Санкт-Петербургский государственный университет

Кафедра системного программирования

Черепанов Алексей Олегович

Анализ потоков системных вызовов ОС Андроид для поиска вредоносных активностей

Курсовая работа

Научный руководитель:
ст. преп. Ханов А. Р.

Санкт-Петербург
2019

Оглавление

Введение

С развитием технологий смартфоны на базе ОС Android завоевали повсеместную популярность. В день происходит около 1,5 миллиона активаций устройств Android и миллиарды установок приложений с официального магазина Google Play. Android стала одной из самых используемых операционных систем для смартфонов и планшетов. Ежегодно увеличивается их вычислительная мощность, добавляются новые датчики и функции, а также расширяются область использования, появляются встраиваемые версии ОС, развивается интернет вещей. Это может быть использовано разработчиками приложений для улучшения взаимодействия с пользователем, но с другой стороны, расширенные возможности, предоставляемые ОС Android для большей гибкости, также привлекают внимание авторов вредоносных программ, которые сместили акцент с ПК и начали создавать вредоносные приложения, предназначенные для смартфонов.

ОС Android позволяет приложениям создавать, изменять и удалять файлы, хранящиеся в системе, в том числе и файлы, принадлежащие другим приложениям. ОС Android даёт возможность управлять телефонными вызовами, что позволяет без ведома пользователя звонить на премиум номера, похищая тем самым деньги с баланса SIM-карты, а также позволяет принимать, отправлять и удалять сообщения, что является потенциальной уязвимостью при использовании сервисов оповещений от банка, более того, существует возможность получения удалённого доступа к мобильному устройству и его функционалу вплоть до снимков экрана, камеры, записи видео и звука. Тем самым злоумышленники могут похитить личную информацию, деньги с лицевых счетов пользователя, а также причинить вред мобильному устройству, данным и приложениям, находящимся на нём.

Большинство угроз для Android приходят из сторонних приложений, скачанных не из официального магазина, в котором существуют алгоритмы Play Protect для проверки публикуемого в Play Market приложения на наличие вредоносного кода. Сторонние приложения не имеют надлежащих механизмов для защиты от вредоносных активностей. И обнаружить их могут разве что установленное на мобильном устройстве

антивирусное ПО. Однако большинство антивирусных сканеров являются сигнатурными, это означает, что данное антивирусное ПО может обнаружить только известные вирусы. У антивирусных мониторов, сканирующих сетевые пакеты и файлы системы, изменяемые приложениями в реальном времени имеют высокий процент ложных срабатываний, а также сильно нагружают систему.

1. Цель работы

Цель данной работы заключается в разработке специализированного ПО, нацеленного на мониторинг поведения Android приложений, сбор необходимой для анализа информации и выявление приложений, которые нарушают предопределенную политику безопасности. В процессе исследования предметной области были сформулированы следующие задачи:

1. Провести обзор существующих исследований в области мониторинга активности процессов на платформе Android.
2. Изучить архитектуру ядра Linux для ОС Android.
3. Освоить технологию получения трасс системных вызовов ядра Android.
4. Собрать и протестировать специализированное ядро Linux, позволяющее вести запись трасс системных вызовов.
5. Разработать конвейер программ для анализа последовательностей номеров системных вызовов из полученных трасс.
6. Полученным инструментом проанализировать собранные данные и выявить паттерны поведения вредоносных программ.

2. Обзор существующих решений

2.1 Поведенческая идентификация программ

В данной статье[1] описан алгоритм выделения шаблонов переменной длины из последовательностей системных вызовов. Эти шаблоны используются для идентификации процессов — установления того, что некоторая последовательность вызовов была сгенерирована тем же самым процессом, из которого были выделены шаблоны. Поведенческая идентификация потоков программ используется для аномального обнаружения вредоносных воздействий на систему. Результаты тестирования алгоритма показали точность идентификации до 85%, но тесты проводились только на ПК, и результаты на мобильных платформах пока неизвестны.

2.1 Монитор системных вызовов

Android-syscall-monitor[2] — это простой Android-руткит, в частности, с помощью которого можно отслеживать программы, которые обращаются к некоторым файлам, то есть данная программа перехватывает только системные вызовы на запись в директорию /data/, а целью данной работы является анализ всех системных вызовов, происходящих в системе.

2.2 Статический анализ разрешений и динамический анализ поведения приложений

В статье “Identification of malicious Android applications using kernel level system calls”[3] Dhruv Jariwala предоставляет фреймворк для автономного анализа приложений Android с использованием статического и динамического подхода к анализу. На этапе статического анализа автор выполняет декомпиляцию анализируемого приложения и извлекает разрешения из его файла «AndroidManifest». В то время как в динамическом анализе он запускает целевое приложение на эмуляторе Android, где инструмент «strace» используется для перехвата системных вызовов в процессе «zygote» и записи всех вызовов, вызываемых приложением. Особенности исследуемого приложения, извлечённые из статического и динамического анализа затем используются для классификации протестированных приложений с использованием

различных алгоритмов. Данный способ подходит для изучения поведения программы, заранее известной, как вредоносное ПО, но не для выявления вредоносных активностей в режиме “runtime”, потому что в процессе анализа используется конвейер из ресурсоёмких программ.

2.3 Статический анализ потоков системных вызовов и монитор интернет-трафика

В статье “Two-phases detecting Android malware in Android markets”[4] автор сначала использует статический анализ для обнаружения и классификации syscall в дизассемблированном коде, а затем использует network traffic monitor для выявления вредоносной активности приложений, связанной с использованием подключения к интернету. Таким образом, в данной статье не учитывается исполняемый код, загруженный из интернета в зашифрованном виде, что является потенциальной уязвимостью в алгоритме анализа.

2.4 Динамический анализ

В статье “Classifying android malware through subgraph mining”[5] авторы используют динамический анализ вредоносных приложений — запускают в специальной среде (sandbox) и строят граф системных вызовов для выявления подграфов вызовов, относящихся непосредственно к классу вредоносных. Этот подход наиболее интересен в рамках данной работы, но он требует существенной доработки в плане автоматизации и перехода от анализа отдельно выбранного вредоносного ПО, к автономному конвейеру программ для выявления неизвестных вредоносных активностей во время непосредственной работы конечного пользователя.

2.5 Утилита, ограничивающая доступ приложения к системе, применяя политики доступа для системных вызовов

Команда systrace[6] позволяет собирать и проверять информацию о времени во всех процессах, запущенных на устройстве на системном уровне. Она объединяет данные из ядра Android, такие как планировщик ЦП, активность диска и потоки приложения, для создания отчета в формате HTML, аналогичного показанному на рис.1.

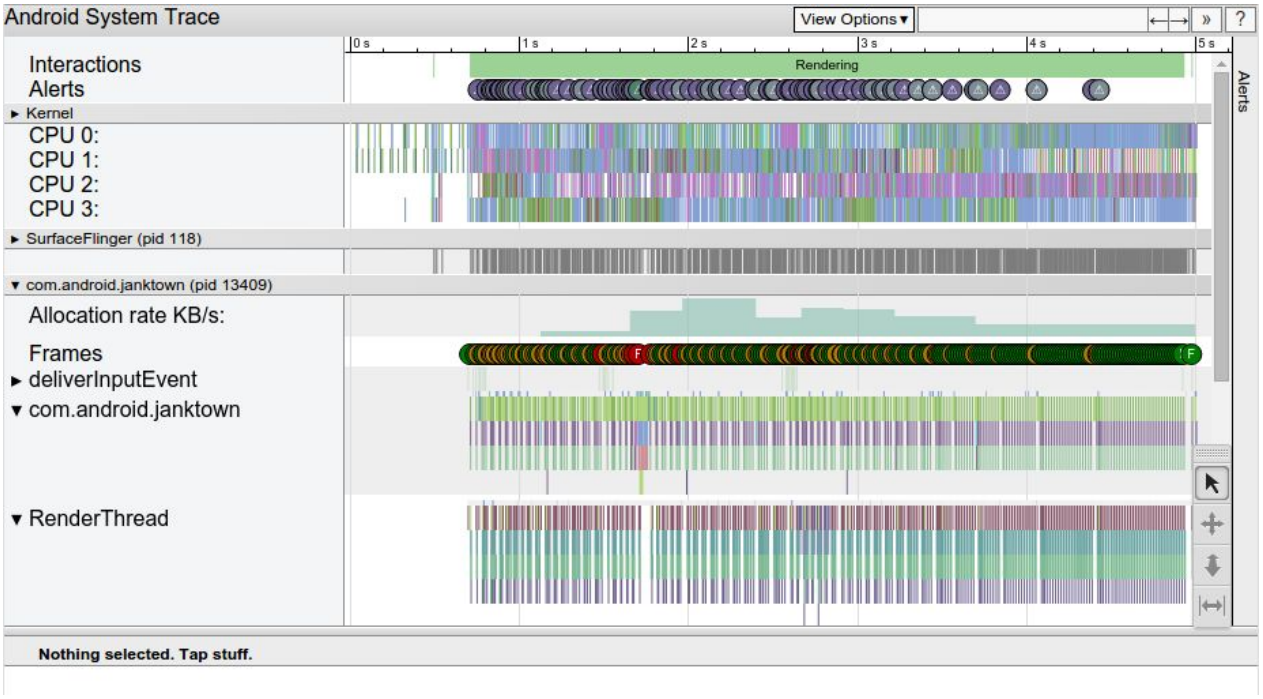

Рисунок 1. Пример HTML-отчета systrace.

На рис.1 представлена общая картина системных процессов устройства Android за определенный период времени. Программа systrace также проверяет захваченную информацию трассировки, чтобы выделить проблемы, которые она наблюдает, такие как рывок пользовательского интерфейса при отображении движения или анимации, и предоставляет рекомендации по их устранению. Однако systrace не собирает информацию о выполнении кода в процессе приложения.

3. Используемые технологии

3.1 Для сбора данных

Для сбора данных в основном были использованы технологии Android Open Source Project[7]. AOSP был выбран, как источник всех исходных кодов, включая кросс-компиляторы, эмулятор, отладчик, а также исходный код ядра и образов мобильных устройств для тестирования.

После эмпирических методов исследования оптимальной платформой для тестирования был выбран Android emulator. Имеющееся в наличии физическое устройство Alcatel Pop 4 оказалось непригодным, так как компания Alcatel не опубликовала исходный код видоизменённого ядра Linux, которое находится под защитой лицензии GNU General Public License, без которого сборка модуля ядра оказалась практически невозможной. Стоит также отметить, что разработчиками ПО была сформирована петиция с обращением опубликовать исходный код, но она была проигнорирована. Таким образом, выбор пал в сторону эмуляторов. Однако, большинство эмуляторов (например, Genymotion, BlueStacks) подразумевают высокоуровневое тестирование системы на уровне приложений, т.е. не предоставляют возможности видоизменять ядро, эмулируют архитектуру процессора x86, а также имеют закрытые репозитории с исходным кодом запускаемых устройств на платформе Android. Android emulator, взятый из репозиториев AOSP отвечает всем требуемым характеристикам, а именно:

1. Имеет гибкую настройку запуска (без графической оболочки, без сохранения снимков состояний при выключении, вывод сообщений ядра в окно терминала).
2. С помощью AVD (Android Device Manager, в комплекте Android Studio) можно создать телефон любой архитектуры процессора, версии api, графическим разрешением.
3. Эмулирует архитектуру реального устройства — arm.
4. Позволяет заменить стандартное ядро устройства на кастомное.

Специализированным ядром для Android emulator является Goldfish kernel, идущий вместе с образом мобильного устройства, но существует

также возможность скомпилировать собственное ядро и модули к нему из исходников. Ядро Goldfish kernel также было взято из репозиториев AOSP.

Для взаимодействия с эмулируемым устройством был использован adb(android debug bridge) из Android Studio.

Для компиляции был использован кросс-компилятор для архитектуры arm на хосте с архитектурой x86_64, который также был взят из репозиториев AOSP.

После анализа наиболее популярных способов снятия трасс системных вызовов было принято решение использовать ftrace[8], потому что данный механизм анализа и отладки процессов встроен в ядро Linux на уровне ассемблера: при обработке системного вызова проверяется флаг отладки, и необходимая информация записывается в специальный файл без лишних переходов и вызовов функций на более высоком уровне. Это позволяет минимизировать нагрузку на производительность системы. Более того ftrace предоставляет удобный программный интерфейс приложения для чтения полученной информации.

Для удобства был настроен конвейер скриптовых программ на языке Bash. Данный конвейер запускает эмулятор, подключает ADB, устанавливает приготовленные для тестирования приложения, после чего обследует все точки входа и по очереди запускает, включив трассировку. После имитации пользовательской активности подаётся команда к остановке снятия трасс, и лог выгружается на хост для последующего анализа.

3.2 Для анализа данных

В качестве инструмента для анализа данных был выбран проект Jupyter Notebook, так как он позволяет создавать наглядные аналитические отчёты в виде диаграмм, в совокупности с Python3, потому что предварительный анализ включает обработку файлов с логами.

4. Анализ

Артефактом сбора данных является множество файлов-логов, где среди всех процессов и их вызовов есть непосредственно то приложение, которое необходимо проанализировать. Но сначала необходимо подготовить собранные данные. Предварительный анализ состоит из двух частей. Сначала обрабатывается системный файл goldfish/arch/arm/kernel/calls.S ядра Linux, который содержит в себе определение всех имен функций, а также соответствующие им номера системных вызовов. По данному файлу строится таблица системных вызовов, которая впоследствии будет использоваться для обработки логов. Полученные логи содержат в себе много избыточной для данной работы информации, поэтому также требуют обработки. Результатом для каждого лога является имя анализируемого процесса и список номеров системных вызовов, к которым обращался данный процесс.

Для непосредственного анализа полученных данных был выбран первичный анализ.

4.1 Интенсивность вызовов различных процессов

Для каждого анализируемого процесса подсчитывается общее количество системных вызовов, а затем вычисляется количество вызовов в единицу времени.

4.2 Статистика по видам вызовов

В каждом исследуемом процессе системные вызовы группируются и подсчитываются. Используя таблицу системных вызовов мы можем обратным отображением получить имена функций, а также сопоставить количество обращений к ним. Таким образом, мы получим наглядное представление, какими системными функциями пользовался процесс.

4.3 Схожесть цепочек (n-граммы)

Для сравнения последовательностей по распределению частот вызовов был выбран алгоритм сравнения по n-граммам, описанный Стефани Форрест[9]. В качестве критерия схожести было решено искать пересечения n-грамм вызовов. Для поиска минимальной энтропии модели n для анализа берется от 2 до 12.

Из полученных цепочек выделяются словари n-грамм, а после они пресекаются, как множества, тем самым сравнивается, насколько эти словари похожи, и как часто появляются вызовы с одинаковым контекстом. Из полученных кортежей n-грамм вредоносных и безобидных приложений составляются контрольные множества, которые используются для определения степени опасности контрольных приложений, которые в анализе не участвовали.

5. Тестирование

В процессе тестирования были собраны трассы системных вызовов более 30 вредоносных приложений, взятых из открытых источников[10] и более 10 системных. 12 вредоносных приложений принадлежали к семейству JSMSers и имели схожий принцип работы. Вредоносные программы данного семейства предлагали пользователю подписаться на обновления, но вместо этого подключали платную подписку без ведома пользователя. Они отправляли СМС, а затем динамически создавали обработчик входящих сообщений, который скрывал сообщения, приходящие от подписки. Стоит отметить, что из-за архитектуры приложений в данном случае невозможно обнаружить вредоносную активность статическим анализом.

В процессе динамического анализа изучалась каждая точка входа в приложение, были созданы различные условия запуска и работы приложений. Вирусы, как и системные приложения запускались на одну минуту, в течение которой производилась имитация пользовательской активности в приложении.

За это время вредоносные приложения совершали от 5 до 10 тысяч системных вызовов, а суммарно за это время было зафиксировано около 60 тысяч обращений к системе. Анализ статистики вызовов показал, что вредоносные программы одного семейства схожи – они обращались к одним и тем же функциям, однако скорость генерации вызовов оказалась различна, что наглядно видно на примере двух вредоносных приложений семейства JSMSers: *com.romaticpost* и *com.imagepets*(см. Граф. 1).

Интенсивность вызовов вредоносных приложений семейства JSMSers была от 75 до 110 системных вызова в секунду. Анализ n-грамм потоков системных вызовов приложений, принадлежащих к семейству JSMSers, показал совпадение в 54 и 115 кортежей соответственно. Аналогично были проанализированы более 10 безвредных и системных приложений. Из полученных кортежей n-грамм вредоносных и безобидных приложений были составлены контрольные множества, которые использовались для определения степени опасности контрольных приложений, которые в анализе еще не участвовали. Эмпирическим методом было выявлено, что наименьшая энтропия модели достигается при построении анализа на

n-граммах с n=5. Результаты работы показали(см. Граф. 2), что данная модель определяет вредоносные приложения с рабочей характеристикой приёмника 80%. При этом точность составляет 73% при пороге в 75%.

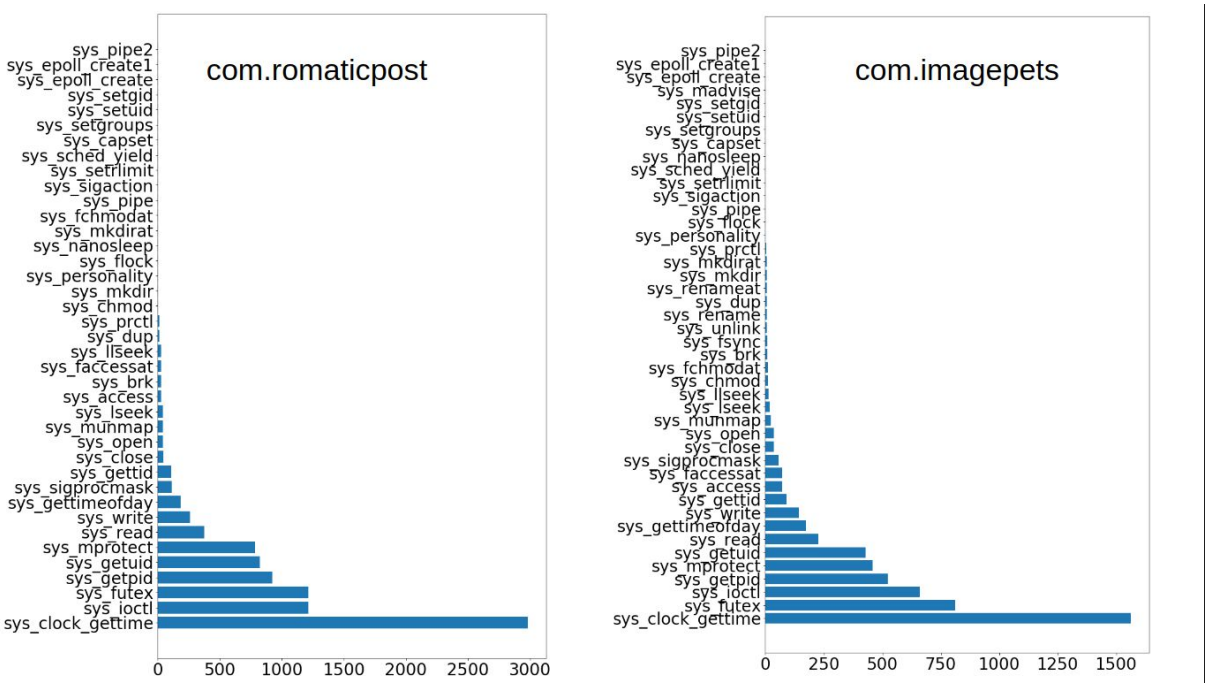

График 1: Статистика вызовов приложений com.romaticpost и com.imagepets из семейства JSMSers

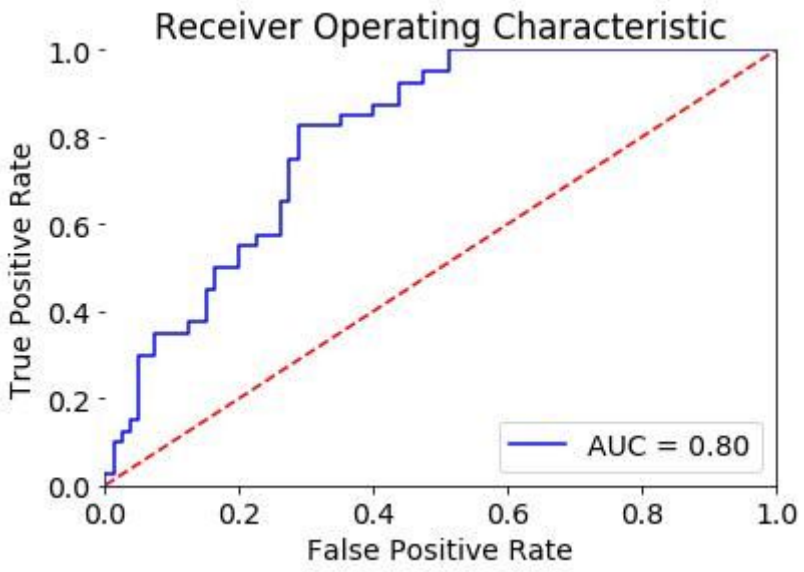

График 2: Оценка качества n-граммной модели

6. Заключение

В ходе работы были получены следующие результаты:

1. Проведён обзор существующих исследований в области мониторинга активности процессов на платформе Android.
2. Изучена архитектура ядра Linux для ОС Android в рамках поставленной проблемы.
3. Освоена технология получения трасс системных вызовов ядра Android.
4. Собрано и протестировано специализированное ядро Linux, позволяющее вести запись трасс системных вызовов.
5. Разработан конвейер программ для анализа последовательностей номеров системных вызовов из полученных трасс.
6. Проведены тесты на вредоносных и системных приложениях.

Резкое увеличение использования устройств на платформе Android привело к созданию и распространению огромного количества вредоносных программ для Android. В этом тезисе был предложен фреймворк для сбора и анализа приложений Android с использованием методов динамического анализа. Эффективность предложенного фреймворка была подтверждена тестированием на специально собранном ядре Linux, позволяющим снимать трассы системных вызовов, где он показал несущественную нагрузку на производительность системы.

Список литературы

[1] М. В. Баклановский, А. Р. Ханов. Поведенческая идентификация программ. Модел. и анализ информ. систем — 2014 — URL: http://www.mathnet.ru/links/eef739addc8f577840dd2d80513b1e67/mais417.pdf

[2] Andrea Sindoni. Android syscall monitor. — 2016 — URL: https://github.com/invictus1306/Android-syscall-monitor

[3] Dhruv Jariwala. Identification of malicious Android applications using kernel level system calls — 2014 — URL: https://pdfs.semanticscholar.org/126c/2a196236fe08f6051dc74f828802728340d8.pdf

[4] Xin Su, Jiuchuan Lin, Fihui Shen, Yi Zheng. Two-phases detecting Android malware in Android markets — 2019 — URL: https://books.google.ru/books?id=IWR2DwAAQBAJ&lpg=PA391&hl=ru&pg=PA389#v=onepage&q&f=false

[5] Fabio Martinelli, Andrea Saracino, Daniel Sgandurra. Classifying Android Malware through Subgraph Mining — 2014 — URL: https://books.google.ru/books?id=Rde6BQAAQBAJ&lpg=PA270&hl=ru&pg=PA268#v=onepage&q&f=false

[6] Systrace — 2018 — URL: https://developer.android.com/studio/command-line/systrace

[7] AOSP — 2018 — URL: https://source.android.com/

[8] ftrace — 2008 — URL: https://www.kernel.org/doc/Documentation/trace/ftrace.txt

[9] Steven A. Hofmeyr, Stephanie Forrest, Anil Somayaji. Intrusion Detection using Sequences of System Calls — 1998 — URL: https://www.cs.unm.edu/~forrest/publications/jcs-sequences-of-system-calls-98.pdf

[10] android-malware — 2019 — URL: https://github.com/ashishb/android-malware