УДК 004.021, 681.518.22
doi: 10.26583/bit.2024.2.07

Антон С. Черевань[1], Алексей П. Лапсарь[2]
Ростовский государственный экономический университет,
ул. Большая Садовая, 69, Ростов-на-Дону, 344002, Россия
[1]e-mail: antuere@yandex.ru, https://orcid.org/0009-0004-5410-411X
[2]e-mail: lapsar1958@mail.ru, https://orcid.org/0009-0005-3166-0524

ПРОБЛЕМЫ ПЛАВАЮЩИХ УЯЗВИМОСТЕЙ ПРИ ОБЕСПЕЧЕНИИ БЕЗОПАСНОСТИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Аннотация. Статья посвящена разработке метода выявления и устранения плавающих уязвимостей в мобильных приложениях, рассмотрена их природа, исследованы существующие способы обнаружения и нейтрализации плавающих уязвимостей, отмечены их недостатки. Разработанный метод основан на анализе состояния приложения в различные моменты времени и сравнении его с эталонным состоянием. Предложенный алгоритм включает фиксацию начального состояния, дискретный анализ, выявление и идентификацию уязвимостей, их устранение, а также предотвращение дополнительных рисков. Область применения результатов включает разработку и обеспечение безопасности мобильных приложений под управлением операционной системы Android. Выводы статьи подтверждают важность обнаружения и предотвращения плавающих уязвимостей для обеспечения высокого уровня информационной безопасности. Разработанный метод позволяет эффективно выявлять и устранять уязвимости, что способствует созданию надежных и защищенных приложений для пользователей мобильных устройств.
Ключевые слова: плавающие уязвимости, гейзенбаги, мобильные приложения, Android, информационная безопасность.
Для цитирования: ЧЕРЕВАНЬ, Антон С.; ЛАПСАРЬ, Алексей П. ПРОБЛЕМЫ ПЛАВАЮЩИХ УЯЗВИМОСТЕЙ ПРИ ОБЕСПЕЧЕНИИ БЕЗОПАСНОСТИ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ. Безопасность информационных технологий, [S.l.], т. 31, № 2, с. 111–120, 2024. ISSN 2074-7136. URL: https://bit.spels.ru/index.php/bit/article/view/1630. DOI: http://dx.doi.org/10.26583/bit.2024.2.07.

Anton S. Cherevan[1], Alexey P. Lapsar[2]
Rostov State University of Economics,
B. Sadovaya str., 69, Rostov-on-Don, 344002, Russia
[1]e-mail: antuere@yandex.ru, https://orcid.org/0009-0004-5410-411X
[2]e-mail: lapsar1958@mail.ru, https://orcid.org/0009-0005-3166-0524

Problems of floating vulnerabilities in ensuring mobile application security

Abstract. The article is focused on the development of a method for detecting and eliminating floating vulnerabilities in mobile applications, their nature is described, and existing methods of detecting and neutralizing floating vulnerabilities are investigated, their defects are noted. The developed method is based on analyzing the state of the application at different points in time and comparing it with a reference state. The proposed algorithm includes fixing the initial state, discrete analysis, detection and identification of vulnerabilities, their elimination, and prevention of additional risks. The scope of application of the results includes the developing and securing of mobile applications for Android operating system. The conclusions of the paper confirm the importance of detecting and preventing floating vulnerabilities to ensure a high level of information protection. The developed method allows to effectively detect and eliminate vulnerabilities, which contributes to the creation of secure and protected applications for users of mobile devices.
Keywords: floating vulnerabilities, geisenbugs, mobile apps, Android, information security.

For citation: CHEREVAN, Anton S.; LAPSAR, Alexey P. Problems of floating vulnerabilities in ensuring mobile application security. IT Security (Russia), [S.l.], v. 31, no. 2, p. 111–120, 2024. ISSN 2074-7136. URL: https://bit.spels.ru/index.php/bit/article/view/1630. DOI: http://dx.doi.org/10.26583/bit.2024.2.07.

Введение

Стратегия развития информационного общества в Российской Федерации на 2017 - 2030 годы, предполагает приоритетное развитие информационных и информационно-телекоммуникационных технологий[1]. Важнейшим условием реализации данной задачи определено создание отечественных программных продуктов высокого уровня. Одним из актуальных и бурно развивающихся направлений в области программирования является разработка различного рода приложений для мобильных телефонов. В последние годы в этой области прослеживается устойчивый тренд на повышение их комплексности, проявляющейся, в том числе, в увеличении кодовой базы программного продукта [1].

Возрастание объема и сложности мобильных приложений выдвигает на первый план проблему обеспечения их надежного функционирования. Исходя из этого задача обеспечения информационной безопасности прикладных программ становится одной из ключевых для разработчиков. Вопросы информационной безопасности особенно актуальны для приложений, связанных с обеспечением функционирования критической информационной инфраструктуры [2]. Так, в прошедшем году, хакерским атакам подверглась треть объектов критической информационной инфраструктуры нашей страны [3]. Угрозы для информационной безопасности в мобильных приложениях могут иметь различную природу и быть обусловленными различными факторами. Например, атаки на мобильные устройства, использующие механизм автоматической настройки, могут внедрять вредоносное программное обеспечение, атаки с использованием технологии Wi-Fi позволяют установить соединение с устройством, с целью выполнения различных команд на нем, а уязвимости технологии NFC могут использоваться злоумышленниками с целью выведения его из строя [4–6]. Относительно недавно появились более изощренные и неочевидные атаки, в частности, атаки по сторонним каналам с использованием радиоустановок являются мощным инструментом взлома криптографических алгоритмов путем изменения физических величин во время выполнения этих алгоритмов на телефонах [7]. Одним из непременных условий успешности проводимых атак на мобильные системы является наличие уязвимостей различной природы. В настоящее время наиболее опасными в силу их непредсказуемости являются угрозы, использующие гейзенбаги (названы в честь физика Вернера Гейзенберга) или плавающие уязвимости [8]. Принцип неопределенности и эффект наблюдателя, сформулированные Гейзенбергом, позволяют наиболее полно описать уязвимости, которые в контексте мобильных приложений представляют собой нестабильные ошибки, возникающие и исчезающие вследствие целого ряда причин. Таким образом, гейзенбаги или плавающие уязвимости это – баги, которые самопроизвольно исчезают или меняют свое поведение при попытках их обнаружения. Такие уязвимости практически неуловимы в процессе тестирования и дебаггинга программ, однако спонтанно проявляются и исчезают в процессе реальной эксплуатации приложения. Вследствие этого их обнаружение критично для обеспечения надежного функционирования и обеспечения информационной безопасности мобильных приложений в процессе штатной эксплуатации.

Цель данной работы – разработка метода обнаружения плавающих уязвимостей в мобильных приложениях на основе анализа причин и условий появления.

[1]Указ Президента РФ от 9 мая 2017 г. N 203 "О Стратегии развития информационного общества в Российской Федерации на 2017 - 2030 годы". URL: https://base.garant.ru/71670570 (дата обращения: 18.02.24).

1. Анализ причин возникновения плавающих уязвимостей и рекомендации по их недопущению

Одним из условий успешной нейтрализации угроз, связанных с плавающими уязвимостями, является анализ и устранение причин их появления. В силу своей открытой природы, операционная система Android становится более уязвимой к атакам и вредоносным воздействиям по сравнению с iOS. Это обусловлено возможностью изменения и модификации кода приложений, чем и пользуются злоумышленники. Кроме того, следует учитывать тот факт, что значительная часть пользователей Android не обновляет свои устройства и остается на старых версиях прошивок, что увеличивает вероятность эксплуатации известных уязвимостей и ослабляет защиту от новых угроз [9, 10]. Не нарушая общности рассуждений, проведем анализ на примерах мобильных приложений для систем типа Android, использующих современный язык программирования Kotlin.

Наиболее часто встречаются следующие ситуации, при которых допускаются ошибки в процессе разработки приложения и написания кода [11–13]:

А. Небезопасные настройки сетевой конфигурации;

Б. Небезопасные настройки WebView – элемент платформы Android, предназначенный для внедрения веб-страниц в приложения;

В. Неточности при работе в многопоточной среде;

Г. Некорректное использование сторонних библиотек;

Д. Ошибки при тестировании.

Рассмотрим эти ситуации более подробно.

А. Небезопасные настройки сетевой конфигурации. Файл AndroidManifest.xml из Android-приложений содержит описания основных свойств, компонентов приложения и различных опций. В разделе application этого файла в процессе эксплуатации возможно несанкционированное включение опции usesCleartextTraffic = "true", что позволяет использовать незашифрованный HTTP-трафик, представляющий угрозу безопасности на уровне всего приложения. Кроме того, в проекте возможно использование декларативного файла Network Security Config конфигурации формата XML, предоставляющего возможность изменять настройки параметров сетевой безопасности приложения. Данный файл позволяет осуществить настройку параметров для конкретных доменов и приложений, тем самым предоставляя возможность разрешения незашифрованного трафика, а также настроить доверие к пользовательским сертификатам. При случайной установке пользователем вредоносного сертификата через фишинг, открывается возможность осуществления MITM-атаки через включение доверия к пользовательским сертификатам в настройках Android.

Б. Небезопасные настройки WebView. К потенциально опасным настройкам и практикам относятся:

1. Включение поддержки JavaScript в WebView, так как это может быть использовано для выполнения вредоносного кода на стороне клиента (XSS-атаки). Включение должно производиться только в случае абсолютной необходимости в условиях мониторинга за вводом данных пользователя.

2. Отключенный режим миксованного содержимого Mixed Content Mode, отсутствие поддержки HTTPS может привести к возможности атаки через небезопасные ресурсы.

3. Отключение режима блокировки всплывающих окон, возможность их открытия через функцию JavaScript может стать потенциальным источником вредоносных действий.

4. Включенный доступ к файлам из WebView может создать уязвимость, особенно если содержимое подвержено изменению внешними источниками.

5. Отключенные ограничения на использование сторонних ресурсов, если ограничение не введено, это может привести к атакам через ненадежные источники.

В. Ошибки при работе в многопоточной среде. Уязвимости могут возникнуть из-за неопределенного поведения, когда несколько потоков конкурируют за доступ и изменение общих данных, не синхронизируя свои действия. Плавающие уязвимости в таких случаях могут проявиться в виде случайных и непредсказуемых значений различных переменных при каждом запуске программы, в зависимости от того, какие потоки успели выполнить свои действия. Использование механизмов синхронизации, таких как Mutex (или аналогичных механизмов синхронизации), позволяет избежать гонок данных и обеспечивает корректное изменение общих ресурсов в многопоточной среде [14, 15].

Г. Использование сторонних библиотек. Использование сторонних библиотек и фреймворков, которые могут обновляться независимо от используемого приложения, представляет дополнительный источник потенциальных плавающих уязвимостей, особенно если они используются в разных версиях приложения.

Д. Ошибки при тестировании. Все аспекты охватить чрезвычайно сложно. Например, в программном коде могут существовать функции, о которых тестировщик не осведомлен, или могут иметься источники искажений. В приложении могут появляться не задокументированные переменные и так далее.

Проведенный анализ показывает, что основными предпосылками реализации угроз являются плавающие уязвимости, проявление которых в специфичных сценариях делает их неочевидными и сложными для выявления, как на этапе отладки мобильных приложений, так и после передачи пользователю. Они служат предпосылками для реализации злоумышленниками деструктивных воздействий, поэтому их обнаружение становится неотъемлемой частью процесса разработки программного продукта. Однако, традиционные методы обнаружения ошибок в программном обеспечении, такие как регулярные аудиты, тестирование и анализ кода, не дают гарантии отсутствия плавающих уязвимостей, так как на момент проверки они могут отсутствовать в используемом приложении.

Следует отметить, что обнаружение плавающих уязвимостей означает не только реакцию на уже возникшие проблемы, но и предполагает внедрение проактивных мер, направленных на предотвращение их появления в будущем.

В контексте вышесказанного, тестирование и анализ кода становятся эффективными инструментами для поддержания высокого уровня безопасности в процессе разработки, но не гарантируют отсутствия плавающих уязвимостей при эксплуатации приложения.

2. Основные способы выявления и купирования плавающих уязвимостей

Известные в настоящее время способы обнаружения всех видов уязвимостей в основном относятся к периоду разработки и финального тестирования и, как правило, не распространяются на этап эксплуатации. Наиболее распространенными из них являются следующие:

А. Регулярное тестирование, как ручное, так и автоматизированное, на различных устройствах и под разными версиями операционной системы. Такие тесты выявляют существенную долю проблем, которые могут возникнуть при различных условиях использования.

Б. Мониторинг и обратная связь с пользователями. Опыт пользователей охватывает применение большого количества разнообразных устройств в различных ситуациях и

может служить основой для формирования базы данных о возможных проблемах, обусловленных плавающими уязвимостями.

В. Анализ сторонних библиотек на предмет изменений в их коде. Анализ кода библиотек осуществляется на предмет возможных конфликтов или уязвимостей, которые могут повлиять на безопасность приложения.

Г. Непрерывное информирование разработчиков в области новых технологий, изменениях в операционных системах и подходах к обеспечению безопасности при ненулевой вероятности появления плавающих уязвимостей.

Д. Подключение к процессу тестирования и отладки независимых экспертов, в том числе из смежных областей.

Е. Использование логов проекта программы с целью воспроизводства сломанного программного обеспечения.

Ж. Отыскивание в базах багов похожих проблем.

З. Применение традиционных методов научных исследований. Системный подход способствует проверкам слабосвязанных фрагментов и определению признаков присутствия плавающих уязвимостей, а декомпозиция позволяет локализовать баг.

И. Проведение стресс-тестов, позволяющих выявить перегруженность программы как причину образования багов.

К. Организация специальных «ловушек» и индикаторов.

Таким образом, в настоящее время способов обнаружения и устранения плавающих уязвимостей немного, а имеющиеся не охватывают период эксплуатации, что осложняет процесс разработки и увеличивает его сроки. Одной из проблем повышения безопасности мобильных приложений является то, что названные выше уязвимости, а также способы их купирования, не включены в базу угроз ФСТЭК России[2]. Следовательно, требуется разработка новых подходов, которые бы позволили в автоматическом режиме обнаруживать плавающие уязвимости и своевременно их устранять.

3. Разработка метода выявления плавающих уязвимостей

Основная идея выявления плавающих уязвимостей заключается в регулярной процедуре сравнения текущего состояния исследуемого программного продукта в некоторые фиксированные моменты времени с его состоянием на момент начала применения пользователем. Состояние мобильного приложения в момент передачи пользователю считается эталонным, то есть заведомо исправным и не имеющим уязвимостей.

Обозначим через $S_N(X,t)$, $X(t)=\{x_i(t)\}$, $i=[1,I]$ состояние мобильного приложения при нормальной эксплуатации, то есть в случае отсутствия любых уязвимостей, где $X(t)$ – набор характеристик объекта исследования. Считаем, что в момент начала эксплуатации приложения t_0, то есть после процедуры тестирования программного обеспечения, плавающие уязвимости отсутствуют, состояние объекта – $S_N(X,t_0)$.

В процессе эксплуатации возможен переход объекта в нештатное состояние, характеризующее появление плавающей уязвимости $S_N(X,t)\to S_{\mathrm{Uk}}(X,t,\tau)$, $k=[1,K]$, где K – набор всех возможных для данного объекта плавающих уязвимостей, τ – время, в течение которого уязвимость присутствует в мобильном приложении. Из определения плавающей уязвимости следует, что по прошествии некоторого времени следует обратный переход $S_{\mathrm{Uk}}(X,t,\tau)\to S_N(X,t)$ и объект возвращается в штатную эксплуатацию. Опыт эксплуатации мобильных приложений свидетельствует, что в произвольный момент

[2]Банк данных угроз безопасности информации. URL: https://bdu.fstec.ru/threat (дата обращения:18.02.2024).

времени в приложении либо отсутствуют уязвимость, либо может присутствовать только одна уязвимость, а вероятность появления одновременно двух и более практически равна нулю. При этом наиболее вероятен процесс нормальной эксплуатации $P_N(t) >> P_{\mathrm{Uk}}(t)$, где $P_N(t)$ и $P_{\mathrm{Uk}}(t)$ – вероятности отсутствия уязвимостей и появления одной уязвимости соответственно.

Обнаружение плавающих уязвимостей осуществляется на основе анализа в различные дискретные моменты времени состояний мобильного приложения, которое характеризуется конечным количеством выполняемого кода, модулей, сервисов, вложенных ресурсов и других составляющих, характеризующих текущие свойства исследуемого объекта. Анализу подлежат связи и алгоритмы взаимодействия между названными элементами. Очевидно, состояние объекта является функцией времени, при этом момент появления и исчезновения плавающих уязвимостей априори неизвестен. Дискретный анализ осуществляется через некоторые, не обязательно равные, промежутки времени Δt. Необходимым условием обнаружения является наличие уязвимости в момент анализа $t_{\mathrm{ai}} \subset \tau$, $t_{ai} = t_0 + n\Delta t$, $n = [1, N]$. Предположение о возможном присутствии плавающей уязвимости определяется фактом отличия текущего состояния объекта исследования от штатного $S_N(X, t_0)$.

После обнаружения и идентификации устранение плавающих уязвимостей осуществляется традиционными способами, наиболее распространенными из которых являются:

А. Статический анализ кода, проведение тщательного анализа исходного кода приложения для выявления потенциальных уязвимостей на этапе компиляции. Этот метод включает в себя сканирование кода на предмет известных уязвимостей и паттернов без его фактического выполнения.

Б. Динамический анализ, запуск приложения в контролируемой среде с целью выявления уязвимостей в реальном времени. Этот метод включает в себя мониторинг взаимодействия приложения с внешними ресурсами и анализ выполнения кода на предмет аномалий.

В. Анализ потоков данных, исследование потоков данных внутри приложения для выявления уязвимостей, связанных с обработкой ввода-вывода и работы с данными.

Г. Тестирование на проникновение, симуляция атак с целью проверки устойчивости приложения к различным видам воздействий. Этот метод включает в себя попытки взлома, изменения данных в транзите и другие атаки на систему.

Д. Мониторинг событий безопасности, установка систем мониторинга для регистрации и анализа событий безопасности в реальном времени. Это позволяет выявлять аномалии и нештатные ситуации, связанные с возможными угрозами.

Программа выявления встраивается в приложение и после настройки периодичности тестирования (оценки) текущего состояния реализуется в автоматическом режиме.

Алгоритм выявления плавающих уязвимостей представлен на рис. 1.

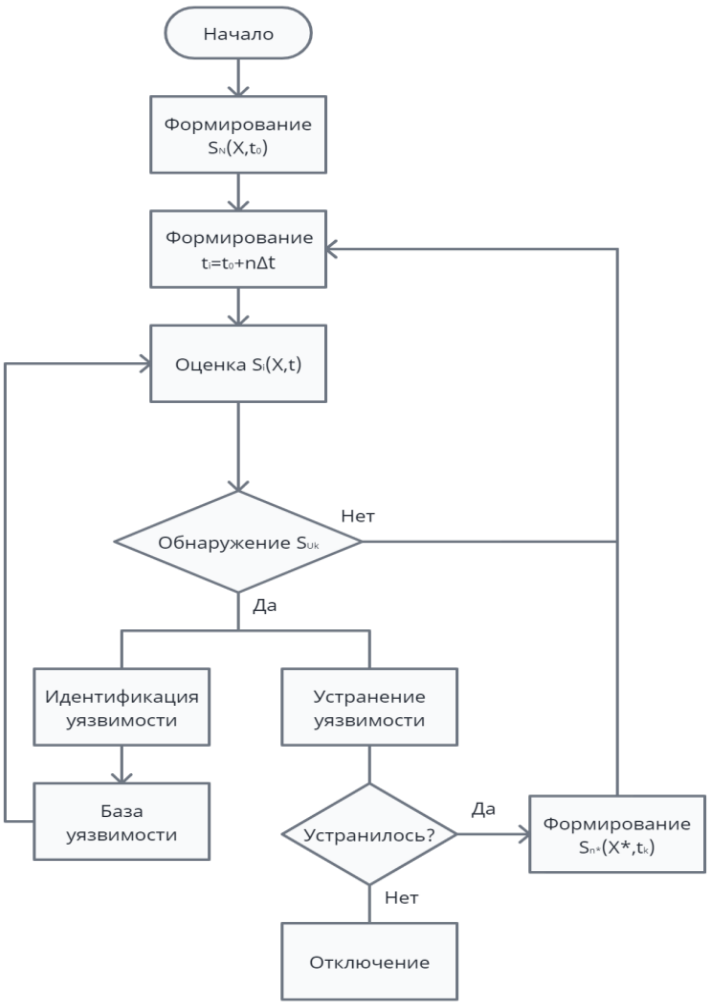

Рис. 1. Алгоритм выявления плавающих уязвимостей

Для выявления плавающих уязвимостей необходимо зафиксировать состояние мобильного приложения в момент передачи его пользователю, то есть на начало эксплуатации, обозначенное как $S_N(X,t_0)$, $X(t_0)=\{x_i(t_0)\}$, $i=[1,I]$ – состояние мобильного приложения при нормальной эксплуатации. Анализ осуществляется в дискретные моменты через промежутки времени (не обязательно равные) Δt, которые определяются как $t_{ai}=t_0+n\Delta t$, $n=[1,N]$. Обозначим функцию анализа причин появления уязвимости $F[G(k)]$, где $G(k)$ – функция провокации уязвимости, характеризующая предпосылки к ее появлению, k – конкретная уязвимость. Проводится детальный анализ текущего состояния приложения, представленного функцией $S_N(X,t_{ai})$, описывающей состояние приложения в момент времени t_{ai}. Это состояние сравнивается с эталонным состоянием. При выявлении различий предполагается появление плавающей уязвимости. В случае обнаружения плавающей уязвимости происходит переход в состояние $F[G(k)]\cdot S_N(X,t_0)\rightarrow S_{Uk}(X_h,t_k)$, где X_h – свойства при наличии k-ой уязвимости, t_k – время ее обнаружения. Переход осуществляется в соответствии с логикой функции провокации, определенной $G(k)$, и происходит с учетом актуальных характеристик приложения X_h и времени обнаружения t_k. Происходит остановка работы приложения и идентификация уязвимости, проверка наличия ее в базе угроз и, при необходимости, добавление в базу. Одновременно формируется сигнал пользователю о потенциальной

опасности. Устранение уязвимости проводится путем изменения исходного кода приложения, приводя к новому состоянию $S_N(X, t_{ai})$. После успешного устранения уязвимости, фиксируется новое штатное состояние $S_N^*(X^*, \mathrm{t}_k^*)$, $X^*(t_k^*) = \{x_i(t_k^*)\}$, $i = [1, I]$, где X^* – свойства после устранения уязвимости, t_k^* – время устранения k-ой уязвимости. Считаем, что время устранения мало, тогда $t_k^* \approx t_k$.

В случае невозможности устранения уязвимости, приложение или само устройство подлежит отключению для предотвращения дополнительных рисков безопасности.

Алгоритм позволяет синтезировать программу, представляющую собой библиотеку, специально разработанную для внедрения в мобильные приложения под управлением Android. Эта библиотека осуществляет выявление и автоматическое устранение плавающих уязвимостей, предоставляя разработчикам эффективный инструментарий безопасности для повышения надежности и защищенности приложений.

Заключение

В данной статье рассмотрено понятие плавающих уязвимостей в мобильных приложениях, выявлена проблема их негативного воздействия на обеспечение информационной безопасности. Проведен анализ основных причин появления плавающих уязвимостей и наиболее действенных методов их устранения. Опираясь на примеры выявлена важность обнаружения и предотвращения плавающих уязвимостей в разработке мобильных приложений.

Сделан вывод о том, что осознание рисков и активное внедрение методов предотвращения гейзенбагов содействуют созданию более безопасных и надежных мобильных приложений, соответствующих высоким стандартам информационной безопасности.

Разработан оригинальный метода обнаружения плавающих уязвимостей в мобильных приложениях, основанный на дискретном анализе их состояния в различные моменты времени, и предложен алгоритм его реализации. Разработанный метод служит основой для синтеза специальной подпрограммы и внедрения ее в мобильное приложение. Следует отметить, что встроенная в приложение подпрограмма обнаружения плавающих уязвимостей, синтезированная на основе представленного алгоритма, не требует значительных объемов памяти, временных ресурсов и вычислительных мощностей и, следовательно, не оказывает существенного влияния на работу устройств, использующих мобильные приложения.

СПИСОК ЛИТЕРАТУРЫ:

1. Шестакова К. Объем программного кода в компаниях за 10 лет вырос в 100 раз – Инфостарт – Октябрь 2020. URL: https://infostart.ru/journal/news/tekhnologii/obem-programmnogo-koda-v-kompaniyakh-za-10-let-vyros-v-100-raz_1310447/ (дата обращения: 18.02.2024).
2. Вострецова Е.В. Основы информационной безопасности: учебное пособие для студентов вузов. Екатеринбург: Изд-во Урал. ун-та. 2019. – 204 с. URL: https://elar.urfu.ru/bitstream/10995/73899/3/978-5-7996-2677-8_2019.pdf (дата обращения: 18.02.2024).
3. Каверин Д. Треть критической инфраструктуры в России подвергалась хакерским атакам – Газета.ru – Декабрь 2023. URL: https://www.gazeta.ru/tech/news/2023/12/25/22000549.shtml (дата обращения: 18.02.2024).
4. Бельтов А.Г. и др. Атаки на мобильные телефоны, использующие механизм автоматической настройки. Безопасность информационных технологий, [S.l.], т. 19, № 2S, с. 22–25, 2012. ISSN 2074-7136. – EDN: PVRXDF.
5. Шептунов А.А. и др. Атаки на мобильные устройства с использованием технологии WI-FI. Безопасность информационных технологий, [S.l.], т. 19, № 2S, с. 26–31, 2012. ISSN 2074-7136. – EDN PVRXDP.

6. Стариковский А.В. и др. Атаки на мобильные телефоны, использующие уязвимости технологии NFC. Безопасность информационных технологий, [S.l.], т. 19, № 2S, с. 60–64, 2012. ISSN 2074-7136. – EDN: OGNNDK.
7. Goller G., Sigl G. (2015). Side Channel Attacks on Smartphones and Embedded Devices Using Standard Radio Equipment. In: Mangard, S., Poschmann, A. (eds) Constructive Side-Channel Analysis and Secure Design. COSADE 2015. Lecture Notes in Computer Science, v. 9064. Springer, Cham. DOI: https://doi.org/10.1007/978-3-319-21476-4_17.
8. Kilpatrick W. What Is Heisenbug? – Cellularnews – December 2023. URL: https://cellularnews.com/definitions/what-is-heisenbug/ (дата обращения: 18.02.2024).
9. Баркалов Ю.М., Нестеров А.Д. Особенности обеспечения информационной безопасности в мобильных устройствах под управлением операционной системы Android. Вестник Дагестанского государственного технического университета. Технические науки. 2019; 46 (2): 71–80. DOI: https://doi.org/10.21822/2073-6185-2019-46-2-71-80. – EDN: QXKGGO.
10. Зобнин Е.Е. Android глазами хакера. СПб.: БХВ-Петербург. 2021. – 274 с. URL: https://coderbooks.ru/books/hacking-and-security/android-glazami-hakera/ (дата обращения: 18.02.2024).
11. Дейтел П., Дейтел Х., Уолд А. Android для разработчиков. 3-е изд. СПб.: Питер Пресс. 2016. – 512 с. URL: https://book-pc.ru/programmirovanie/951-android-dlja-razrabotchikov-3-e-izdanie-code.html (дата обращения: 18.02.2024).
12. Гавриленко А.И., Корчагина Е.В. Анализ обеспечения информационной безопасности мобильных устройств. Актуальные проблемы деятельности подразделений УИС: Сборник материалов Всероссийской научно-практической конференции, Воронеж, 2022. Издательско-полиграфический центр «Научная книга». 2022, с. 141–143. – EDN: ZNCELF.
13. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер. 2019. – 1120 с. URL: https://www.litres.ru/book/endru-tanenbaum/sovremennye-operacionnye-sistemy-42227045/ (дата обращения: 18.02.2024).
14. Göetz B., Bloch J. Java Concurrency In Practice. Addison Wesley Professional. 2010, p. 425. URL: https://library-it.com/wp-content/uploads/2020/06/java_concurrency_in_practice.pdf (дата обращения: 18.02.2024).
15. Билл Ф., Крис С., Кристин М., Брайан Г. Android. Программирование для профессионалов. 4-е изд. СПб.: Питер. 2021. – 704 с. URL: https://www.labirint.ru/books/775044/ (дата обращения: 18.02.2024).

REFERENCES:

[1] Shestakova K. The volume of software code in companies has grown 100 times over 10 years - Infostart - October 2020. URL: https://infostart.ru/journal/news/tekhnologii/obem-programmnogo-koda-v-kompaniyakh-za-10-let-vyros-v-100-raz_1310447/ (accessed: 18.02.2024) (in Russian).
[2] Vostretsova E.V. Fundamentals of information security: a textbook for university students. Yekaterinburg: Publishing house Ural.un-ta. 2019. – 204 p. URL: https://elar.urfu.ru/bitstream/10995/73899/3/978-5-7996-2677-8_2019.pdf (accessed: 18.02.2024) (in Russian).
[3] Kaverin D. A third of critical infrastructure in Russia was subject to hacker attacks - Gazeta.ru - December 2023. URL: https://www.gazeta.ru/tech/news/2023/12/25/22000549.shtml (accessed: 18.02.2024) (in Russian).
[4] Beltov A.G. et al. Attacks on mobile phones that use the automatic configuration mechanism. Information Technology Security, [S.l.], v. 19, no. 2S, p. 22–25, 2012. ISSN 2074-7136 (in Russian). – EDN: PVRXDF.
[5] Sheptunov A.A. et al. Attacks on mobile devices using WI-FI technology. Information Technology Security, [S.l.], v. 19, no. 2S, p. 26–31, 2012. ISSN 2074-7136 (in Russian). – EDN: PVRXDP.
[6] Starikovsky A.V. et al. Attacks on mobile phones that use the vulnerability of NFC technology. Information Technology Security, [S.l.], v. 19, no. 2S, p. 60–64, 2012. ISSN 2074-7136 (in Russian). – EDN: OGNNDK.
[7] Goller G., Sigl G. (2015). Side Channel Attacks on Smartphones and Embedded Devices Using Standard Radio Equipment. In: Mangard, S., Poschmann, A. (eds) Constructive Side-Channel Analysis and Secure Design. COSADE 2015. Lecture Notes in Computer Science, v. 9064. Springer, Cham. DOI: https://doi.org/10.1007/978-3-319-21476-4_17.
[8] Kilpatrick W. What Is Heisenbug? – Cellularnews – December 2023. URL: https://cellularnews.com/definitions/what-is-heisenbug/ (accessed: 18.02.2024).
[9] Barkalov Yu.M., Nesterov A.D. Peculiarities of information security in mobile devices running the Android operating system. Herald of Daghestan State Technical University. Technical Sciences. 2019; 46 (2): 71-80. DOI: https://doi.org/10.21822/2073-6185-2019-46-2-71-80 (in Russian). – EDN: QXKGGO.
[10] Zobnin E.E. Android through the eyes of a hacker. St. Petersburg: BHV-Petersburg. 2021. – 274 p. URL: https://coderbooks.ru/books/hacking-and-security/android-glazami-hakera/ (accessed: 18.02.2024) (in Russian).

[11] Deytel P., Deytel H., Wall A. Android for developers. 3rd ed. St. Petersburg: Piter Press. 2016. – 512 p. URL: https://book-pc.ru/programmirovanie/951-android-dlja-razrabotchikov-3-e-izdanie-code.html (accessed: 18.02.2024) (in Russian).
[12] Gavrilenko A.I., Korchagina E.V. Analysis of ensuring the information security of mobile devices. Actual problems of the activities of the penitentiary system units: Collection of materials of the All-Russian Scientific and Practical Conference, Voronezh, 2022. Publishing and Printing Center "Scientific Book". 2022, p. 141–143 (in Russian). – EDN: ZNCELF.
[13] Tanenbaum E., Bos H. Modern operating systems. 4th ed. St. Petersburg: Peter. 2019. – 1120 p. URL: https://www.litres.ru/book/endru-tanenbaum/sovremennye-operacionnye-sistemy-42227045/ (accessed: 18.02.2024) (in Russian).
[14] Göetz B., Bloch J. Java Concurrency In Practice. Addison Wesley Professional. 2010, p. 425. URL: https://library-it.com/wp-content/uploads/2020/06/java_concurrency_in_practice.pdf (accessed: 18.02.2024).
[15] Bill F., Chris S., Christine M., Brian M. Android. Programming for professionals. 4th ed. St. Petersburg: Piter, 2021. – 704 p. URL: https://www.labirint.ru/books/775044/ (accessed: 18.02.2024) (in Russian).

Поступила в редакцию – 27 февраля 2024 г. Окончательный вариант – 20 апреля 2024 г.
Received – February 27, 2024. The final version – April 20, 2024.