|  |
| --- |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  РОССИЙСКОЙ ФЕДЕРАЦИИ  Федеральное государственное бюджетное образовательное учреждение высшего образования  «МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» |
| Методические рекомендации по проекту № 42  Мобильное приложение Ак Барс  Для:  «Команды фронтенд-разработчиков» |
|  |
| Выполнили:  Студенты 3 курса  Группы 181-352  Захаров В.А.  Констанитов Д.А. |

Москва – 2020

**Советы по безопасности для Android приложений**

Следующие основные функции безопасности помогают создавать безопасные приложения:

* Песочница приложения Android, которая изолирует данные вашего приложения и выполнение кода от других приложений.
* Платформа приложения с надежной реализацией общих функций безопасности, таких как криптография, разрешения и безопасный IPC .
* Такие технологии, как ASLR, NX, ProPolice, safe\_iop, OpenBSD dlmalloc, OpenBSD calloc и Linux mmap\_min\_addr для снижения рисков, связанных с распространенными ошибками управления памятью.
* Зашифрованная файловая система, которую можно включить для защиты данных на утерянных или украденных устройствах.
* Предоставляемые пользователем разрешения для ограничения доступа к системным функциям и пользовательским данным.
* Определяемые приложением разрешения для управления данными для каждого приложения.

**Хранение данных**

Наиболее распространенная проблема безопасности приложения на Android - это то, доступны ли данные, которые вы сохраняете на устройстве, другим приложениям. Есть три основных способа сохранить данные на устройстве:

* Внутреннее хранилище.
* Внешнее хранилище.
* Контент-провайдеры.

**Использование внутренней памяти**

По умолчанию файлы, которые вы создаете во внутренней памяти, доступны только вашему приложению. Android реализует эту защиту, и этого достаточно для большинства приложений.

Как правило, избегайте режимов MODE\_WORLD\_WRITEABLE или MODE\_WORLD\_READABLE для файлов IPC, потому что они не предоставляют возможности ограничить доступ к данным для определенных приложений и не обеспечивают никакого управления форматом данных. Если вы хотите поделиться своими данными с другими процессами приложения, вместо этого рассмотрите возможность использования поставщика контента, который предлагает разрешения на чтение и запись для других приложений и может предоставлять динамические разрешения в каждом конкретном случае.

Чтобы обеспечить дополнительную защиту конфиденциальных данных, вы можете зашифровать локальные файлы с помощью библиотеки безопасности. Эта мера может обеспечить защиту потерянного устройства без шифрования файловой системы.

**Использование внешнего хранилища**

Файлы, созданные на внешнем хранилище, например на SD-картах, доступны для чтения и записи во всем мире. Поскольку внешнее хранилище может быть удалено пользователем, а также изменено любым приложением, не храните конфиденциальную информацию, используя внешнее хранилище.

Чтобы читать и записывать файлы во внешнем хранилище более безопасным способом, рассмотрите возможность использования библиотеки безопасности, которая предоставляет класс: *EncryptedFile*

При обработке данных из внешнего хранилища следует выполнять проверку ввода, как если бы вы поступали с данными из любого ненадежного источника. Вы не должны хранить исполняемые файлы или файлы классов на внешнем хранилище до динамической загрузки. Если ваше приложение действительно извлекает исполняемые файлы из внешнего хранилища, файлы должны быть подписаны и криптографически проверены перед динамической загрузкой.

**Использование поставщиков контента**

Поставщики контента предлагают механизм структурированного хранения, который может быть ограничен вашим собственным приложением или экспортирован, чтобы разрешить доступ другим приложениям. Если вы не собираетесь предоставлять другим приложениям доступ к своим ContentProvider, отметьте их как android:exported=false в манифесте приложения. В противном случае установите для android:exported атрибута значение, true чтобы разрешить другим приложениям доступ к сохраненным данным.

При создании файла, ContentProvider который экспортируется для использования другими приложениями, вы можете указать одно разрешение для чтения и записи или вы можете указать отдельные разрешения для чтения и записи. Вы должны ограничить свои разрешения теми, которые необходимы для выполнения поставленной задачи. Имейте в виду, что обычно проще добавить разрешения позже, чтобы раскрыть новые функции, чем отнять их и повлиять на существующих пользователей.

Если вы используете поставщика контента для обмена данными только между вашими собственными приложениями, предпочтительно использовать android:protectionLevel атрибут, установленный для signature защиты. Разрешения на подпись не требуют подтверждения пользователя, поэтому они обеспечивают лучший пользовательский интерфейс и более контролируемый доступ к данным поставщика контента, когда приложения, получающие доступ к данным, подписаны одним и тем же ключом.

Контент - провайдеры могут также обеспечить более детальный доступ, объявив android:grantUriPermissions атрибут и с помощью FLAG\_GRANT\_READ\_URI\_PERMISSION и FLAG\_GRANT\_WRITE\_URI\_PERMISSION флагов в Intent объекте , который активирует компонент. Объем этих разрешений может быть дополнительно ограничен <grant-uri-permission> элементом.

При доступе к контент - провайдеру, использовать параметризованные методы запроса , такие как query(), update(), и , delete() чтобы избежать возможных инъекции SQL из ненадежных источников. Обратите внимание, что использования параметризованных методов недостаточно, если selection аргумент создается путем объединения пользовательских данных перед их отправкой в ​​метод.

Не стоит быть уверенным в безопасности в отношении разрешения запись. Разрешение на запись позволяет использовать операторы SQL, которые позволяют подтверждать некоторые данные с помощью творческих WHERE предложений и анализа результатов. Например, злоумышленник может проверить наличие определенного номера телефона в журнале вызовов, изменив строку, только если этот номер телефона уже существует. Если данные поставщика контента имеют предсказуемую структуру, разрешение на запись может быть эквивалентно предоставлению как чтения, так и записи.

**Использование разрешений**

Поскольку Android отделяет приложения друг от друга, приложения должны явно совместно использовать ресурсы и данные. Они делают это, объявляя разрешения, необходимые для дополнительных возможностей, не предоставляемых базовой песочницей, включая доступ к функциям устройства, таким как камера.

**Запросы разрешений**

Вы должны минимизировать количество разрешений, запрашиваемых вашим приложением. Ограничение доступа к конфиденциальным разрешениям снижает риск непреднамеренного неправильного использования этих разрешений, улучшает принятие пользователями и делает ваше приложение менее уязвимым для злоумышленников. Как правило, если для работы вашего приложения не требуется разрешение, не запрашивайте его. Если есть функция, без которой приложение не может работать, объявите ее с помощью <uses-feature> элемента в файле манифеста.

Если возможно спроектировать ваше приложение способом, не требующим каких-либо разрешений, это предпочтительнее. Например, вместо того, чтобы запрашивать доступ к информации об устройстве для создания уникального идентификатора, создайте GUID для своего приложения (см. Раздел « Обработка пользовательских данных» ). Или, вместо использования внешнего хранилища (для которого требуется разрешение), храните данные во внутреннем хранилище.

Помимо запроса разрешений, ваше приложение может использовать этот <permission> элемент для защиты IPC, который чувствителен к безопасности и доступен для других приложений, таких как ContentProvider. В общем, мы рекомендуем по возможности использовать средства управления доступом, отличные от разрешений, подтвержденных пользователем, поскольку разрешения могут сбивать пользователей с толку. Например, рассмотрите возможность использования уровня защиты сигнатур для разрешений на взаимодействие IPC между приложениями, предоставляемых одним разработчиком.

Не допускайте утечки данных, защищенных разрешениями. Это происходит, когда ваше приложение предоставляет данные через IPC, которые доступны только потому, что ваше приложение имеет разрешение на доступ к этим данным. Клиенты интерфейса IPC вашего приложения могут не иметь такого же разрешения на доступ к данным.

**Создание разрешений**

Как правило, вы должны стремиться определить как можно меньше разрешений при соблюдении требований безопасности. Создание нового разрешения относительно редко для большинства приложений, потому что определенные системой разрешения охватывают множество ситуаций. При необходимости выполните проверки доступа, используя существующие разрешения.

Если вам необходимо создать новое разрешение, подумайте, сможете ли вы выполнить свою задачу с уровнем защиты подписи . Разрешения подписи прозрачны для пользователя и разрешают доступ только приложениям, подписанным тем же разработчиком, что и приложение, выполняющее проверку разрешений. Если новое разрешение по-прежнему требуется, оно объявляется в манифесте приложения с помощью <permission> элемента. Приложения, которые хотят использовать новое разрешение, могут ссылаться на него, добавляя каждый <uses-permission> элемент в свои соответствующие файлы манифеста. Вы также можете динамически добавлять разрешения с помощью addPermission() метода.

Если вы создаете разрешение с опасным уровнем защиты, возникает ряд сложностей, которые необходимо учитывать:

* Разрешение должно иметь строку, которая кратко выражает пользователю решение по безопасности, которое он должен принять.
* Строка разрешения должна быть локализована на множество разных языков.
* Пользователи могут отказаться от установки приложения, потому что разрешение сбивает с толку или воспринимается как рискованное.
* Приложения могут запрашивать разрешение, если создатель разрешения не установлен.

Каждый из них представляет собой серьезную нетехническую задачу для вас как разработчика, а также сбивает с толку ваших пользователей, поэтому мы не рекомендуем использовать опасный уровень разрешений.

**Использование сети**

Сетевые транзакции по своей природе опасны для безопасности, потому что они связаны с передачей данных, которые потенциально являются конфиденциальными для пользователя. Люди все больше осознают проблемы конфиденциальности мобильного устройства, особенно когда устройство выполняет сетевые транзакции, поэтому очень важно, чтобы в вашем приложении реализованы все передовые методы обеспечения безопасности данных пользователя в любое время.

**Используйте IP-сеть**

Сеть на Android не сильно отличается от других сред Linux. Ключевым моментом является обеспечение использования соответствующих протоколов для конфиденциальных данных, например HttpsURLConnectionдля безопасного веб-трафика. Вы должны использовать HTTPS поверх HTTP везде, где HTTPS поддерживается на сервере, поскольку мобильные устройства часто подключаются к незащищенным сетям, например к общедоступным точкам доступа Wi-Fi.

Аутентифицированная, зашифрованная связь на уровне сокета может быть легко реализована с помощью SSLSocket класса. Учитывая частоту, с которой устройства Android подключаются к незащищенным беспроводным сетям с помощью Wi-Fi, использование защищенных сетей настоятельно рекомендуется для всех приложений, которые обмениваются данными по сети.

Некоторые приложения используют сетевые порты localhost для обработки конфиденциальных IPC. Вы не должны использовать этот подход, потому что эти интерфейсы доступны другим приложениям на устройстве. Вместо этого используйте механизм Android IPC, где возможна аутентификация, например, с помощью Service. Привязка к INADDR\_ANY хуже, чем использование обратной петли, потому что тогда ваше приложение может получать запросы откуда угодно.

Убедитесь, что вы не доверяете данным, загруженным по HTTP или другим небезопасным протоколам. Это включает в себя проверку ввода WebViewи любых ответов на намерения, выданные по протоколу HTTP.

**Использование телефонной сети**

Протокол SMS был в первую очередь разработан для связи между пользователями и не очень подходит для приложений, которые хотят передавать данные. Из-за ограничений SMS вам следует использовать Google Cloud Messaging (GCM) и IP-сеть для отправки сообщений с данными с веб-сервера в ваше приложение на пользовательском устройстве.

Помните, что SMS не зашифрованы и не проходят строгую аутентификацию ни в сети, ни на устройстве. В частности, любой получатель SMS должен ожидать, что злоумышленник мог отправить SMS в ваше приложение. Не полагайтесь на неаутентифицированные данные SMS для выполнения конфиденциальных команд. Кроме того, вы должны знать, что SMS-сообщения могут подвергаться спуфингу и / или перехвату в сети. На самом устройстве под управлением Android SMS-сообщения передаются как широковещательные, поэтому они могут быть прочитаны или захвачены другими приложениями, у которых есть READ\_SMS разрешение.

**Проверка пользовательского ввода**

Недостаточная проверка входных данных - одна из наиболее распространенных проблем безопасности, влияющих на приложения, независимо от того, на какой платформе они работают. В Android предусмотрены меры противодействия на уровне платформы, которые снижают подверженность приложений проблемам проверки ввода, и вам следует использовать эти функции там, где это возможно. Также обратите внимание, что выбор языков с типобезопасностью снижает вероятность проблем с проверкой ввода.

Если вы используете собственный код, любые данные, считанные из файлов, полученные по сети или полученные от IPC, могут создать угрозу безопасности. Наиболее частые проблемы - это переполнение буфера, ошибки использования после освобождения и единичные ошибки. Android предоставляет ряд технологий, таких как ASLR и DEP, которые снижают возможность использования этих ошибок, но не решают основную проблему. Вы можете предотвратить эти уязвимости, осторожно обращаясь с указателями и управляя буферами.

Динамические строковые языки, такие как JavaScript и SQL, также подвержены проблемам проверки ввода из-за escape-символов и внедрения скриптов.

Если вы используете данные в запросах, которые отправляются в базу данных SQL или поставщику контента, внедрение SQL может быть проблемой. Лучшая защита - использовать параметризованные запросы, как описано в предыдущем разделе о поставщиках контента. Ограничение разрешений только для чтения или только для записи также может снизить вероятность ущерба, связанного с внедрением SQL.

Если вы не можете использовать описанные выше функции безопасности, убедитесь, что вы используете хорошо структурированные форматы данных и убедитесь, что данные соответствуют ожидаемому формату. Хотя блокировка определенных символов или выполнение замены символов может быть эффективной стратегией, эти методы на практике подвержены ошибкам, и их следует по возможности избегать.

**Обработка пользовательских данных**

В общем, лучший подход к безопасности пользовательских данных - свести к минимуму использование API-интерфейсов, которые обращаются к конфиденциальным или личным данным пользователя. Если у вас есть доступ к пользовательским данным и вы можете избежать их хранения или передачи, не храните и не передавайте данные. Подумайте, есть ли способ, которым логика вашего приложения может быть реализована с использованием хеш-функции или необратимой формы данных. Например, ваше приложение может использовать хэш адреса электронной почты в качестве первичного ключа, чтобы избежать передачи или сохранения адреса электронной почты. Это снижает вероятность непреднамеренного раскрытия данных, а также снижает вероятность попытки злоумышленников использовать ваше приложение.

Если ваше приложение получает доступ к личной информации, такой как пароли или имена пользователей, имейте в виду, что в некоторых юрисдикциях от вас может потребоваться предоставить политику конфиденциальности, объясняющую использование и хранение этих данных. Следование передовой практике безопасности по минимизации доступа к пользовательским данным также может упростить соблюдение требований.

Вы также должны учитывать, может ли ваше приложение непреднамеренно раскрывать личную информацию другим сторонам, например сторонним компонентам для рекламы или сторонним сервисам, используемым вашим приложением. Если вы не знаете, почему компоненту или услуге требуется личная информация, не предоставляйте ее. В целом, ограничение доступа вашего приложения к личной информации снижает вероятность возникновения проблем в этой области.

Если вашему приложению требуется доступ к конфиденциальным данным, оцените, нужно ли вам передавать их на сервер, или вы можете запустить операцию на клиенте. Рассмотрите возможность запуска любого кода с использованием конфиденциальных данных на клиенте, чтобы избежать передачи пользовательских данных. Кроме того, убедитесь, что вы непреднамеренно не предоставляете данные пользователя другим приложениям на устройстве через чрезмерно разрешающий IPC, файлы с возможностью записи для всех или сетевые сокеты. Чрезмерно разрешающий IPC - это особый случай утечки данных, защищенных разрешениями, который обсуждается в разделе «Запрос разрешений».

Если требуется GUID, создайте большой уникальный номер и сохраните его. Не используйте идентификаторы телефона, такие как номер телефона или IMEI, которые могут быть связаны с личной информацией.

Будьте осторожны при записи в журналы на устройстве. В Android журналы являются общим ресурсом и доступны приложению с соответствующим READ\_LOGS разрешением. Несмотря на то, что данные журнала телефона являются временными и стираются при перезагрузке, неправильное ведение журнала пользовательской информации может непреднамеренно привести к утечке пользовательских данных в другие приложения. В дополнение к тому, что PII не регистрируется, производственные приложения должны ограничивать использование журналов. Чтобы легко реализовать это, используйте флаги отладки и настраиваемые Log классы с легко настраиваемыми уровнями ведения журнала.

**Использование WebView**

Поскольку WebView потребляет веб-контент, который может включать HTML и JavaScript, неправильное использование может вызвать общие проблемы веб-безопасности, такие как межсайтовый скриптинг (внедрение JavaScript). Android включает в себя ряд механизмов для уменьшения объема этих потенциальных проблем, ограничивая возможности WebView минимальными функциями, необходимыми для вашего приложения.

Если ваше приложение не использует JavaScript напрямую в a WebView, не вызывайте setJavaScriptEnabled(). В некоторых примерах кода используется этот метод, который вы можете использовать в производственном приложении, поэтому удалите этот вызов метода, если он не требуется. По умолчанию WebViewJavaScript не выполняется, поэтому выполнение межсайтовых сценариев невозможно.

Используйте addJavaScriptInterface() с особой осторожностью, поскольку он позволяет JavaScript вызывать операции, которые обычно зарезервированы для приложений Android. Если вы его используете, открывайте addJavaScriptInterface() только те веб-страницы, все данные с которых заслуживают доверия. Если разрешен ненадежный ввод, ненадежный JavaScript может вызывать методы Android в вашем приложении. В общем, мы рекомендуем использовать addJavaScriptInterface()только JavaScript, который содержится в APK вашего приложения.

Если ваше приложение получает доступ к конфиденциальным данным с помощью WebView, вы можете использовать этот clearCache()метод для удаления любых файлов, хранящихся локально. Вы также можете использовать заголовки на стороне сервера, например, no-cache чтобы указать, что приложение не должно кэшировать определенный контент.

Устройства, работающие на платформах старше Android 4.4 (уровень API 19), используют версию, webkit которая имеет ряд проблем с безопасностью. В качестве обходного пути, если ваше приложение работает на этих устройствах, оно должно подтверждать, что WebView объекты отображают только доверенный контент. Чтобы убедиться, что ваше приложение не подвержено потенциальным уязвимостям в SSL, используйте обновляемый Provider объект безопасности, как описано в разделе Обновление вашего поставщика безопасности для защиты от эксплойтов SSL. Если ваше приложение должно отображать контент из открытой сети, подумайте о предоставлении собственного средства визуализации, чтобы вы могли поддерживать его в актуальном состоянии с помощью последних исправлений безопасности.

**Обработка учетных данных**

Чтобы фишинговые атаки были более заметными и с меньшей вероятностью были успешными, сведите к минимуму частоту запроса учетных данных пользователя. Вместо этого используйте токен авторизации и обновите его.

По возможности не храните имена пользователей и пароли на устройстве. Вместо этого выполните первоначальную аутентификацию, используя имя пользователя и пароль, предоставленные пользователем, а затем используйте краткосрочный токен авторизации для конкретной службы.

Доступ к службам, доступным нескольким приложениям, следует осуществлять с помощью AccountManager. Если возможно, используйте AccountManager класс для вызова облачной службы и не храните пароли на устройстве.

После использования AccountManager для получения Account, используйте CREATOR перед передачей учетных данных, чтобы случайно не передать учетные данные не тому приложению.

Если учетные данные используются только приложениями, которые вы создаете, вы можете проверить приложение, которое обращается к AccountManagerusing checkSignature(). В качестве альтернативы, если только одно приложение использует учетные данные, вы можете использовать KeyStore для хранения.

**Использование криптографию**

Помимо обеспечения изоляции данных, поддержки полного шифрования файловой системы и обеспечения безопасных каналов связи, Android предоставляет широкий спектр алгоритмов для защиты данных с помощью криптографии.

В общем, вы должны знать, какие провайдеры безопасности Java Cryptography Architecture (JCA) использует ваше программное обеспечение. Попробуйте использовать самый высокий уровень уже существующей реализации фреймворка, который может поддерживать ваш вариант использования. Если возможно, используйте поставщиков, предоставленных Google, в указанном Google порядке.

Для более безопасного чтения и записи локальных файлов используйте библиотеку безопасности.

Если вам нужно безопасно получить файл из известного сетевого местоположения, может подойти простой HTTPS-URI, который не требует знания криптографии. Если вам нужен безопасный туннель, подумайте об использовании HttpsURLConnection или SSLSocket вместо написания собственного протокола. Если вы используете SSLSocket, имейте в виду, что он не выполняет проверку имени хоста.

Если вы обнаружите, что вам необходимо реализовать свой собственный протокол, вам не следует реализовывать собственные криптографические алгоритмы. Используйте существующие криптографические алгоритмы, такие как реализации AES и RSA, предоставленные в Cipherклассе. Кроме того, вам следует придерживаться следующих рекомендаций:

* Используйте 256-битный AES в коммерческих целях. (Если недоступно, используйте 128-битный AES.)
* Для криптографии с эллиптической кривой (EC) используйте 224- или 256-битные размеры открытого ключа.
* Знайте, когда использовать режимы блока CBC, CTR или GCM.
* Избегайте повторного использования IV / счетчика в режиме CTR. Убедитесь, что они криптографически случайны.

При использовании шифрования реализуйте целостность, используя режим CBC или CTR с одной из следующих функций:

* HMAC-SHA1
* HMAC-SHA-256
* HMAC-SHA-512
* Режим GCM

Используйте безопасный генератор случайных чисел SecureRandom для инициализации любых криптографических ключей, сгенерированных KeyGenerator. Использование ключа, который не генерируется с помощью безопасного генератора случайных чисел, значительно ослабляет силу алгоритма и может сделать возможным автономные атаки.

Если вам необходимо сохранить ключ для повторного использования, используйте механизм, например KeyStore, который обеспечивает механизм для длительного хранения и извлечения криптографических ключей.

**Использование межпроцессного взаимодействия**

Некоторые приложения пытаются реализовать IPC с использованием традиционных методов Linux, таких как сетевые сокеты и общие файлы. Однако вместо этого вы должны использовать функции системы Android для IPC, такие как Intent, Binderили Messengerс Service, и BroadcastReceiver. Механизмы Android IPC позволяют вам проверять подлинность приложения, подключающегося к вашему IPC, и устанавливать политику безопасности для каждого механизма IPC.

Многие элементы безопасности являются общими для всех механизмов IPC. Если ваш механизм IPC не предназначен для использования другими приложениями, установите для android:exported атрибута значение false в элементе манифеста компонента, например, для <service> элемента. Это полезно для приложений, которые состоят из нескольких процессов с одним и тем же UID, или если вы на позднем этапе разработки решили, что на самом деле не хотите предоставлять функциональность как IPC, но не хотите переписывать код.

Если ваш IPC доступен для других приложений, вы можете применить политику безопасности с помощью <permission> элемента. Если IPC находится между вашими собственными отдельными приложениями, подписанными одним и тем же ключом, предпочтительнее использовать signature разрешения уровня в android:protectionLevel.

**Использование сервисов**

Сервис часто используется для предоставления функций другим приложениям. Каждый класс обслуживания должен иметь соответствующее <service> объявление в файле манифеста.

По умолчанию службы не экспортируются и не могут быть вызваны никаким другим приложением. Однако, если вы добавляете какие-либо фильтры намерений в объявление службы, оно экспортируется по умолчанию. Лучше всего явно объявить android:exported атрибут, чтобы убедиться, что он ведет себя так, как вам нужно. Услуги также можно защитить с помощью android:permission атрибута. Таким образом, другие приложения должны объявить соответствующий <uses-permission> элемент в своем собственном манифесте, чтобы иметь возможность запускать, останавливать или связываться с сервисом.

Служба может защитить отдельные вызовы IPC в нем с помощью разрешений, вызывая checkCallingPermission() перед выполнением реализации этого вызова. Вы должны использовать декларативные разрешения в манифесте, поскольку они менее подвержены контролю.

**Динамическая загрузка кода**

Мы настоятельно не рекомендуем загружать код из-за пределов APK вашего приложения. Это значительно увеличивает вероятность взлома приложения из-за внедрения кода или подделки кода. Это также усложняет управление версиями и тестирование приложений. Это также может сделать невозможным проверку поведения приложения, поэтому в некоторых средах это может быть запрещено.

Если ваше приложение динамически загружает код, самое важное, что нужно помнить о динамически загружаемом коде, - это то, что он работает с теми же разрешениями безопасности, что и APK приложения. Пользователь принимает решение об установке вашего приложения на основе вашей личности, и пользователь ожидает, что вы предоставите любой код, выполняемый в приложении, включая код, который динамически загружается.

Основной риск безопасности, связанный с динамической загрузкой кода, заключается в том, что код должен поступать из проверяемого источника. Если модули включены непосредственно в ваш APK, они не могут быть изменены другими приложениями. Это верно независимо от того, является ли код собственной библиотекой или классом, загружаемым с использованием DexClassLoader. Многие приложения пытаются загрузить код из небезопасных мест, например, загруженных из сети по незашифрованным протоколам или из доступных для записи мест, таких как внешнее хранилище. Эти местоположения могут позволить кому-либо в сети изменять передаваемый контент или другому приложению на устройстве пользователя для изменения контента на устройстве.

**Использование виртуализации**

Dalvik - это виртуальная машина времени выполнения (ВМ) Android. Dalvik был создан специально для Android, но многие проблемы, связанные с безопасным кодом в других виртуальных машинах, также применимы к Android. В общем, вам не следует беспокоиться о проблемах безопасности, связанных с виртуальной машиной. Ваше приложение работает в безопасной среде песочницы, поэтому другие процессы в системе не могут получить доступ к вашему коду или личным данным.

**Безопасность в коде**

В общем, вы должны использовать Android SDK для разработки приложений, а не использовать собственный код с Android NDK. Приложения, созданные с использованием собственного кода, более сложны, менее переносимы и более подвержены распространенным ошибкам повреждения памяти, таких как переполнение буфера.