его получения является прямая передача лога от пользователя к разработчикам. Таким образом, в отличие от большинства других приложений, разработанное приложение не собирает данные в тайне от пользователей, пользователи имеют возможность самостоятельно просмотреть данные, которые передают разработчикам или не передавать эти данные вовсе.

Несмотря на достаточно широкий функционал, приложение не требует большого количества разрешений. Созданный в разделе 3 автоматический анализатор причисляет приложению низкий индекс опасности и относит его к 0 классу (Рис 4.6).

PermissionsChecker	
RA1	96
Duolingo	64
Домофон	59
Yandex.Maps	56
Office	54
Firefox	53
ЦРБ Онлайн	44
Yandex.Mail	43
Robot36	43
weawow	40
Firefox Focus	35
Мой МТС	34
Notepad	30
QR & Barcode Scanner	30
Snortiv	30

Рисунок 4.6. Анализ созданного приложения

4.3. Выводы по разделу 4

В ходе работы было создано приложение для передачи данных по защищенному каналу как часть системы для контроля доступа. Приложение создано с учётом принципов безопасной разработки. Так, передача данных осуществляется с применением современного шифрования. Пользовательские данные сохраняются во внутреннем защищенном хранилище, особенно чувствительные данные дополнительно шифруются. Приложение не собирает никакой конфиденциальной информации, а его лог не передаётся разработчикам без непосредственного вмешательства пользователя.

Приложение использует минимальное количество опасных и критических разрешений для своей работы, поэтому автоматический анализатор, описанный в разделе 3, относит приложение к нулевому, самому низкому, классу опасности. При этом приложение предоставляет широкие функциональные возможности, дополнительно подтверждая утверждение о том, что использующие большое количество разрешений приложения требуют пристального внимания и могут с большой вероятностью являться шпионскими.

РАЗДЕЛ 5. ОХРАНА ТРУДА

Регулировочные операции: при выполнении регулировочных работ различной РЭА наиболее опасным видом травматизма является поражение электрическим током. Рабочий, выполняющий регулировочные работы, должен соблюдать правило техники безопасности, в частности электробезопасности:

а) все доступные для прикосновения токоведущие части электрооборудования должны быть ограждены;

б) рубильники и выключатели должны быть мгновенного действия;

в) щетки и рубильники должны быть установлены в глухих металлических кожухах, запираются на замок и иметь надпись о применяемом напряжении;

г) ручки, рукояти должны быть сделаны из изолирующих материалов;

д) металлические детали должны быть изолированы от токоведущих частей и заземлены;

е) все электрооборудование, а также оборудование и механизмы, которые могут оказаться под напряжением должны быть надежно заземлены;

ж) работы по ремонту оборудования и механизмов должны производиться только после полного отключения от сети электропитания, на месте работ обязательно вывешивают предупредительные плакаты;

з) ручной инструмент, применяемый при регулировочных работах (отвертки, плоскогубцы, кусачки) должен быть снабжен изолированными ручками;

измерительные приборы должны быть заземлены, соединительные провода и щупы не должны иметь повреждений [23].

Электромонтажные работы: при электромонтажных работах необходимо выполнять следующие правила и рекомендации:

а) Электрическое напряжение выше 40В опасно для жизни. Степень поражения зависит от пути прохождения электрического тока через тело

человека и от силы тока, особенно той его части, которая проходит через сердце. Наиболее опасны пути тока – «рука–нога», «рука-рука». Поэтому при настройке необходимо стараться работать одной рукой в одежде с длинными рукавами, чтобы избежать прикосновения к токоведущим частям обеими руками. Другую руку следует держать за спиной или в кармане и не прикасаться ей к корпусу устройства или другим заземлённым предметам или использовать инструменты с изолированными рукоятками;

б) Любые работы электронного направления нужно стараться вести вдали от водопроводных труб и радиаторов, исключить случайное прикосновение к ним;

в) Заменять детали следует только после отключения прибора от сети, обязательно вынимая вилку шнура питания из сетевой розетки. После отключения источника электропитания необходимо разрядить конденсаторы фильтра питающего напряжения. Нельзя проверять исправность плавких предохранителей в аппаратуре путем их замыкания;

г) Подключать измерительный прибор к высоковольтным цепям можно только при обесточенной аппаратуре, предварительно неоднократно разрядив конденсаторы фильтра. Во время таких измерений щуп, подсоединённый к корпусу устройства, нельзя держать рукой;

Необходимо работать у открытого окна, чаще проветривать помещение. После окончания радиомонтажных работ мыть руки.

Работа с компьютером. Основные правила организации пространства вокруг рабочего места:

При длительном и интенсивном использовании, на поверхности модулей ПК (системный блок, монитор, мышка и т.д.) возникают небольшие разряды тока. Эти частицы активизируются во время прикосновений к ним и приводят к выходу техники из строя. Нужно регулярно использовать нейтрализаторы, увлажнители воздуха, антистатика; вокруг стола не должно быть свисающих проводов, пользователь не должен контактировать с ними; важна целостность корпуса розетки и штепсельной вилки; отсутствие

заземления перед экранного фильтра проверяется с помощью измерительных приборов.

Желательно во время строительных работ в офисе использовать минимальное количество легко воспламеняемых материалов (дерева, пенопласта), а также горючего пластика в изоляции.

Рекомендуется отдавать предпочтение кирпичу, стеклу, металлу и т.д.; помещение должно хорошо вентилироваться и охлаждаться в жаркую пору года. Важен своевременный отвод избыточного тепла от техники.

В случае обнаружения трещины на корпусе или повреждений другого рода, нужно обратиться за помощью в сервисный центр. Это же относится к ПК с неисправным индикатором включения/выключения. предметы на столе не должны мешать обзору, пользованию мышкой и клавиатурой.

Поверхность экрана должна быть абсолютно чистой; на системном блоке не должно находиться никаких предметов, так как в результате вибраций может нарушиться работа устройства. Нужно убедиться в том, что никакие посторонние предметы не мешают работе системе охлаждения.

Запрещается начинать работу в помещениях с повышенной влажностью, а также в случае, если рядом присутствуют открытые источники влаги (лужи, мокрый пол).

Включить технику можно лишь после полного высыхания окружающих предметов. недопустимо часто включать и выключать компьютер в течение рабочего дня без особой нужды. Система просто не справляется с необходимостью быстро сворачивать все процессы [23].

При выполнении работы. Поскольку персональный компьютер обладает всеми свойствами электрического прибора, то на него распространяются основные правила безопасности при взаимодействии с проводниками тока: нельзя размещать какие-либо вещи на проводах, а также самостоятельно менять их расположение без особой нужды; рекомендуется избегать расположения жидкостей рядом с модулями компьютера.

Поэтому кулер с водой или кофейный автомат необходимо размещать в стороне от рабочих мест в офисе. Пользователи должны осознавать опасность потенциального замыкания в случае пролития воды на клавиатуру или системный блок. Нельзя работать на ПК с мокрыми руками; нельзя очищать поверхность компьютера от загрязнений, когда он находится во включенном состоянии; недопустимо снимать корпус любой из составных частей ПК во время его работы.

Кроме того, разбор и ремонт техники имеют совершают только специализированные работники; во время работы на компьютере нельзя одновременно прикасаться к другим металлическим конструкциям, которые стоят на той же поверхности. Это касается отопительных батарей или трубопроводов; в помещении с компьютерами непозволительно курить или употреблять пищу непосредственно на рабочем месте; при ощущении даже незначительного запаха гари, нужно как можно быстрее выключить ПК из сети и обратиться к ответственному за обслуживание компьютерной техники.

ЗАКЛЮЧЕНИЕ

В ходе данной работы были созданы три различных мобильных приложения, каждое из которых использовалось для исследования безопасности мобильных приложений для операционной системы Android, а также самой операционной системы.

С помощью написанного приложения для тестирования уровня защиты операционной системы Android от шпионского программного обеспечения удалось беспрепятственно собирать конфиденциальные данные. Это показало, что операционная система Android не обладает фактически никакой защитой от шпионского программного обеспечения. Выявлением и блокировкой таких приложений вынуждены заниматься сами пользователи.

Написанный на основе предложенной математической модели анализа приложений автоматический анализатор позволил с большой точностью выявлять потенциально опасные приложения.

Разработанный алгоритм аутентификации с установлением безопасного канала связи был внедрён на предприятии ФИРМА «МДЛ» при разработке нового программного обеспечения.

Предварительные результаты работы были опубликованы в статье “Алгоритм аутентификации пользователя с созданием безопасного канала передачи данных в коммуникационных системах на основе протокола UDP” в журнале “Вестник Донецкого национального университета”, серия Г, технические науки, 2020, номер 4, стр. 12 (приложение 5).

Представленные результаты могут быть предложены в качестве рекомендаций разработчикам приложений для государственных структур и частных предприятий Донецкой Народной Республики, связанных с информационной безопасностью.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Elenkov N., Android Security Internals / N. Elenkov. - San Francisco, USA: No Starch Press, Inc., 2014. - 434 p.
2. Android Hacker's Handbook / Joshua J. Drake [и др.]. - Indianapolis, USA: John Wiley & Sons, Inc., 2014. - 577 p.
3. Satish Bommisetty, Practical Mobile Forensics / Satish Bommisetty, Rohit Tamma, Heather Mahalik.- Birmingham, UK: Packt Publishing Ltd., 2014. - 503 p.
4. Keith Makan, Android Security Cookbook / Keith Makan, Scott Alexander-Brown. - Birmingham, UK: Packt Publishing Ltd., 2013. - 350 p.
5. Mobile App Hacker's Handbook / Dominic Chell [и др.]. - Indianapolis, USA: John Wiley & Sons, Inc., 2015. - 535 p.
6. Android Malware And Analysis / Ken Dunham [и др.]. - Boca Raton, USA: Taylor & Francis Group, LLC, 2015. - 209 p.
7. Oleg Skulkin, Learning Android Forensics / Oleg Skulkin, Donnie Tindall, Rohit Tamma. - Birmingham, UK: Packt Publishing Ltd., 2019. - 328 p.
8. Sheran Gunasekera, Android Apps Security: Mitigate Hacking Attacks and Security Breaches / Sheran Gunasekera. - New York, USA: Apress, 2020. - 312 p.
9. Ajin Abraham, Automated Security Analysis of Android and iOS Applications with Mobile Security Framework / Ajin Abraham, Henry Dalziel. - Waltham, USA: Elsevier, Inc., 2016.
10. Miroslav Vitula, Learning zANTI2 for Android Pentesting / Miroslav Vitula. - Birmingham, UK: Packt Publishing Ltd., 2015. - 132 p.
11. Mu Zhang, Android Application Security: A Semantics and Context-Aware Approach / Mu Zhang, Heng Yin. - Princeton, USA: NEC Laboratories America, Inc., 2016.
12. Pragati Ogal Rai, Android Application Security Essentials / Pragati Ogal Rai. - Birmingham, UK: Packt Publishing Ltd., 2013.
13. David Thiel, iOS Application Security / David Thiel. - San Francisco, USA: No Starch Press, Inc., 2016. - 297 p.

14. Jim Doherty, Wireless and Mobile Device Security / Jim Doherty. - Burlington, USA: Jones & Bartlett Learning, 2016. - 377 p.
15. Vulnerability Details: CVE-2017-0781 [Электронный ресурс]. - Электрон. дан. - Режим доступа: https://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2017-0781
16. CVE-2017-13156 Detail [Электронный ресурс]. - Электрон. дан. - Режим доступа: <https://nvd.nist.gov/vuln/detail/CVE-2017-13156>
17. Vulnerability Details: CVE-2015-3864 [Электронный ресурс]. - Электрон. дан. - Режим доступа: <https://www.cvedetails.com/cve/CVE-2015-3864/>
18. Jonathan Levin, Android Internals: Power User's View / Jonathan Levin. - Jonathan Levin, 2015 - 250 p.
19. Jonathan Levin, MacOS and iOS Internals, Volume I: User Mode / Jonathan Levin. - Jonathan Levin, 2017 - 540 p.
20. Jonathan Levin, Android Internals: Developer's View / Jonathan Levin. - Jonathan Levin, 2017 - 400 p.
21. Android permissions: User attention, comprehension, and behavior, Adrienne Porter Felt [и др.], [Электронный ресурс]. - Электрон. дан. - Режим доступа: https://www.researchgate.net/publication/241770660_Android_permissions_User_attention_comprehension_and_behavior
22. Michael Lane, Does the android permission system provide adequate information privacy protection for end-users of mobile apps?, Michael Lane, [Электронный ресурс]. - Электрон. дан. - Режим доступа: https://www.researchgate.net/publication/233960125_Does_the_android_permission_system_provide_adequate_information_privacy_protection_for_end-users_of_mobile_apps
23. Руденков Н.А., Долинер Л.И. Основы сетевых технологий: Учебник для вузов. Екатеринбург: Изд-во Уральского. Федерального ун-та, 2011.

ПРИЛОЖЕНИЕ 1

```

fun getInfo(): String {
    val telephonyManager = Application.context.getSystemService(Context.TELEPHONY_SERVICE) as? TelephonyManager?
    var numbers = ""
    if (ActivityCompat.checkSelfPermission(
        Application.context,
        Manifest.permission.READ_PHONE_STATE
    ) == PackageManager.PERMISSION_GRANTED
    ) {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP_MR1) {
            val subscriptionManager = Application.context.getSystemService(Context.TELEPHONY_SUBSCRIPTION_SERVICE) as? SubscriptionManager?
            subscriptionManager?.activeSubscriptionInfoList?.forEach { it: SubscriptionInfo?
                | numbers += "Number ${it.simSlotIndex}: ${it.number}\n"
            }
        } else {
            numbers = "Number 1: ${telephonyManager?.line1Number}"
        }
    }
}

return "Build info:\n\tBoard: ${Build.BOARD}\n" +
    "\tBootloader: ${Build.BOOTLOADER}\n" +
    "\tBrand: ${Build.BRAND}\n" +
    "\tDevice: ${Build.DEVICE}\n" +
    "\tDisplay: ${Build.DISPLAY}\n" +
    "\tFingerprint: ${Build.FINGERPRINT}\n" +
    "\tHardware: ${Build.HARDWARE}\n" +
    "\tHost: ${Build.HOST}\n" +
    "\tID: ${Build.ID}\n" +
    "\tManufacturer: ${Build.MANUFACTURER}\n" +
    "\tModel: ${Build.MODEL}\n" +
    "\tProduct: ${Build.PRODUCT}\n" +
    "\tTags: ${Build.TAGS}\n" +
    "\tType: ${Build.TYPE}\n" +
    "\tUser: ${Build.USER}\n" +
    "\tCodename: ${Build.VERSION.CODENAME}\n" +
    "\tSDK Int: ${Build.VERSION.SDK_INT}\n" +
    "\tPhone numbers: numbers"
}

```

Рисунок П1.1. Получение информации об устройстве

```

val location = locationManager?.getLastKnownLocation(LocationManager.GPS_PROVIDER)
?: locationManager?.getLastKnownLocation(LocationManager.NETWORK_PROVIDER)
"Longitude: ${location?.longitude} Latitude: ${location?.latitude} Accuracy: ${location?.accuracy} " +
    "Updated: ${SimpleDateFormat( pattern: "YYYY.MM.dd HH:mm").format(location?.time)}"

```

Рисунок П1.2. Получение последнего зафиксированного
местоположения

```

fun getContacts(context: Context): String {
    val builder = StringBuilder()

    val resolver = context.contentResolver
    val cursor = resolver.query(
        ContactsContract.Contacts.CONTENT_URI,
        projection: null,
        selection: null,
        selectionArgs: null,
        sortOrder: null
    ) ?: return "none"

    if (cursor.count > 0) {
        while (cursor.moveToNext()) {
            val id = cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts._ID))
            val name = cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME))
            val phoneNumber = cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts.HAS_PHONE_NUMBER)).toInt()
            if (phoneNumber > 0) {
                val cursorPhone = context.contentResolver.query(
                    ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
                    projection: null,
                    selection: ContactsContract.CommonDataKinds.Phone.CONTACT_ID + "=?",
                    arrayOf(id),
                    sortOrder: null
                )

                if (cursorPhone?.count ?: 0 > 0) {
                    while (cursorPhone?.moveToNext() == true) {
                        val phoneNumberValue = cursorPhone.getString(
                            cursorPhone.getColumnIndex(ContactsContract.CommonDataKinds.Phone.NUMBER)
                        )
                        builder.append("Contact: ")
                            .append(name)
                            .append(", Phone Number: ")
                            .append(phoneNumberValue)
                            .append("\n")
                    }
                }
                cursorPhone?.close()
            }
        }
        cursor.close()

        Log.d( tag: "", msg: "Contacts read successful")
        return builder.toString()
    }
}

```

Рисунок П1.3. Получение списка контактов

```

fun readInbox(context: Context): Array<String> {
    val cursor = context.contentResolver.query(
        Telephony.Sms.Inbox.CONTENT_URI,
        arrayOf(
            Telephony.Sms.Inbox.DATE_SENT,
            Telephony.Sms.Inbox.ADDRESS,
            Telephony.Sms.PERSON,
            Telephony.Sms.Inbox.BODY
        ),
        selection: null,
        selectionArgs: null,
        Telephony.Sms.Inbox.DEFAULT_SORT_ORDER
    )

    if (cursor == null) {
        cursor?.close()
        return emptyArray()
    }

    val result = ArrayList<String>()
    if (cursor.moveToFirst()) {
        do {
            var msgData = ""
            for (idx in 0 until cursor.columnCount) {
                msgData += " " + cursor.getColumnName(idx).toString() + ":" + cursor.getString(idx)
            }
            result.add(msgData)
        } while (cursor.moveToNext())
    }
    cursor.close()

    return result.toArray()
}

```

Рисунок П1.4. Получение принятых сообщений

```

fun readSent(context: Context): Array<String> {
    val cursor = context.contentResolver.query(
        Telephony.Sms.Sent.CONTENT_URI,
        arrayOf(
            Telephony.Sms.Sent.DATE_SENT,
            Telephony.Sms.Sent.ADDRESS,
            Telephony.Sms.PERSON,
            Telephony.Sms.Sent.BODY
        ),
        selection: null,
        selectionArgs: null,
        Telephony.Sms.Sent.DEFAULT_SORT_ORDER
    )

    if (cursor == null) {
        cursor?.close()
        return emptyArray()
    }

    val result = ArrayList<String>()
    if (cursor.moveToFirst()) {
        do {
            var msgData = ""
            for (idx in 0 until cursor.columnCount) {
                msgData += " " + cursor.getColumnName(idx).toString() + ":" + cursor.getString(idx)
            }
            result.add(msgData)
        } while (cursor.moveToNext())
    }
    cursor.close()

    return result.toTypedArray()
}

```

Рисунок П1.5. Получение отправленных сообщений

```

fun readDraft(context: Context): Array<String> {
    val cursor = context.contentResolver.query(
        Telephony.Sms.Draft.CONTENT_URI,
        arrayOf(
            Telephony.Sms.Draft.DATE_SENT,
            Telephony.Sms.Draft.ADDRESS,
            Telephony.Sms.PERSON,
            Telephony.Sms.Draft.BODY
        ),
        selection: null,
        selectionArgs: null,
        Telephony.Sms.Draft.DEFAULT_SORT_ORDER
    )

    if (cursor == null) {
        cursor?.close()
        return emptyArray()
    }

    val result = ArrayList<String>()
    if (cursor.moveToFirst()) {
        do {
            var msgData = ""
            for (idx in 0 until cursor.columnCount) {
                msgData += " " + cursor.getColumnName(idx).toString() + ":" + cursor.getString(idx)
            }
            result.add(msgData)
        } while (cursor.moveToNext())
    }
    cursor.close()

    return result.toArray()
}

```

Рисунок П1.6. Получение сообщений в черновиках

ПРИЛОЖЕНИЕ 2

```

class AppView(context: Context, attributes: AttributeSet?): ConstraintLayout(context, attributes) {

    private val view = LayoutInflater.from(context).inflate(R.layout.app_view, root: this, attachToRoot: false)

    private val tvAppName = view.findViewById<TextView>(R.id.tvAppName)
    private val tvIndex = view.findViewById<TextView>(R.id.tvIndex)

    var model: AppModel =
        AppModel( id: 0, name: "", packageName: "", emptyList(), index1: 0, index2: 0.0f)
    set(value) {
        field = value
        id = value.id
        tvAppName.text = value.name
        tvIndex.text = value.index1.toString()

        when {
            value.index2 > 5.5 -> tvIndex.setTextColor(ContextCompat.getColor(context, R.color.red_secret_data))
            value.index2 > 3 -> tvIndex.setTextColor(ContextCompat.getColor(context, R.color.orange_sensitive_data))
            else -> tvIndex.setTextColor(ContextCompat.getColor(context, R.color.green_not_secret_data))
        }
    }

    constructor(context: Context): this(context, attributes: null)

    init {
        this.addView(view)
    }
}

```

Рисунок П2.1. Код класса AppView

ПРИЛОЖЕНИЕ 3

```
private fun getInstalledPackages(): HashMap<String, String> {
    val packageManager = packageManager

    val intent = Intent(Intent.ACTION_MAIN, uri: null)
    intent.addCategory(Intent.CATEGORY_LAUNCHER)
    intent.flags = (Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_RESET_TASK_IF_NEEDED)
    val resolveInfoList = getPackageManager().queryIntentActivities(intent, flags: 0)
    val map: HashMap<String, String> = HashMap()
    for (resolveInfo in resolveInfoList) {
        val activityInfo = resolveInfo.activityInfo
        val packageName = activityInfo.applicationInfo.packageName
        val info = packageManager.getApplicationInfo(packageName, flags: 0)
        if (info.flags and ApplicationInfo.FLAG_SYSTEM != 0) continue
        val label = packageManager.getApplicationLabel(activityInfo.applicationInfo) as String
        map[packageName] = label
    }

    return map
}
```

Рисунок ПЗ.1. Код сбора списка установленных приложений

```
private fun getPermissionForPackage(packageName: String): List<String> {
    val result = ArrayList<String>()

    try {
        val packageInfo =
            packageManager.getPackageInfo(packageName, PackageManager.GET_PERMISSIONS)
        var counter = 1

        if (packageInfo.requestedPermissions == null) return emptyList()
        for (i in packageInfo.requestedPermissions.indices) {
            if (packageInfo.requestedPermissionsFlags[i] and PackageInfo.REQUESTED_PERMISSION_GRANTED != 0) {
                val permission = packageInfo.requestedPermissions[i]
                result.add(permission)
                counter++
            }
        }
    } catch (e: Exception) {
        e.printStackTrace()
    }

    return result.toList()
}
```

Рисунок ПЗ.2. Код сбора списка разрешений для конкретного приложения

ПРИЛОЖЕНИЕ 4

```

<TextView
    android:id="@+id/tvConstant"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:textSize="14sp"
    android:textColor="@color/black"
    android:text="com.android.voicemail.permission.WRITE_VOICEMAIL"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toLeftOf="@id/tvWeight"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toTopOf="@id/tvDescription"/>

<TextView
    android:id="@+id/tvWeight"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="20sp"
    android:textColor="@color/black"
    android:text="18"
    app:layout_constraintLeft_toRightOf="@id/tvConstant"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>

<TextView
    android:id="@+id/tvDescription"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginHorizontal="10dp"
    android:text="Allows an application to modify and remove existing voicemails in the system. "
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@id/tvWeight"
    app:layout_constraintBottom_toBottomOf="parent"/>

```

Рисунок П4.1. Разметка элемента списка разрешений

```

class PermissionView(context: Context, attributes: AttributeSet?): ConstraintLayout(context, attributes) {

    private val view = LayoutInflater.from(context).inflate(R.layout.permission_view, root: this, attachToRoot: false)

    private val tvConstant = view.findViewById<TextView>(R.id.tvConstant)
    private val tvWeight = view.findViewById<TextView>(R.id.tvWeight)
    private val tvDescription = view.findViewById<TextView>(R.id.tvDescription)

    var model: PermissionModel = PermissionModel( constant: "", weight: 0, description: "")
    set(value) {
        field = value
        tvConstant.text = value.constant
        tvWeight.text = value.weight.toString()
        tvDescription.text = value.description

        when {
            value.weight > 15 -> tvWeight.setTextColor(
                ContextCompat.getColor(context, R.color.red_secret_data))
            value.weight > 10 -> tvWeight.setTextColor(
                ContextCompat.getColor(context, R.color.orange_sensitive_data)
            )
            value.weight > 0 -> tvWeight.setTextColor(
                ContextCompat.getColor(context, R.color.green_not_secret_data)
            )
            else -> tvWeight.setTextColor(ContextCompat.getColor(context, R.color.blue_no_data))
        }
    }

    constructor(context: Context): this(context, attributes: null)

    init { this.addView(view) }
}

```

Рисунок П4.1. Код Kotlin класса элемента списка

УДК 004.056.5

**РАЗРАБОТКА АЛГОРИТМА АУТЕНТИФИКАЦИИ ПОЛЬЗОВАТЕЛЯ С
СОЗДАНИЕМ БЕЗОПАСНОГО КАНАЛА ПЕРЕДАЧИ ДАННЫХ В
КОММУНИКАЦИОННЫХ СИСТЕМАХ НА ОСНОВЕ ПРОТОКОЛА UDP**

© 2020 *М. В. Бабичева, А. А. Поздняков*

Проанализированы распространённые методы аутентификации пользователя и создания безопасного канала передачи данных на прикладном уровне модели TCP/IP. Разработан алгоритм аутентификации с одновременным обменом ключом на основе протокола транспортного уровня UDP.

Ключевые слова: безопасная передача данных, мобильные коммуникационные системы, аутентификация, UDP, HTTPS.

Введение. Говоря о современных веб-технологиях в последние несколько лет, исследователи и эксперты всё чаще обращают взгляд на тему перехода пользователей от использования стационарных компьютеров и ноутбуков к использованию мобильных устройств. Так, за 10 лет, в период с 2009 по 2019 год, количество проданных по всему миру смартфонов увеличилось с 172.38 миллионов единиц в год, до 1524.84 миллионов. А количество пользователей смартфонов к 2020 году составило 3.5 миллиарда человек, или 44.81% от всего населения.

Кроме того, если в 2013 году только 16.2% всего интернет-трафика приходилось на смартфоны, то в 2019 году он составлял уже 53.3%. Важно отметить, что эти данные представляют только трафик через смартфоны, не включая другие мобильные устройства, такие как планшетные компьютеры, например, использование которых также выросло за этот период. Таким образом, большая часть пользователей всемирной сети уже перешла на использование мобильных устройств.

Учитывая эту тенденцию многие компании находят очень выгодным разработку мобильных решений для своего бизнеса. Это касается не только компаний, создающих веб-продукты, такие как интернет-магазины, социальные сети, сервисы развлечений и пр. Свою нишу на мобильном рынке сейчас также активно занимают автомобильные производители, производители бытовой техники, транспортные компании и многие другие. Такие компании, в отличие от производителей веб-продуктов, предпочитают строить коммуникацию своих систем на более простых в реализации протоколах транспортного уровня (TCP, UDP), нежели на протоколах прикладного уровня (HTTP).

Однако, в то время как для HTTP существует расширение, позволяющее создавать безопасные сессии с шифрование данных - HTTPS, для протоколов TCP и UDP нет подобных общепринятых расширений на транспортном уровне. В связи с этим компании вынуждены самостоятельно разрабатывать и реализовывать протоколы аутентификации пользователей, создание сессий и шифрование данных. Это, в свою очередь, раз за разом приводит к появлению небезопасных коммуникационных систем, подверженных атакам и утечкам данных в лучшем случае и к потенциально опасным для жизни и здоровья пользователей выходам оборудования из строя в худшем.

Постановка задачи. Целью данной работы является анализ известных протоколов аутентификации пользователя и создания безопасного канала передачи данных, а также разработка оптимального алгоритма для использования в системе с протоколом передачи UDP. Результаты предполагается использовать при построении коммуникационной системы с архитектурой клиент-сервер.

Схемы аутентификации HTTP. В первую очередь следует выбрать схему аутентификации пользователя. Во время аутентификации пользователя безопасный канал передачи данных ещё может быть не организован. Поэтому будем считать, что все данные передаются в незашифрованном виде. Наиболее распространёнными схемами аутентификации в HTTP являются:

1) **Basic authentication.** Для аутентификации клиент отправляет серверу строку вида «логин:пароль», закодированную с помощью Base64 (рис. 1). Очевидным недостатком такого метода является то, что при перехвате пакета аутентификации, злоумышленник сможет легко получить логин и пароль пользователя. В таком случае злоумышленник сможет не только получить доступ к данному серверу, но и потенциально к другим сервисам, которыми пользуется данный пользователь, так как зачастую пользователи предпочитают использовать одинаковые или похожие логины и пароли для различных сервисов. Таким образом, чтобы использовать basic authentication необходимо сначала установить безопасный канал передачи, гарантирующий защиту от перехвата.

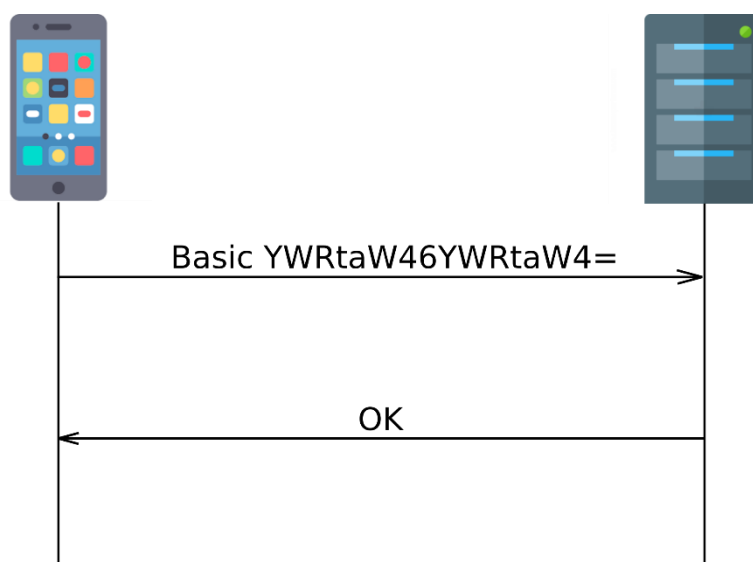


Рисунок 1. Basic authentication

2) **Bearer authentication.** В данной схеме, клиент не передаёт свои логин и пароль для аутентификации. Вместо этого он отправляет серверу заранее выданный ему токен аутентификации. Эта схема также уязвима для перехвата. Получив токен, злоумышленник сможет аутентифицироваться на сервере под видом клиента. Однако такая схема не ставит под удар другие сервисы, так как токен аутентификации уникален для данного сервера и не будет принят другими сервисами.

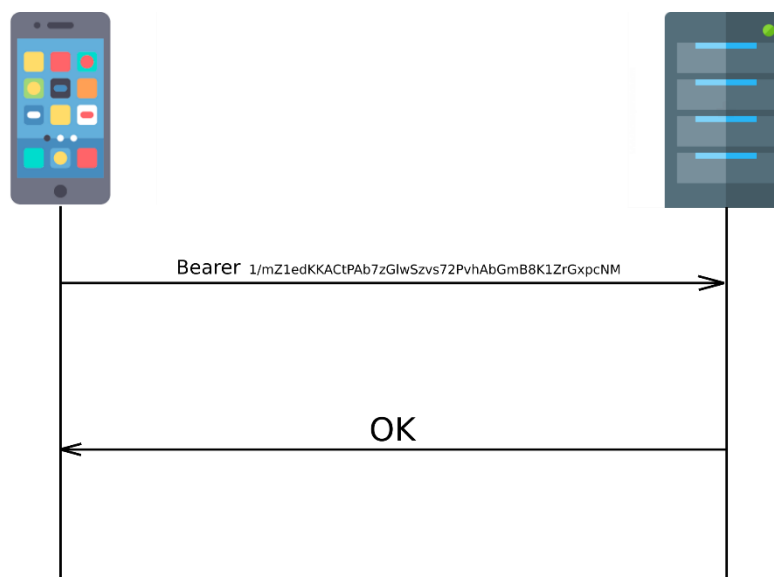


Рисунок 2. Bearer authentication

3) **Digest authentication.** В схеме digest authentication клиент отправляет серверу запрос на аутентификацию. Сервер в ответ присылает случайное число (nonce). Далее клиент вычисляет MD5 хеш от конкатенации логина, пароля и nonce и отправляет его на сервер. Сервер вычисляет такой же хеш и сравнивает его с полученным от клиента, если данные совпадают, то пользователь проходит аутентификацию. Графически схема представлена на рисунке 3. Несмотря на то, что данная схема требует больше шагов, она же является и самой надёжной из рассмотренных. Digest authentication гарантирует, что даже при перехвате данных злоумышленник не имеет возможности обратить хеш-функцию и получить необходимые для аутентификации данные.

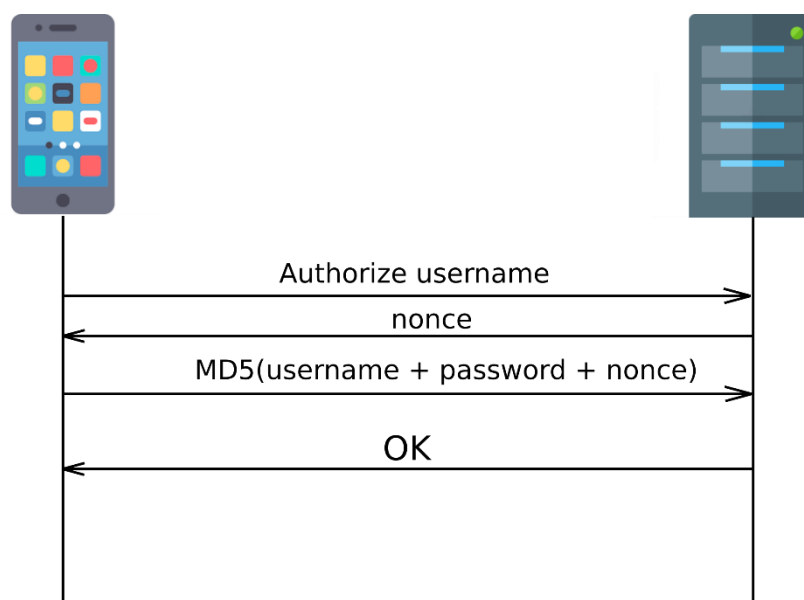


Рисунок 3. Digest authentication

Создание безопасного канала передачи данных в HTTPS. Как уже было сказано, протокол HTTP имеет расширение HTTPS, позволяющее создавать безопасный канал передачи между клиентом и сервером. В HTTPS клиент и сервер в начале сеанса связи договариваются о ключе шифрования при помощи асимметричного алгоритма (RSA, Diffie-Hellman), а затем шифруют данные с помощью симметричного алгоритма (AES, DES, RC4,

RC2, 3-DES) используя этот ключ. Наиболее существенной и наиболее уязвимой частью данного протокола является обмен ключом. Рассмотрим оба алгоритма.

1) **RSA**. Алгоритм RSA был разработан в конце 1970-х тремя учёными: Ривестом, Шамиром и Адлеманом, по первым буквам фамилий которых и назвали алгоритм. Обмен ключом по этому алгоритму осуществляется следующим образом. Алиса отправляет Бобу свой открытый ключ RSA (e, n) . Далее Боб создаёт ключ шифрования для симметричного алгоритма, шифрует его открытым ключом Алисы и возвращает Алисе зашифрованный ключ. Алиса, используя свой приватный ключ (d, n) , расшифровывает сообщение и получает ключ шифрования. Таким образом и у Алисы, и у Боба есть одинаковый ключ, которым они теперь могут шифровать данные и передавать по незащищенному каналу. Графически алгоритм представлен на рисунке 4.

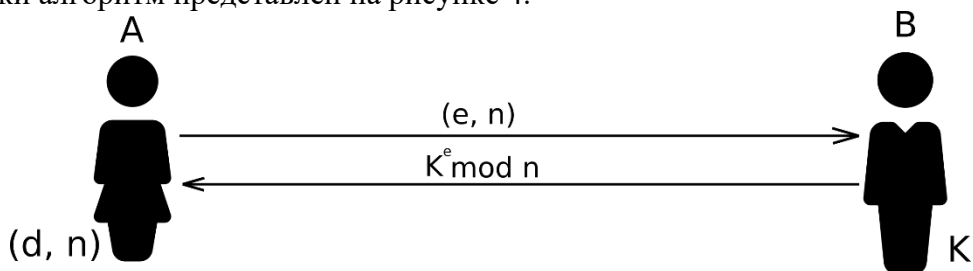


Рисунок 4. Обмен ключом по RSA

2) **Diffie-Hellman**. Алгоритм опубликован Диффи и Хеллманом в 1976, на год раньше RSA. По этому алгоритму Алиса и Боб заранее договариваются о публичном ключе (g, n) . Когда им нужно договориться о ключе, Алиса и Боб выбирают секретные числа $(a$ и $b)$. После этого они обмениваются значениями $A = g^a \bmod n$ и $B = g^b \bmod n$. Далее Алиса вычисляет $K = B^a \bmod n$, а Боб $K = A^b \bmod n$. Таким образом теперь они оба имеют ключ для дальнейшего шифрования данных (рис. 5).

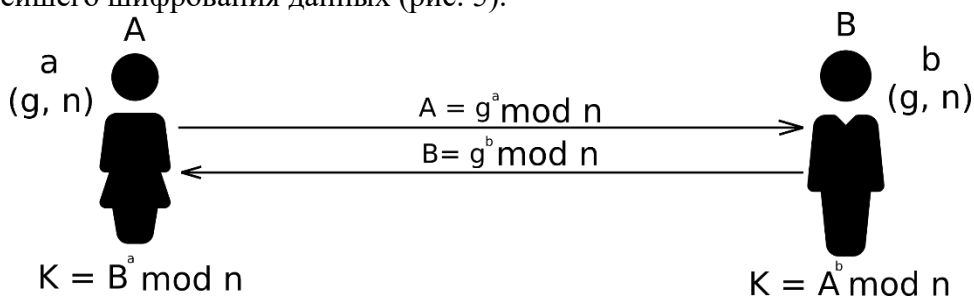


Рисунок 5. Обмен ключом по Diffie-Hellman

Создание собственного алгоритма. Вначале разработки, была предложена схема с базовой аутентификацией и обменом ключом по алгоритму RSA. Рассмотрим эту схему.

В первую очередь клиент инициирует процесс аутентификации, отправляя серверу логин пользователя. Сервер присылает клиенту публичный ключ для шифрования пароля. Клиент шифрует пароль пользователя и отправляет шифротекст серверу. Сервер расшифровывает сообщение и проверяет пароль пользователя. Если пароли совпадают, то пароль используется как ключ шифрования для алгоритма AES в дальнейшей передаче данных. Графически алгоритм представлен на рисунке 6.

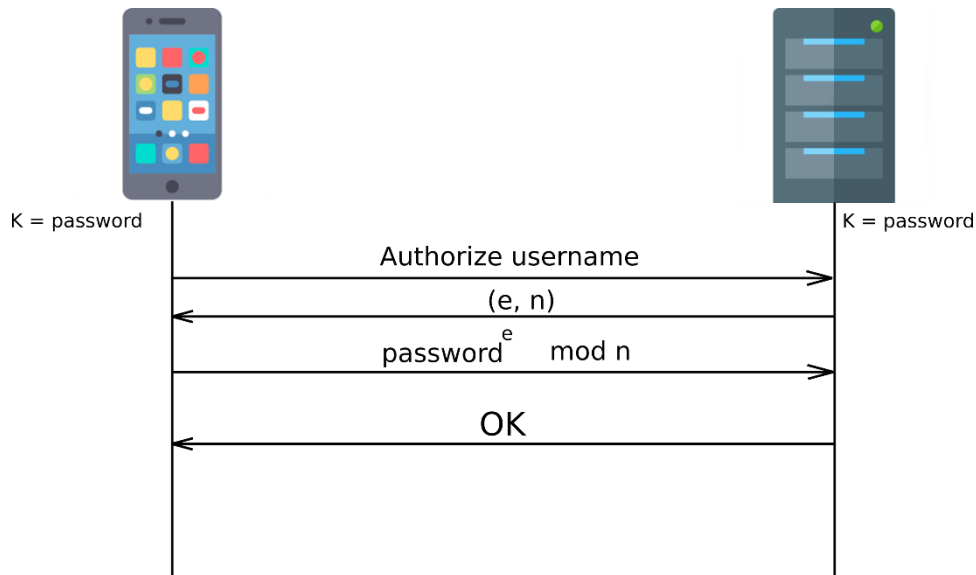


Рисунок 6. Аутентификация RSA

В данной схеме перехвативший трафик злоумышленник не сможет получить пароль пользователя, так как данные не передаются в открытом виде. Однако при дальнейшем анализе системы выяснилось, что злоумышленник потенциально может не только перехватывать трафик, но и подменять его. В таком случае предложенная схема является уязвимой к атаке Man-In-The-Middle (MITM).

Для разрешения этой проблемы был разработан модифицированный алгоритм, основанный на digest authentication (рис. 7). Первый шаг алгоритма не изменился - клиент отправляет серверу логин, под которым он хочет аутентифицироваться. Сервер отвечает клиенту случайным числом. Далее клиент вычисляет SHA256 от конкатенации пароля пользователя и этого случайного числа и отправляет хеш-сумму серверу. Сервер проводит те же вычисления и сравнивает результат с ответом клиента. Если они совпадают, то аутентификация считается успешной и пароль пользователя можно использовать как ключ шифрования для дальнейшего обмена данными.

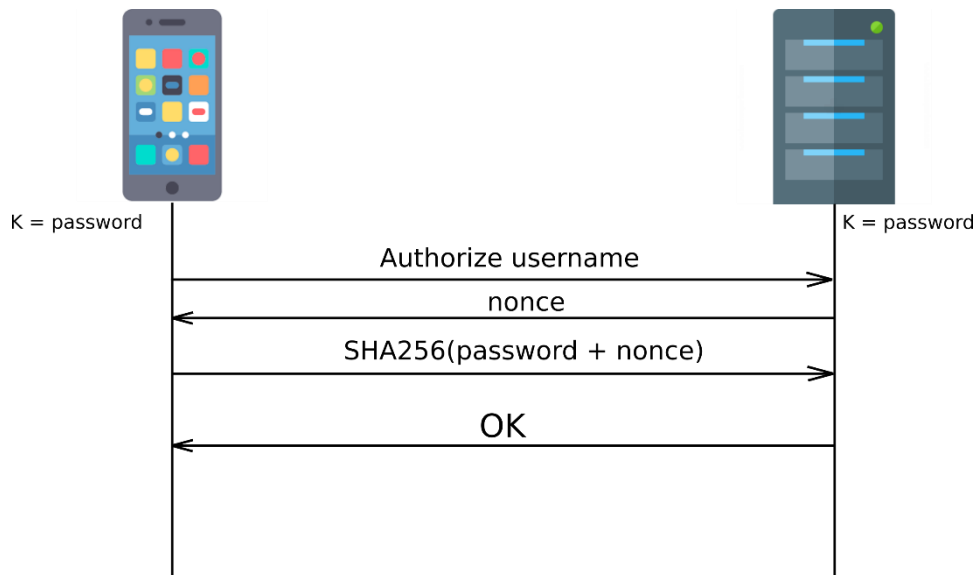


Рисунок 7. SHA256 аутентификация

Выводы. Проведенный анализ алгоритмов аутентификации пользователя и создания безопасного канала передачи данных, позволил разработать алгоритм аутентификации с одновременным обменом ключом в клиент-серверной

коммуникационной системе, основанной на протоколе UDP. Алгоритм был улучшен для устранения уязвимости MITM, что также значительно ускорило скорость работы системы, за счёт устранения сложных вычислений в рамках алгоритмов RSA или Диффи-Хеллмана.

СПИСОК ЛИТЕРАТУРЫ

1. Diffie W. New Directions In Cryptography / W. Diffie, M. E. Hellman // IEEE Transactions on Information Theory. - 1976. - № 6. - P. 644 – 654.
2. Gardner M. A New Kind of Cipher that Would Take Millions of Years to Break / M. Gardner, M. Gardner // Scientific American. - NYC.- 1977. - Vol. 237. - P. 120 – 124.
3. Rescola E. SSL and TLS: Designing and Building Secure Systems / E. Rescola // Addison-Wesley Professional. - 2000. - 386 p.
4. Hypertext Transfer Protocol – HTTP/1.1 / The Internet Society. - Текст : электронный. - URL: <https://tools.ietf.org/html/rfc2616> (дата обращения: 11.10.2020).

DESIGNING AN ALGORITHM OF AUTHENTICATION WITH SECURE TRANSMISSION CHANNEL CREATION WITHIN COMMUNICATION SYSTEMS BASED ON UDP PROTOCOL

M. V. Babicheva, A. A. Pozdniakov

Analysed most common methods of user authentication and creation secure data transmission channel on the application level of TCP/IP network model. Designed authentication algorithm with secure data transmission channel creation based on UDP protocol.

Keywords: secure data exchange, mobile communication systems, authentication, UDP, HTTPS.

Бабичева Маргарита Вадимовна

ст. преподаватель кафедры радиофизики и
инфокоммуникационных технологий ГОУ
ВПО “Донецкий национальный
университет”, ДНР, Донецк.
E-mail: m.v.babicheva60@gmail.com

Babicheva Margarita Vadimovna

Senior Lecturer at Department of Radiophysics
and
Infocommunication Technologies
of Donetsk
National University,
DPR, Donetsk.

Поздняков Александр Андреевич

студент кафедры радиофизики и
инфокоммуникационных технологий ГОУ
ВПО “Донецкий национальный
университет”, ДНР, Донецк.
E-mail: mail.0awawa0@gmail.com

Pozdniakov Alexander Andreyevich

Student at Department of Radiophysics and
Infocommunication Technologies of Donetsk
National University, DPR, Donetsk.