

МИНИСТЕРСТВО ВНУТРЕННИХ ДЕЛ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ КАЗЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

МОСКОВСКИЙ УНИВЕРСИТЕТ МВД РОССИИ ИМЕНИ В.Я.КИКОТЯ

Кафедра специальных информационных технологий

учебно-научного комплекса информационных технологий.

КУРСОВАЯ РАБОТА

по учебной дисциплине

Компьютерная разведка

**ИССЛЕДОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**С ЦЕЛЬЮ ОБНАРУЖЕНИЯ**

**НЕДОКУМЕНТИРОВАННЫХ ВОЗМОЖНОСТЕЙ**

|  |  |
| --- | --- |
| Курсовую работу выполнил: | Курсант 971 взвода  3 курса ФПСОИБ  Покинен Андрей Эдуардович |
| Научный руководитель курсовой работы: | Начальник кафедры специальных информационных технологий учебно-научного комплекса информационных технологий, кандидат технических наук, подполковник полиции  Поликарпов Евгений Сергеевич |
| Работа защищена на оценку | Научный руководитель курсовой работы |

Москва 2020

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc43047189)

[ГЛАВА I. ПОНЯТИЕ И КЛАССИФИКАЦИЯ ПО 5](#_Toc43047190)

[§ 1.1. Понятие ПО 5](#_Toc43047191)

[§ 1.2. Классификация ПО 5](#_Toc43047192)

[§ 1.3. Общая структура прикладного ПО 6](#_Toc43047193)

[§ 1.4. Недокументированные возможности и документация ПО 8](#_Toc43047194)

[§ 1.5. Подходы к изучению ПО 9](#_Toc43047195)

[§ 1.6. Декомпиляция и дизассемблирование. 12](#_Toc43047196)

[ГЛАВА II. ПРАКТИЧЕСКОЕ ИССЛЕДОВАНИЕ 15](#_Toc43047197)

[ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 15](#_Toc43047198)

[§ 2.1. Исследование мобильного приложения 15](#_Toc43047199)

[ЗАКЛЮЧЕНИЕ 20](#_Toc43047200)

[СПИСОК БИБЛИОГРАФИЧЕСКИХ ИСТОЧНИКОВ 21](#_Toc43047201)

# ВВЕДЕНИЕ

На сегодняшний день использование программного обеспечения во всех сферах человеческой деятельности для её упрощения, ускорения и автоматизации уже давно стало нормой. Но чем больше становится программ, тем больше уязвимостей вероятно оно может содержать, учитывая низкий порог входа в индустрию разработки и доступность обучающих материалов. Что бы быть уверенным в безопасности программного обеспечения, в его отказоустойчивости, а также в отсутствии угрозы для конечного пользователя необходимо документировать программные продукты и проводить их всесторонне исследование и тестирование перед их использованием.

В данной курсовой работе рассмотрено программное обеспечение во всем его многообразии и некоторые способы его анализа на предмет наличия недокументированного функционала.

Целью данной работы является изучение процесса обнаружения недокументированных возможностей программного обеспечения путем его комплексного анализа.

В ходе достижения поставленной цели были поставлены следующие задачи:

1. Изучить виды программного обеспечения и их особенности.
2. Познакомиться с общей архитектурой программного обеспечения.
3. Исследовать типы документации к ПО.
4. Рассмотреть способы анализа ПО и его функционала.
5. Провести анализ одного из наиболее эффективных решений на предмет скрытых возможностей.

Объектом представленной работы является программное обеспечение.

Предмет исследования составляет изучения методов и средств реверс инжиниринга, направленных на обнаружение недокументированных возможностей программных продуктов.

Практическая значимость данной работы определяется тем, что материалы исследования могут найти применение при изучении таких дисциплин как «Основы информационной безопасности», «Информационно-техническая защита информации» и послужить пособием по обратному проектированию отдельных видов приложений и их комплексному анализу.

Теоретическое значение исследования состоит в расширение представлений видах и архитектурах ПО, возможностях его обратного проектирования и способах обнаружение угроз безопасности, создаваемых некачественным программным обеспечением.

Курсовая работа состоит из введения, двух глав, образованных 7 параграфов, заключения, библиографического списка. Работа изложена на 21 листах, машинописного текста (основной текст занимает 20 страниц и содержит 11 рисунков).

# ГЛАВА I. ПОНЯТИЕ И КЛАССИФИКАЦИЯ ПО

## § 1.1. Понятие ПО

Программное обеспечение - это набор инструкций, данных или программ, используемых для управления компьютерами и выполнения определенных задач. В отличие от аппаратного обеспечения, которое описывает физические аспекты компьютера, программное обеспечение - это общий термин, используемый для обозначения приложений, сценариев и программ, работающих на устройстве. Программное обеспечение можно рассматривать как переменную часть компьютера, а аппаратную часть - как неизменную часть.

## § 1.2. Классификация ПО

Для успешного анализа программного обеспечения необходимо знать о том, каких видов оно бывает, для чего предназначено и как устроено изнутри.

Все программное обеспечение можно разделить на следующие категории:

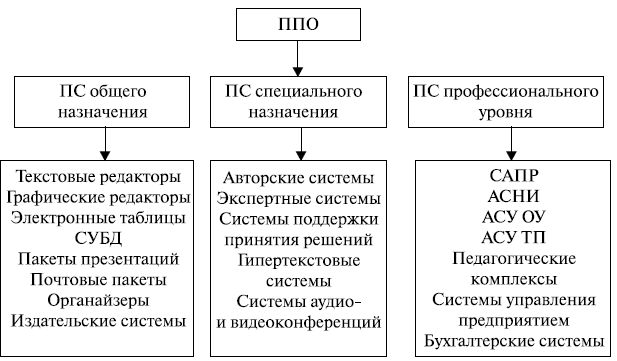
1. Системное ПО
2. Утилиты
3. Прикладное ПО

Системное программное обеспечение включает в себя операционные системы и любую программу, которая поддерживает прикладное программное обеспечение, а также представляет собой среду для его работы и взаимодействия с другими приложениями и аппаратной частью.

Утилиты представляют собой небольшие полезные программы с ограниченными возможностями. Некоторые утилиты поставляются с операционными системами. Как и приложения, утилиты, как правило, устанавливаются отдельно и могут использоваться независимо от остальной части операционной системы.

Под прикладным программным обеспечением понимаются загруженные пользователем программы, которые удовлетворяют те или иные потребности. Примеры приложений включают офисные пакеты, программы для работы с базами данных, веб-браузеры, текстовые редакторы, инструменты разработки программного обеспечения, редакторы изображений и коммуникационные платформы.

В контексте данной работы будет произведен акцент на исследование именно прикладного ПО, так как оно наиболее подвержено анализу и более вероятно может содержать недокументированный функционал, созданный с целью получения данных о пользователях. Рассмотрим классификацию прикладного ПО (рис.1).



Рисунок– 1. Классификация прикладного ПО

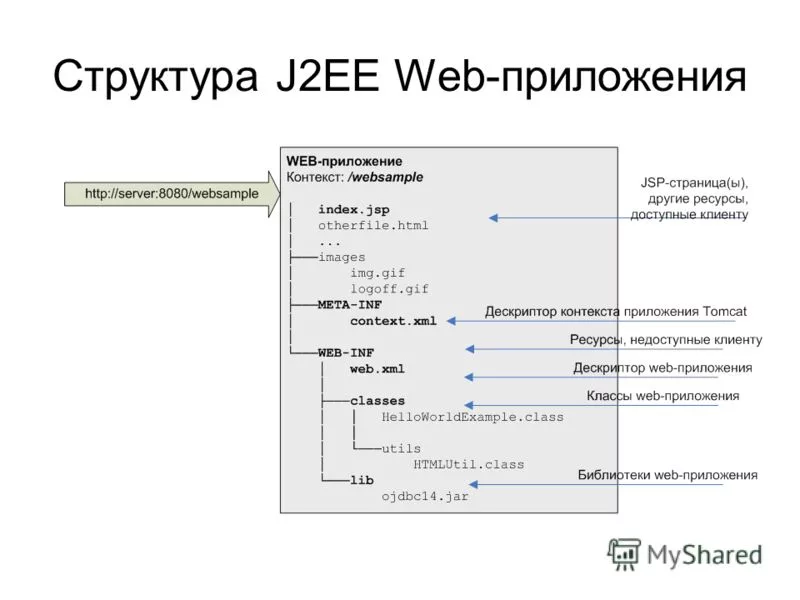
## § 1.3. Общая структура прикладного ПО

Любое прикладное программное обеспечение предназначено для определенной платформы или операционной системы. Под каждую из таких платформ существуют свои технологии и средства разработки ПО (SDK), к ним привязаны языки программирования, фреймворки, а также присущи определенные шаблоны проектирования.

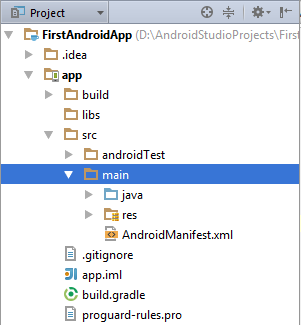
Для того чтобы разобраться во всем этом многообразии необходимо как минимум быть специалистом хотя бы в одном из направлений разработки. Со стороны разработчика любое программное обеспечение выглядит как огромная папка с множеством тесно взаимосвязанных между собой файлов внутри, которые после определенной процедуры начинают работать как единое целое и выполнять ряд предписанных функций.

Недокументированный функционал может быть расположен в любом файле приложения, и чтобы его обнаружить необходим анализ кода, о котором речь пойдет в следующей главе.

Приведем примеры структур проектов приложений:



Рисунок– 2. Структура J2EE Web-приложения



Рисунок– 3. Структура Android приложения

## § 1.4. Недокументированные возможности и документация ПО

Недокументированные возможности **-** возможности [программного обеспечения](https://ru.wikipedia.org/wiki/Программное_обеспечение), не отраженные в документации. Такой функционал может быть добавлен разработчиками в целях тестирования, дальнейшего расширения функциональности, или же в целях скрытого контроля за пользователем. Также недокументированные возможности могут стать следствием поспешных обновлений ПО, не учтённых разработчиками.

Следует отличать скрытые от пользователя возможности приложения, приведенные в официальной сервисной документации и лицензионном соглашении от недокументированных возможностей. Недокументированные возможности обнаруживаются, обычно, в процессе обратного проектирования, но могут быть обнаружены и случайно или с помощью автоматического исследования.

В случае программного обеспечения, отдельный интерес, представляют недокументированные возможности, которые могут поставить [безопасность](https://ru.wikipedia.org/wiki/Информационная_безопасность) программного обеспечения. В этом контексте обычно используются термин [уязвимость](https://ru.wikipedia.org/wiki/Уязвимость_(компьютерная_безопасность)). Существует четыре основных типа документации на ПО:

- проектная — обзор программного обеспечения, включающий описание рабочей среды и принципов, которые должны быть использованы при создании ПО;

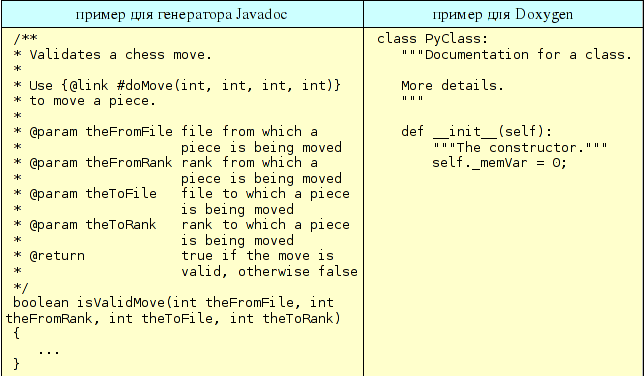
- техническая — документация на код, алгоритмы, интерфейсы, API;

- пользовательская —руководства для конечных пользователей, администраторов системы и другого персонала;

- маркетинговая —совокупность рекламной информации о ПО.

В контексте данной работы наибольший интерес представляет техническая документация, из которой возможно выделить структуры программы и ее функционал для дальнейшего исследования. Такая документация описывает что делает каждый структурный элемент в коде и имеет явно выраженный технический характер.

Зачастую для ее составления используют генераторы документации такие как javadoc, Ndoc, Doxygen и другие (Рисунок–4).



Рисунок– 4. Примеры генерации документации

Стандарты разработки документации. В настоящее время действуют следующие стандарты документирования:

1. ГОСТ 19.201 (Единая система программной документации (ЕСПД);
2. ГОСТ 2.015-2013 (Единая система конструкторской документации (ЕСКД);
3. ГОСТ 34.602 (Комплекс стандартов на автоматизированные системы (КСАС).

# Применяются приведенные выше стандарты в основном в разработке ПО по государственному заказу. Коммерческие компании зачастую следуют западным стандартам или же разрабатывают собственные.§ 1.5. Подходы к изучению ПО

В настоящее время выделяют два основных подхода к изучению программного обеспечения и его тестированию:

1. метод черного ящика;
2. метод белого ящика.

Рассмотрим каждый метод подробнее.

Метод чёрного ящика – метод, при котором исходный код не доступен, и необходимо изучить весь функционал, не прибегая к его анализу. Зачастую производится автоматическое тестирование ПО, готовыми утилитами. Существует много типов тестирования черного ящика, но ниже перечислены наиболее известные из них.

Функциональное тестирование - этот тип тестирования черного ящика связан с функциональными требованиями системы.

Нефункциональное тестирование-этот тип тестирования черного ящика не связан с тестированием конкретной функциональности, а связан с нефункциональными требованиями, такими как производительность, масштабируемость, удобство использования.

Регрессионное тестирование- тестирование, которое проводится после исправления кода, обновления или любого другого технического обслуживания системы, чтобы проверить, что новый код не повлиял на существующий.

Преимущества метода чёрного ящика:

1. Позволяет быстро находить баги в разработанной функциональности ПО;
2. Тестировщик не обязательно должен обладать узкопрофильной специальностью;
3. Проверка проходит с позиции конечного потребителя;
4. Разрабатывать тест-кейсы можно сразу же после завершения работы со спецификацией.

Существуют общие действия, выполняемые для производства любого типа анализа черного ящика:

- первоначально рассматриваются требования и технические характеристики системы;

- тестер выбирает допустимые входные данные;

- тестер определяет ожидаемые выходы для всех этих входов;

- тестер программного обеспечения строит тестовые случаи с выбранными входными сигналами;

- тестовые случаи выполняются;

- тестер программного обеспечения сравнивает фактические выходы с ожидаемыми выходами;

- дефекты, если таковые имеются, исправляются и повторно проверяются.

Метод белого ящика **–** метод, при котором исходный код доступен для анализа полностью. В данном случае используются специальные программы для статического и динамического анализа всего кода или его исполняемых частей.

Выделяют следующие типы такого тестирования:

- модульное тестирование. Разработчик выполняет модульное тестирование, чтобы проверить, работает ли определенный модуль или единица кода. Модульное тестирование проходит на самом базовом уровне, поскольку оно выполняется, когда разрабатывается блок кода, или задается определенная функциональность;

- статический и динамический анализ. Статический анализ включает в себя покрытие всего кода, чтобы узнать о любом возможном дефекте в коде. Динамический анализ включает в себя выполнение кода и анализ выходных данных;

- покрытие операторов. В этом виде тестирования код пишется таким образом, что каждый оператор приложения выполняется хотя бы один раз. Это помогает гарантировать, что все утверждения выполняются без какого-либо побочного эффекта;

- покрытие ветвей. Никакое программное приложение не может быть записано в непрерывном режиме кодирования, в какой-то момент нам нужно разветвить код, чтобы выполнить задать функциональность. Покрытие ветвей помогает в проверке всех ветвей в коде и в обеспечении того, чтобы ветвление не привет к ненормальному поведению приложения;

- тестирование защищенности. Тестирование защищенности проводится для того, чтобы выяснить, насколько хорошо система может защитить себя от несанкционированного доступа, взлома, любого повреждения кода и т. д., – всего, касающегося кода приложения. Этот вид требует сложных методов тестирования.

Преимущества тестирования методом белого ящика:

- поскольку знание внутренней структуры кодирования является предпосылкой, становится очень легко выяснить, какой тип ввода данных может помочь в эффективном тестировании приложения;

- другим преимуществом тестирования белого ящика является то, что он помогает оптимизировать код;

- это помогает в устранении дополнительных строк кода, которые могут привести к скрытым дефектам.

Все вышеприведенные подходы активно используются при анализе ПО, компаниями различного уровня, а также часто применяются их комбинации при разработке, тестировании и инспектировании ПО. В данной работе остановим наше внимание на методе белого ящика, исходные коды при этом необходимо получить путём декомпиляции.

## § 1.6. Декомпиляция и дизассемблирование.

Декомпиляция - это преобразование исполняемого программного кода в некоторую форму языка программирования более высокого уровня, чтобы он мог быть прочитан человеком. Декомпиляция - один из способов обратного проектирования, который делает противоположное тому, что делает компилятор. Инструмент, который выполняет это, называется декомпилятором. Аналогичный инструмент, называемый дизассемблером, преобразует объектный код в язык ассемблера.

Декомпиляция не всегда успешна по ряду причин. Невозможно декомпилировать все программы, и данные и код трудно разделить, потому что оба они представлены аналогично в большинстве современных компьютерных систем. Имена функций и переменных обычно не хранятся в исполняемом файле, поэтому они обычно не восстанавливаются при декомпиляции, а ведь зачастую именно благодаря им намного проще понять ход мысли разработчика. Некоторые программы могут быть разработаны, чтобы быть устойчивыми к декомпиляции с помощью защитных средств, таких как обфускация, что делает этот процесс еще более сложным.

Зачастую бывает так что если программа написана на достаточно высокоуровневом языке или основана на популярном фреймворке, то декомпилятор достаточно успешно справляется со свой задачей, так как такое технологии оставляют заметные следы в исполняемых файлах. К таким случаям относятся программы написанные на .NET фреймворке, android приложения на Java и Kotlin и другие.

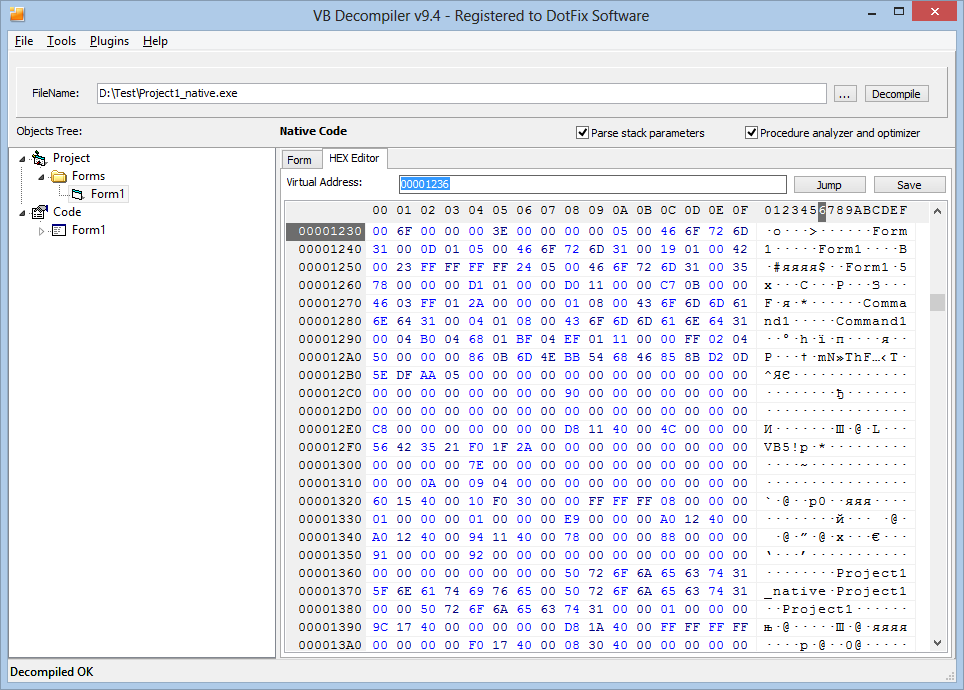
Примеры утилит для анализа ПО:

1) .NET Reflector;

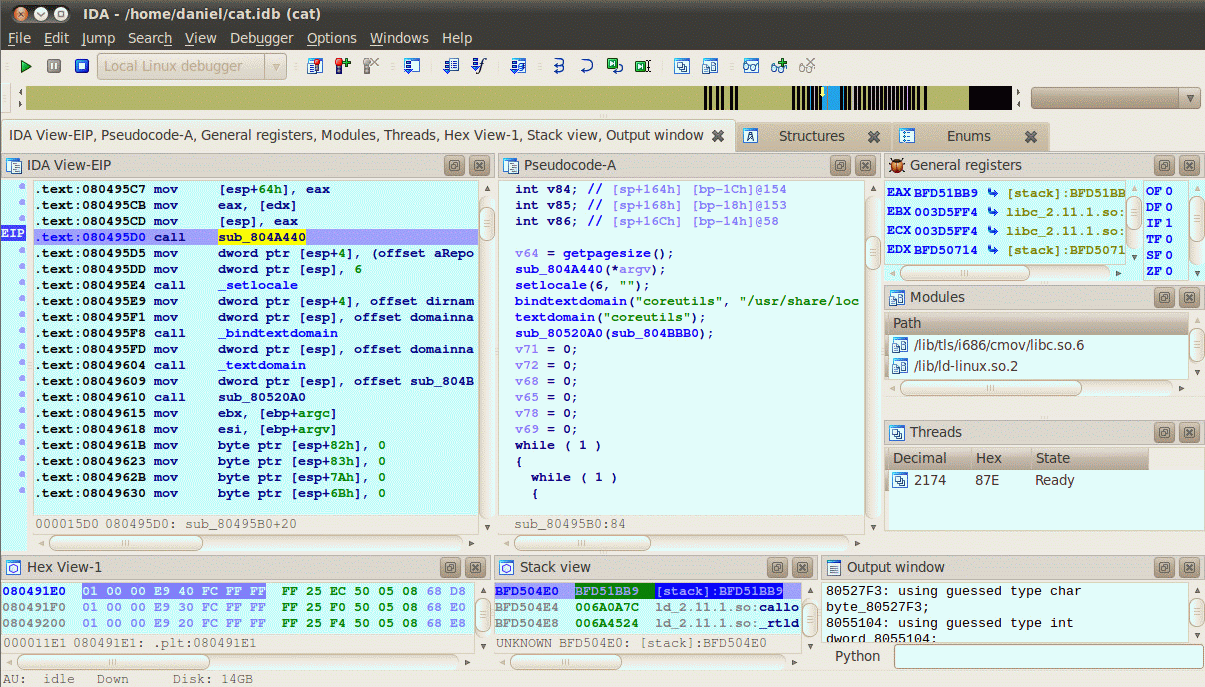
2) IDA pro;

3) Fernflower;

4) ApkTool.



Рисунок– 5. Декомпилятор VB



Рисунок– 6. IDA PRO

# ГЛАВА II. ПРАКТИЧЕСКОЕ ИССЛЕДОВАНИЕ

# ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## § 2.1. Исследование мобильного приложения

с целью обнаружения недокументированных возможностей

Практическая реализация исследования программного обеспечения в данной работы будет состоять в анализе функционала мобильного приложения под операционною систему Android путем статического анализа исходного кода приложения, полученного в результате декомпиляции.

Для этого необходимо произвести следующие действия:

1. Выбрать приложение для анализа.
2. Подобрать утилиты, необходимые для восстановления исходного кода приложения.
3. Провести статический анализ кода приложения, используя один из редакторов кода (например, Android studio).
4. Выявить функционал, не описанный в документации

У каждого Android приложения есть структура. Ниже приведен список обязательных файлов, которые автоматически создаются и обязательно используются при разработке:

AndroidManifest.xmlважный конфигурационный XML-файл, который определяет такие свойства как: имя пакета; минимально необходимую версию Android OS, на которой будет запущено приложение; иконку и т.д. Одним из наиболее известных свойств Android приложений определенных в этом конфигурационном файле является определение «прав доступа приложения».

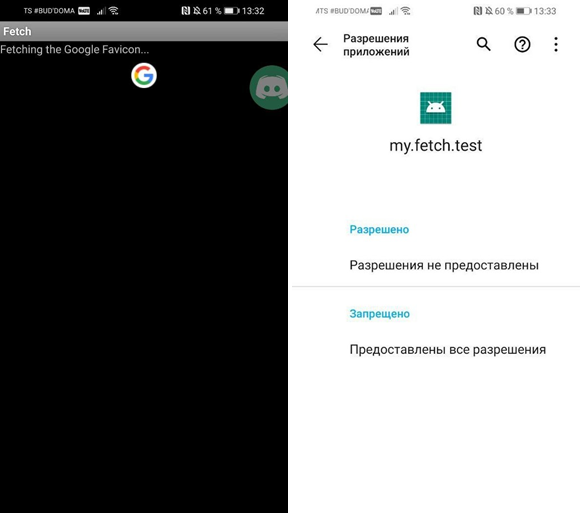
/src место расположения .java файлов;

/res папка ресурсов, которая содержит файлы, зависящие от пользовательского интерфейса. Графика и XML-документы обычно расположены здесь;

/res/layout/main.xml конфигурационный файл содержит определения, которые управляют дизайном основного приложения.

/res/values/strings.xml – Как можно догадаться, файл содержит строки. Это статические определения строк, на которые ссылается ваше приложение.

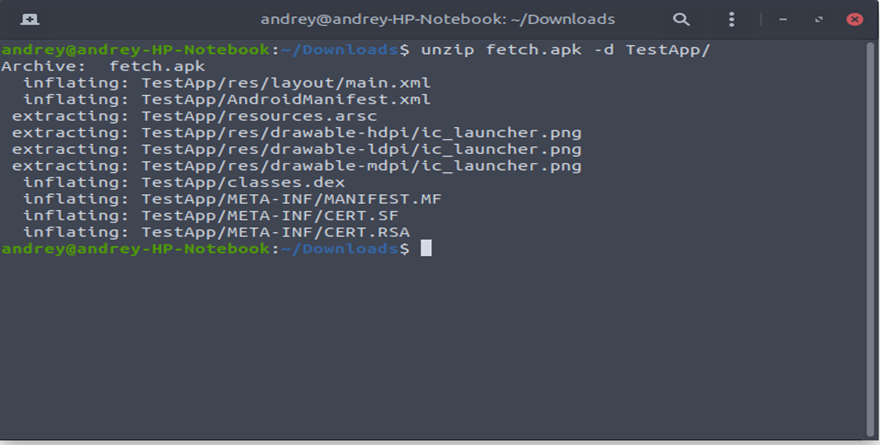
Для практического примера возьмем приложение, которое разработано таким образом, что возможна загрузка данных из интернета без получения соответствующего разрешения от пользователя (Рисунок– 7).



Рисунок– 7. Приложение для анализа

Для начала работы необходим apk файл исследуемого приложения. Файлы пакета приложений Android (APK) - это файлы, которые используются операционной системой Android для распространения и установки мобильных приложений. Обычно файл APK представляет собой просто zip-файл, который был переименован в APK, чтобы операционная система Android распознала его как исполняемый файл. Утилита unzip может быть использована для извлечения файлов, которые хранятся внутри.

Извлечь apk в память телефона можно с помощью специальной утилиты, например, Skit. После чего достаточно по USB подключить телефон к компьютеру и скопировать его. Для начала разархивируем apk (Рисунок– 8).



Рисунок– 8. Распаковка apk файла

Извлечены следующие основные файлы:

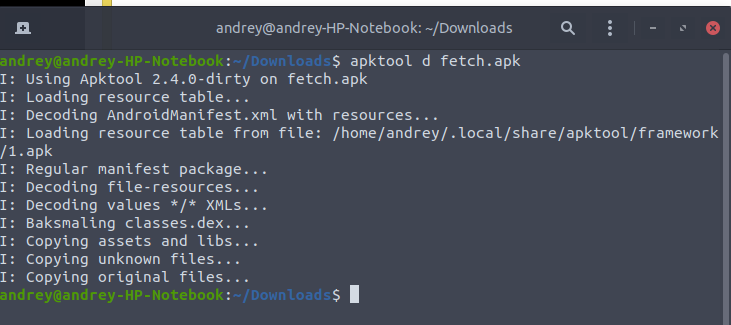
1. AndroidManifest.xml - конфигурационный файл.
2. classes.dex - все файлы классов Java.
3. resources.arsc - вся мета информацию о ресурсах.

Далее необходимо придать приложению состояние, близкое к его первоначальной форме. Для этой цели существует ряд открытых инструментов:

1. apktool– инструмент, используемый для работы с .apk файлами.
2. jad– декомпилятор Java (только для Windows).
3. JD-Core + JD-GUI– еще один декомпилятор, поддерживающий более новые версии и функции Java.
4. dex2jar– инструмент для преобразования. dex файлов в файлы .class.

Для начала воспользуемся apktool. Это позволит распаковать. apk файл, воссоздав структуру каталогов проекта. Также это преобразует двоичный XML файл в текстовый XML. Это быстрый и простой способ исследовать AndroidManifest.xml для того, чтобы увидеть какие именно права доступа требуются приложению. Также. dex код будет преобразован в формат .smali. - дизассемблированную версию байт-кода .dex.

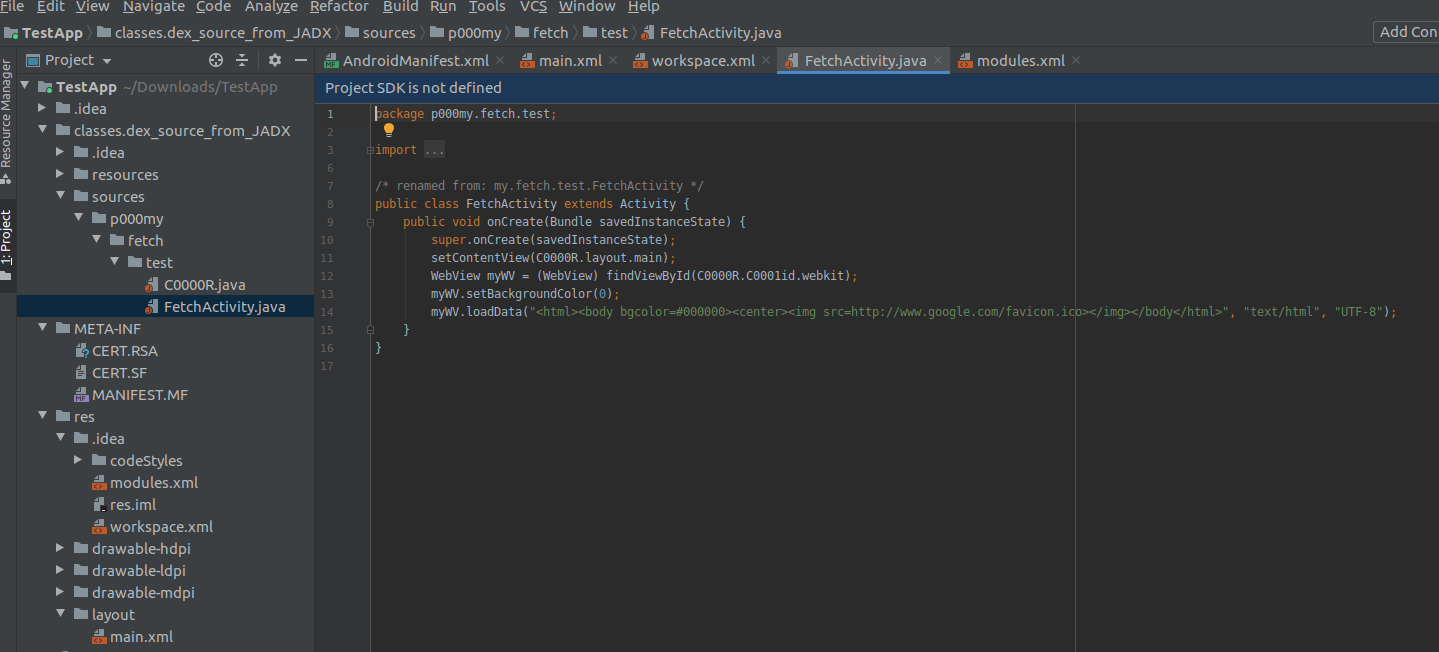
Чтобы получить исходный код приложения необходимопреобразовать clasees.dex в .jar файл с помощью утилиты *dex2jar.* Полученные файлы откроем в IDE Android studio (Рисунок– 9).



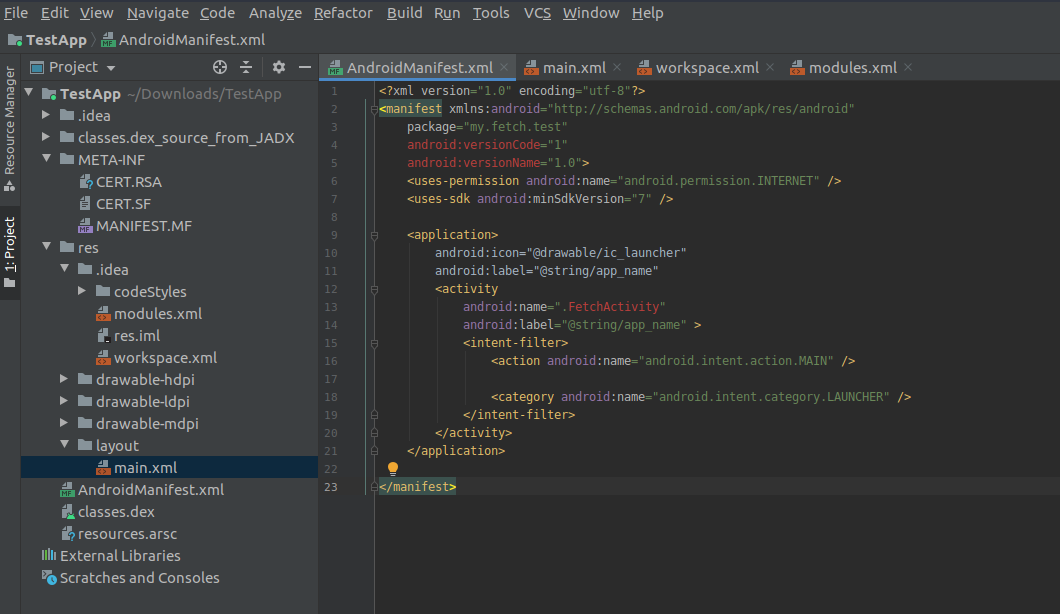
Рисунок– 9. Использование apktool

Получены полностью декомпилировнные исходники приложения (Рисунок– 10), весь java код состоит из одного файла activity, где по указанной ссылке в элементе webview подгружается иконка. Также доступны файлы ресурсов и манифеста. В манифесте можем увидеть получение разрешения на доступ к интернету (Рисунок– 11), но нигде в коде приложения обработки данного разрешения нет.

Это лишь один из примеров того, что можно обнаружить, получив исходные коды приложения. Данное приложение было написано под операционную систему android уже устаревшей версии, и в данный момент даже среда разработки не позволит допустить такую ошибку. Однако этот пример хорошо иллюстрирует то, как важно проверять актуальность ПО.



Рисунок– 10. Декомпилированный код приложения



Рисунок– 11. Файл манифеста

# ЗАКЛЮЧЕНИЕ

Программное обеспечение и дальше будет наполнять жизнь каждого человека. Его многообразие и архитектурная сложность становится с каждым годом все сложнее, поэтому важно каждому специалисту, специализирующемуся на анализе ПО идти в ногу со временем и постоянное изучать новые технологии и инструменты. Недокументированные возможности и потенциальные уязвимости всегда будут присутствовать в программах, написанных человеком, поэтому важно поддерживать специалистов по тестированию и инспектированию ПО, а также проводить тщательное документирование продукта на всех его этапах.

В данной работе был произведен общий обзор программного обеспечения разных видов, показана архитектура некоторых программ, способы их документации и рассмотрены основные виды анализа ПО на предмет скрытых возможностей.

В заключении хотелось бы отметить значимость базовых знаний и навыков необходимых для изучения любого вида ПО. В любом случае, чтобы успешно разобраться хотя бы с одной программой и понять, что она делает, а тем более понять есть ли в ней нечто опасное для пользователя требуется большой опыт в программировании и ряд общих представлений и программном обеспечении, и платформах, на которых оно функционирует. На мой взгляд, необходимо включить изучение обратного проектирования и анализа ПО в рабочие учебные программы вузов, изучающих безопасность информационных технологий, где этого еще не произошло.

# СПИСОК БИБЛИОГРАФИЧЕСКИХ ИСТОЧНИКОВ

Литература

1. Брайан У. Керниган,Роберт Л. Пайк, «Программное окружение UNIX».

2. О.В. Герман, Ю.О Герман. Программирование на Java и С# для студента (+ CD-ROM). – СПб.: БХВ-Петербург, 2005. – 512с.

3. О.Л. Голицына, И.И. Попов. Программирование на языках высокого уровня. – М.: Форум, 2010. – 496с.

4. В.Ю. Добрынин. Технологии компонентного программирования. – СпБ.: Издательство Санкт-Петербургского университета, 2004. – 216с.

5. В.А. Камаев, В.В. Костерин. Технологии программирования. – М.: Высшая школа, 2005. – 360с.

6. Barry Burd. Android Application Development All–in–One For Dummies®. – М.:, 2011. – 816с.

7. Jonathan Simon. Head First Android Development. – М.: , 2011. – 608 с.

8. Ken Ford. Android Epistemology. – М.: , 1995. – 334 с.

9. Reto Meier. Professional Android 2: Application Development. – М.:Wiley Publishing, Inc, 2010. – 576с.

10. Reto Meier. Expert Android Application Development. – М.: , 2012. – 432с.

11. Sally E. Slack. Android Companion. – М.: , 2011. – 288с.

12 Veronique Brossier. Developing Android Applications with Adobe AIR. – М.: , 2011. – 300с.