DOI: 10.15514/ISPRAS-2026-38(4)-24

Методы формирования структур запросов к серверным точкам мобильных приложений для определения поверхности атаки

[1] *Ш.Ф. Курмангалеев, ORCID: 0000-0002-0558-2850 <kursh@ispras.ru>*
[1] *А.С. Юрьев, ORCID: 0009-0003-0369-0422 <forhhpurpose@yandex.ru>*
[2] *Н.А. Дацук, ORCID: 0009-0006-5403-0960 <dazuk07@yandex.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, Россия, 109004, г. Москва, ул. А. Солженицына, д. 25.*

[2] *Московский государственный технический университет им. Н.Э. Баумана, Россия, 105005, г. Москва, 2-я Бауманская ул., д. 5, с. 1.*

Аннотация. В статье рассматривается актуальная проблема определения поверхности атаки для мобильных приложений (МП) в условиях роста числа атак через API-интерфейсы. Отмечено отсутствие унифицированных подходов, учитывающих роль МП как точки входа в корпоративную инфраструктуру. Предложен комплексный подход к поиску функций входа без анализа серверной части, включающий методы поиска серверных интерфейсов и формирования шаблонов HTTP-запросов для динамического анализа (DAST) или фаззинга. Выдвигается гипотеза о том, что рассматриваемый подход позволит повысить точность определения поверхности атаки и обеспечить превентивное выявление дефектов на ранних стадиях разработки. Практическая значимость заключается в возможности увеличения покрытия проверок и снижения рисков эксплуатации через серверные интерфейсы. Разработанные методы применимы к платформам Android (Java/Kotlin) и iOS (Swift/Objective-C). В ходе проведённого исследования разработан и апробирован подход к анализу исходного кода мобильных приложений, направленный на выявление механизмов взаимодействия с серверными точками. В отличие от традиционных методов анализа защищённости, основанных преимущественно на динамическом тестировании по принципу «чёрного ящика» или ручном реверс-инжиниринге, предложенный подход реализует статический анализ с семантическим извлечением точек взаимодействия. Существующие инструменты автоматизированного анализа ориентированы в основном на выявление типовых дефектов ИБ в клиентском коде: небезопасное хранение данных, некорректная валидация сертификатов, избыточные разрешения. Однако, задача систематического картирования серверных API на основе анализа исходного кода с целью формирования поверхности атаки остаётся недостаточно проработанной в научной литературе и практике.

Ключевые слова: безопасность мобильных приложений; API; Android; iOS; анализ кода; WEB.

Для цитирования: Курмангалеев Ш.Ф., Юрьев А.С., Дацук Н.А. Методы формирования структур запросов к серверным точкам мобильных приложений для определения поверхности атаки. Труды ИСП РАН, том 38, вып. 4, часть 2, 2026 г., стр. 161–176. DOI: 10.15514/ISPRAS–2026–38(4)–24.

Methods for Constructing Request Structures to Mobile Application Server Endpoints for Attack Surface Identification

[1] *S.F. Kurmangaleev, ORCID: 0000-0002-0558-2850 <kursh@ispras.ru>*
[1] *A.S. Yurev, ORCID: 0009-0003-0369-0422 <forhhpurpose@yandex.ru>*
[2] *N.A. Datsuk, ORCID: 0009-0006-5403-0960 <dazuk07@yandex.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow,109004, Russia.*

[2] *Bauman Moscow State Technical University, bldg. 5, p. 1, 2nd Baumanskaya st., Moscow, 105005, Russia.*

Abstract. This paper addresses the problem of identifying the attack surface of mobile applications in the context of the increasing number of attacks exploiting API interfaces. The absence of unified approaches that consider mobile applications as entry points into corporate infrastructure is highlighted. A comprehensive approach to identifying vulnerable functionalities without analyzing the server-side implementation is proposed. The approach includes methods for discovering server interfaces and constructing HTTP request templates for dynamic application security testing (DAST) or fuzzing. The hypothesis is formulated that the proposed approach can improve the accuracy of attack surface identification and enable proactive detection of information security vulnerabilities at early stages of the software development lifecycle. The practical significance of the study lies in increasing security testing coverage and reducing the risks of exploitation through server-side interfaces. The proposed methods are applicable to major mobile platforms, including Android (Java/Kotlin) and iOS (Swift/Objective-C). Unlike traditional security analysis methods based primarily on dynamic black box testing or manual reverse engineering, the proposed approach implements static analysis with semantic extraction of interaction points. The existing automated analysis tools are mainly focused on identifying typical vulnerabilities in client code: insecure data storage, incorrect certificate validation, and redundant permissions. However, the task of systematically mapping server-side APIs based on source code analysis in order to form an attack surface remains insufficiently developed in scientific literature and practice.

Keywords: mobile application security; API; Android; iOS; code analysis; web security.

For citation: Kurmangaleev S.F., Yurev A.S., Datsuk N.A. Methods for Constructing Request Structures to Mobile Application Server Endpoints for Attack Surface Identification. Trudy ISP RAN/Proc. ISP RAS, vol. 38, issue 4, part 2, 2026. pp. 161-176 (in Russian). DOI: 10.15514/ISPRAS-2026-38(4)-24.

1. Введение

В эпоху интенсивной цифровизации и экспоненциального роста количества мобильных приложений, разрабатываемых различными организациями, возникает задача качественного обеспечения информационной безопасности (ИБ). Ключевым индикатором устойчивости информационных систем (ИС) к внешним угрозам становится понятие поверхности атаки: совокупность потенциальных дефектов ИБ, которые могут быть использованы злоумышленниками для компрометации инфраструктуры организации. Согласно ГОСТ 56939-2024 [1], поверхность атаки представляет собой множество подпрограмм (функций, модулей) программного обеспечения (ПО), обрабатывающих данные, поступающие через интерфейсы, которые прямо или косвенно подвержены риску атаки. При этом для мобильных приложений (МП) важное место занимает интерфейс, поскольку подавляющее большинство современных МП взаимодействует с сервером посредством WEB-интерфейсов по протоколу HTTP (Hyper Text Transfer Protocol).

Согласно различным исследованиям, растет доля атак на организации через мобильные приложения [2-3], где основной вектор атак приходится на API-интерфейсы (Application Programming Interface). Сегодня отсутствуют унифицированные подходы к определению поверхности атаки именно для мобильных приложений в отношении ИС. Мобильные

приложения часто интегрированы в сложную цифровую экосистему, включающую сетевую инфраструктуру, программное и аппаратное обеспечение, облачные сервисы и данные. Несмотря на наличие разнообразных методов анализа защищённости, существующие решения зачастую рассматривают МП изолированным объектом, не учитывая их роль как точки входа в сеть организаций.

В современной практике динамического анализа и фаззинга WEB-приложений всё чаще используются шаблоны (динамические правила – playbooks) для специализированных инструментов. Их ключевое преимущество заключается в систематизации и унификации тестовых сценариев: заранее определённые шаблоны позволяют быстро конфигурировать проверки на широкий спектр дефектов информационной безопасности (ИБ) (SQL-инъекции, XSS, переполнения буфера, обход аутентификации и др.) без необходимости каждый раз разрабатывать тесты «с нуля». Такие шаблоны, как правило, включают:

- параметризованные наборы входных данных для провокации ошибок (payloads);
- условия сопоставления (matchers) для автоматического детектирования аномалий в ответах;
- правила извлечения признаков ошибок (extractors);
- настройки выполнения (ограничения скорости, повторные попытки, таймауты).

Цель настоящей работы – провести исследование современных методов и на их основе разработать комплексный подход к определению поверхности атаки для МП с помощью алгоритма формирования структур HTTP-запросов для практик динамического анализа (DAST – Dynamic Application Security Testing) и фаззинга с покрытием, близким к полному. Для достижения поставленной цели необходимо решить следующие задачи:

- определить поверхность атаки для мобильных приложений без анализа серверной части;
- оценить долю поверхности атаки, формируемой для МП;
- разработать методы поиска и обнаружения серверных интерфейсов (точек взаимодействия) в коде мобильных приложений;
- разработать практические методы формирования шаблонов на основе структур HTTP-запросов для более точного фаззинг-тестирования и динамического анализа (DAST).

Объектами исследований выступают МП операционных систем Android (разрабатываемые на языках Java/Kotlin) и iOS (разрабатываемые на языках Swift/Objective-C).

В работе выдвигается гипотеза о том, что предложенные методы позволят:

- существенно повысить точность определения поверхности атаки;
- интегрировать разработанные подходы в жизненный цикл разработки ПО (в том числе в процессы CI/CD);
- обеспечить превентивное выявление дефектов ИБ на ранних стадиях разработки;
- разработать системный подход к определению поверхности атаки для МП, которое учитывается, как элемент общей информационной инфраструктуры организации.

Текущие практики анализа защищенности в организациях могут быть дополнены разработанными механизмами поиска точек взаимодействия с сервером, с интеграцией в CI/CD-процессы для превентивного выявления дефектов ИБ на ранних стадиях разработки (методология "сдвиг влево" – shift-left). Это может существенно повысить точность анализа ИБ в контексте общей ИТ-инфраструктуры организации. В отличие от существующих решений, рассматривающих мобильные приложения изолированно, предложенный подход базируется на алгоритме формирования структур HTTP-запросов, что позволяет выявлять

серверные интерфейсы непосредственно в коде приложений (для платформ Android и iOS) и генерировать параметризованные шаблоны для динамического анализа (DAST) и фаззинга. Предполагается, что практическая значимость разработанного подхода позволит адаптировать методы динамического анализа, что в свою очередь увеличит покрытие проверками и повысит уровень защищённости.

2. Современные подходы к определению поверхности атаки мобильных приложений

Относительно задач текущей статьи был проведен анализ литературы. На конференции USENIX Security Symposium 2021 была представлена работа [4], посвящённая проблеме анализа поверхности атаки современных мобильных приложений. В исследовании подчёркивается, что выявление статических URL-адресов в коде приложения не обеспечивает полноценного понимания функциональности API. Для эффективной эксплуатации дефектов ИБ необходимо определить набор параметров, принимаемых конкретной серверной точкой и установить связь между UI-элементами (поля ввода, кнопки) и инициируемыми ими сетевыми запросами. Авторы формализуют поверхность атаки как последовательную цепочку преобразований:

ввод пользователя (UI) → обработка данных → сетевой запрос (API).

В качестве решения предложен автоматизированный метод, позволяющий устанавливать соответствие между элементами пользовательского интерфейса и конкретными сетевыми запросами, выявлять семантику API-взаимодействий (например, определять, что нажатие определённой кнопки инициирует POST-запрос с фиксированным набором параметров) и обнаруживать сетевые точки, требующие сложной логики взаимодействия (многошаговые сценарии, такие как регистрация или платёжные операции). Ключевое преимущество подхода заключается в автоматизации процесса идентификации уязвимых точек, которые остаются необнаруженными при использовании традиционных методов фаззинга, ориентированных на статический анализ кода или перебор параметров.

В работе [5], представленной на IEEE Symposium on Security and Privacy (S&P) 2020, предложен подход к выявлению скрытой поверхности атаки в мобильных приложениях. Авторы констатируют существенный недостаток традиционных методов анализа безопасности: они систематически игнорируют скрытый функционал, включающий как тестовые механизмы, оставленные разработчиками в сборках промышленных сред, так и недокументированные бэкдоры и пути выполнения кода, активируемые специфическими входными данными. В рамках исследования разработан инструмент InputScope, реализующий комплексный анализ приложения посредством: исследования входных данных, принимаемых приложением, статического анализа исходного кода, семантического анализа строковых констант и литералов. Эта работа демонстрирует необходимость переосмысления традиционных подходов к оценке защищённости мобильных приложений с учётом наличия неявных, но потенциально опасных механизмов взаимодействия, активируемых специфическими входными воздействиями.

В статье [6], представленной на USENIX Security Symposium 2022, исследуется проблема безопасности мобильных приложений, использующих парадигму Backend-as-a-Service (BaaS) (на примере приложений Alibaba Cloud, AWS Amplify, Firebase). Авторы демонстрируют, что в условиях широкого распространения BaaS-решений логика доступа к серверным базам данных зачастую инкапсулируется непосредственно в клиентское приложение, при этом конфигурационные файлы и SDK выступают в роли криптографических ключей к облачной инфраструктуре. Для выявления дефектов ИБ разработан инструмент Huihong, реализующий анализ, основанный на помеченных данных (taint-анализ), который отслеживает пути распространения токенов доступа и API-ключей облачных провайдеров в коде приложения. Экспериментальные результаты свидетельствуют, что значительная доля мобильных

приложений содержит критические дефекты конфигурации прав доступа, приводящие к несанкционированному чтению и записи данных в облачные базы. Исследование концептуально переосмысливает модель поверхности атаки, показывая, что в контексте BaaS-архитектур вектор угрозы смещается с традиционных серверных атак на эксфильтрацию конфиденциальных данных через анализ клиентской составляющей приложения.

В последние годы заметное развитие получили методы динамического анализа мобильных приложений с фокусом на сетевом взаимодействии. На конференции NDSS 2023 [7] был предложен инструмент NetTrace [8], комбинирующий динамический мониторинг сетевых вызовов с символьной интерпретацией входных данных. Подход позволяет:

- автоматически реконструировать схемы API-запросов;
- выявлять зависимости между последовательностью HTTP-запросов;
- генерировать тестовые сценарии для DAST, учитывающие контекст выполнения (например, авторизацию через протокол OAuth).

Данный подход связывает сетевые запросы с конкретными UI-элементами и состояниями приложения. Это позволяет сократить ложные срабатывания при фаззинге и повысить покрытие тестируемых сценариев.

Альтернативный подход предложен в исследовании, представленном на конференции CCS 2022 [9], где рассматривается метод анализа потоков данных для выявления уязвимых точек взаимодействия с сервером. Авторы фокусируются на:

- отслеживании путей распространения чувствительных данных (токены, персональные данные) от пользовательских интерфейсов;
- выявлении неявных зависимостей между параметрами API (например, когда значение одного поля влияет на структуру другого);
- автоматическом построении графов вызовов для визуализации поверхности атаки.

В 2025 году на конференции Ломоносовские чтения была представлена работа [10], которая описывает подходы к извлечению шаблонов HTTP-запросов из мобильных приложений методом статического анализа. Однако полноценного описания методики, порядка проведения работ и алгоритмов с экспериментальной частью не приводится.

Анализ современных исследований демонстрирует нарастающую сложность проблемы анализа поверхности атаки мобильных приложений: традиционные методы, ориентированные на статический анализ кода или перебор параметров, оказываются недостаточными в условиях динамически генерируемых API-запросов, многошаговых сценариев взаимодействия и распространения BaaS-архитектур. Актуальность разработки новых подходов подтверждается выявленными пробелами: игнорированием скрытого функционала (недокументированных скрытых функций, тестовых механизмов), отсутствием учёта семантики логики, а также недостаточной интеграцией инструментов анализа в процессы CI/CD. Указанные выше решения чаще демонстрируют прогресс в автоматизации корреляции UI-элементов с сетевыми запросами и отслеживании потоков данных, однако остаются нерешёнными задачи полного покрытия «скрытых» сценариев, когда элементы UI не участвуют, включая анализ приложений с унификацией форматов шаблонов для инструментария. Особую значимость приобретает интеграция таких методов в жизненный цикл разработки ПО для превентивного выявления дефектов. Необходимость совершенствования подходов к определению поверхности атаки обусловлена и ростом числа мобильных приложений, а также и усложнением их взаимодействия с серверной инфраструктурой, что делает данную область критически важной для обеспечения информационной безопасности организаций.

3. Описание методов формирования структур запросов к серверным точкам

В рамках настоящей работы применяемые методологические подходы не ориентированы на анализ дефектов ИБ непосредственно клиентской части МП, регламентированных в руководстве OWASP MASTG [11]. Фокус исследования смещён в сторону изучения потенциальных векторов проникновения нарушителя в корпоративную сеть организации через МП. Серверные точки (интерфейсы) определим, как ключевой элемент угрозы, через которую может быть осуществлено несанкционированное воздействие на ИС.

На рис. 1 представлен процесс формирования набора шаблонов тестирования в общем виде.

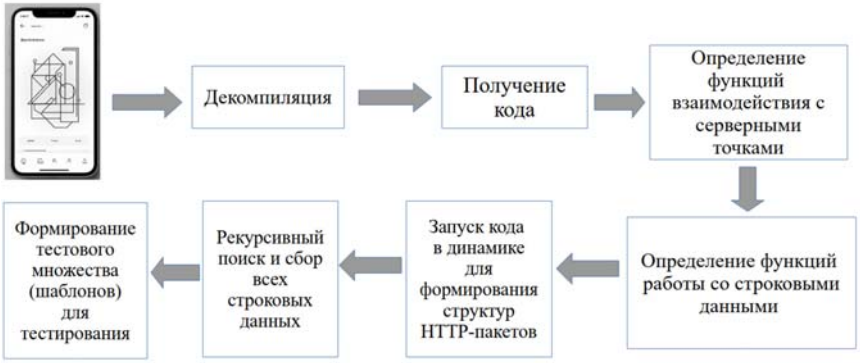

Рис. 1. Процесс формирования набора тестовых векторов HTTP-пакетов.
Fig. 1. The process of generating a set of test vectors for HTTP packets.

Под шаблонами понимаются наборы текстовых правил (обычно в формате YAML или JSON), которые регламентируют процесс динамического анализа или фаззинга для соответствующих инструментов.

С точки зрения объекта исследования порядок проведения работы для получения серверных точек на стороне МП может быть разделен на две категории: когда код мобильного приложения отсутствует (например, был утерян) или когда код мобильного приложения доступен для анализа. На рис. 2 приведен пример получения кода исполняемого файла мобильного приложения [12] с помощью программы JADX [13].

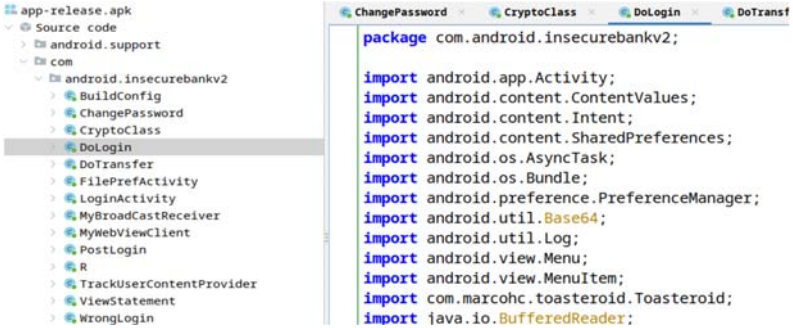

Рис. 2. Пример получения кода мобильного приложения.
Fig. 2. Example of obtaining a mobile app code.

На рис.3 приведен пример подстановки соответствующих значений из кода, полученного методом декомпиляции в тестовый HTTP-запрос.

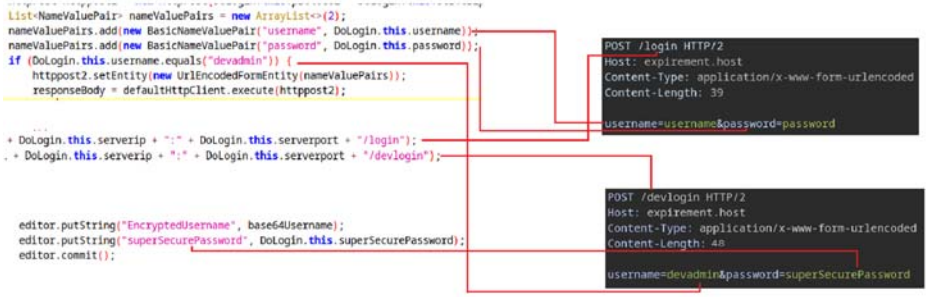

Рис. 3. Формирование HTTP-запроса после применения методов декомпиляции.
Fig. 3. Forming an HTTP request after applying decompilation methods.

Когда код мобильного приложения доступен для анализа, то процесс формирования набора тестовых векторов HTTP-пакетов аналогичен представленному на рис.1, но без этапа декомпиляции. На рис. 4 приведен пример подстановки соответствующих значений в тестовый HTTP-запрос из исходного кода.

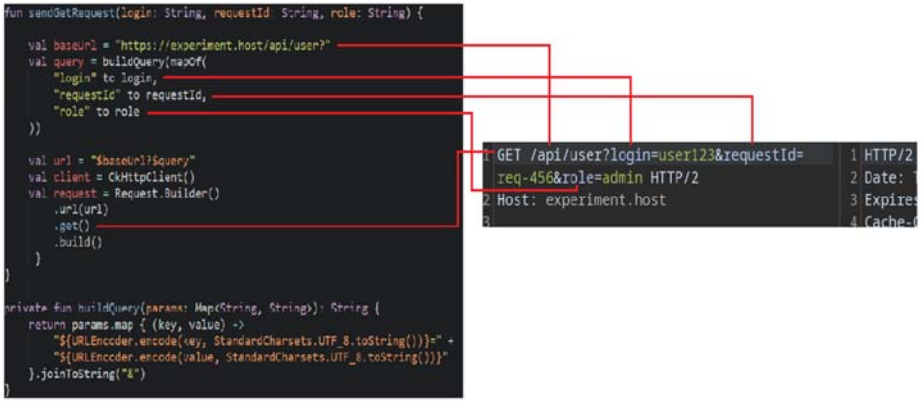

Рис. 4. Формирование запроса с помощью анализа исходного кода приложения.
Fig. 4. Creating a request by analyzing the application's source code.

Полученные строковые переменные в ходе рекурсивного поиска формируют тестовое множество из которых требуется воссоздать структуры HTTP-пакетов для формирования шаблонов тестирования. Главной задачей является определение полного множества структур всех возможных запросов к серверным точкам. Определим это множество:

$$E = \{E_1, E_2, \dots, E_n\} \quad (1)$$

где каждая конечная точка E_i – это кортеж:

$$E_i = (method_i, path_i, queries_i, headers_i, cookies_i, body_i) \quad (2)$$

Здесь ***method*** $\subseteq M =$ *{GET, POST, PUT, DELETE, PATCH, HEAD, OPTIONS, ...}* – все возможные методы запросов, а ***path*** – множество серверных точек (ресурсы, объекты, API-функции и тд). Для точек могут соответствовать различные параметры запросов [14], как правило, передаваемые после знака *"?"*, в формате ключ/значение и перечисленные через амперсенд *"&"*.

$$queries \subseteq Q = \{(k, v) \vee k \in K_{query}, v \in V\} \quad (3)$$

где K_{query} – множество ключей query-параметров, а V – множество значений.

$$headers \subseteq H = \{(k, v) \vee k \in K_{header}, v \in V\} \quad (4)$$

где K_{header} – это все возможные заголовки пакета.

Все значения объединяются в единое множество для более полного покрытия. При этом заголовок ***Cookie*** обладает, как правило, большой вложенностью (токены авторизации, идентификатовы, флаги, данные и т. д.) [14].

$$cookies \subseteq C = \{(name, cookie_{obj}, (k, v)) \vee k \in K_{cookie}, v \in V\} \quad (5)$$

где K_{cookie} – это все возможные заголовки пакета, а *cookie_obj* – структура стандартизованных значений:

$$cookieobj = (value, domain, path, expires, httpOnly, secure, sameSite, \dots) \quad (6)$$

Наибольшей вариативностью обладает тело запроса $body \in B$, где B – множество возможных данных в различных форматах: *{application/json, application/xml, application/x-www-form-urlencoded, multipart/form-data, text/plain, application/octet-stream, ...}*. Также необходимо учитывать и кодировку.

$$B = (b, g) \vee b \in \Sigma * \wedge |b| \geq 0 \wedge g \in G \quad (7)$$

где G – множество возможных кодировок {UTF-8, ASCII, ISO-8859-1, Windows-1251, UTF-16, UTF-32, binary, …}, а g – кодировка тела запроса.

Для извлечения искомых последовательностей в коде мобильных приложений могут быть использованы определенные структуры, например, представленные в Табл. 1.

Табл. 1. Примеры функций кода мобильных приложений, которые могут содержать структуры данных для формирования тестового множества.
Table 1. Examples of mobile application code functions that may contain data structures for generating a test set.

Swift (IOS)	**Kotlin (Android)**
URLQueryItem(name: ..., value: ...) – формирование query-параметров	HttpUrl.Builder().addQueryParameter(…) – сборка query-строки.
request.httpBody = … – ручное заполнение тела запроса	RequestBody.create(...) – ручное формирование тела запроса
JSONSerialization.data(withJSONObject: ...) – сериализация JSON без схемы	Gson().toJson(...) – сериализация без строгой схемы
UserDefaults.standard.string(forKey: ...) – чтение из хранилища (может быть	SharedPreferences.getString(...) – чтение из локального хранилища
UITextField.text – ввод от пользователя без валидации.	EditText.getText().toString() – необработанный ввод от пользователя

Представим поверхность атаки, как объединение всех входных точек, где могут передаваться вредоносные последовательности (точки внедрения для E):

$$S_{attack} = \bigcup_{E \in API} T_E \quad (8)$$

Множество атакующих векторов могут определены, как декартово произведение:

$$A = M \times E \times Q \times H \times C \times B \quad (9)$$

Для покрытия проверками (аудите) WEB-сервиса соответствующего МП необходимо построить отображение, $\Phi: R \to 2^A$, которое по элементу объектов анализа выдаёт все

комбинации $(m, e, q, h, c, b) \in A$, потенциально затрагиваемые этим элементом. Тогда, оценка покрытия:

$$coverage = \frac{|\Phi_{max}|}{|A|} \times 100\% \quad (10)$$

при следующем числе комбинаций

$$A_{max} = \sum_{i=1}^{\Phi} A(n, m) \quad (11)$$

где n – количество текстовых значений, а m – количество позиций в запросе.

4. Признаки WEB-взаимодействий

Основными маркерами присутствия WEB-составляющей можно выделить следующие:

- Методы и функции формирования HTTP (GET, POST, PUT, DELETE и др.);
- Методы и функции работы с форматами представления данных, например, JSON/XML (сериализация/десериализация данных);
- Функции и классы управление сессиями и токенами (OAuth, JWT);
- Использование URL-схем (например, https://api.example.com/v1/users);
- Обработчики статусов ответов (200, 401, 404, 500 и т. д.).

Множество всех возможных вариантов реализации – конечно. Оно определено функциями, методами и классами на разных языках разработки для платформ Android или iOS. В табл. 2 приведен пример нескольких реализаций.

Анализ МП в iOS возможен через программы аудита, например, otool. Это стандартная консольная программа в macOS/iOS, предназначенная для анализа динамических библиотек. Для систем Android может использоваться apkanalyzer[15], dex2jar[16] или JD-GUI / JADX[13]. При этом существуют и специализированные решения для анализа поверхности атаки: Natch [17] – это инструмент для определения поверхности атаки, применимый, в том числе и для мобильных приложений. Он позволяет выявлять серверные точки (интенты, URL-схемы), строить графы компонентов выявляя интерфейсы взаимодействий. Инструмент PANDA (Platform for Architecture-Neutral Dynamic Analysis) [18] - программа для полносистемного анализа, которая позволяет эмулировать выполнение кода приложений, а также записывать и анализировать трассы выполнения. Инструмент SootUp [10] предоставляет обширный набор готовых примитивов и методов для статического анализа: от построения графа вызовов приложения до готовых реализаций алгоритмов статического анализа, например, анализа достижимых определений.

Вышеуказанные инструменты [10, 17, 18] вполне применимы для поиска соответствующих интерфейсов, однако они не являются инструментами динамического анализа и функциональных возможностей для формирования конечного тестового множества шаблонов для проведения анализа защищенности не содержат. Вероятно, указанные методы в текущей статье позволят обогатить текущий инструментарий более широкой функциональностью.

5. Исследование применимости методов

Для верификации гипотезы о применимости исследуемых методов был разработан экспериментальный стенд, включающий два мобильных устройства на платформах Android и iOS, соответственно. Для каждого из устройств было разработано МП с идентичным набором функциональных возможностей, что позволило обеспечить сопоставимость условий эксперимента и корректность сравнительного анализа. Оба приложения содержат ряд функций, формирующих запросы к серверным точкам. Состав и структура запросов строго

определены. Серверная часть едина для двух приложений: аналогия с тем, как устроена архитектура мобильных приложений в большинстве организаций.

Табл. 2. Примеры функций, методов и классов на разных языках разработки для платформ Android или iOS для взаимодействия с серверными точками.
Table. 2. Examples of functions, methods, and classes in different development languages for Android or iOS platforms for interacting with server endpoints.

№	Описание реализации	Пример реализации
1	OkHttp – популярный HTTP-клиент Android приложений для языка Java. Наличие импортов okhttp3.* – явный индикатор WEB-взаимодействия.	*OkHttpClient client = new OkHttpClient();* *Request request = new Request.Builder()* *.url("https://localhost.ru/data").build();* *Response response =* *client.newCall(request).execute();*
2	*Retrofit* – типобезопасный HTTP-клиент на основе аннотаций.	*@GET("/users/{id}")* *Call<User> getUser(@Path("id") int userId);*
3	HttpURLConnection – встроенный HTTP-клиент Java для Android	URL url = new URL("https://api.example.com/data"); HttpURLConnection conn = (HttpURLConnection) url.openConnection(); conn.setRequestMethod("GET");
4	NSURLSession – основной фреймворк для сетевой коммуникации iOS приожений для языка objectiveC.	NSURLSession *session = [NSURLSession sharedSession]; NSURLSessionDataTask *task = [session dataTaskWithURL:[NSURL URLWithString:@"https://api.example.com/data"] completionHandler:^(NSData *data, NSURLResponse *response, NSError *error) {}];

5	package:http/http.dart – кросс-платформенный фреймворк Flutter для Android	import 'package:http/http.dart' as http;
		final response = await
		http.get(Uri.parse('https://api.example.com/data'));
		if (response.statusCode == 200) { }

6	В React Native используется fetch () (язык JavaScript)	fetch('https://api.example.com/data') .then(response => response.json()) .then(data => console.log(data));

Всего в наборе пять API-функций. Сервер приложений написан на языке Python. Запросы для данных функций имеют следующую структуру (сокращенно, без заголовков):

1) GET /api/v1/auth?login=fjdk3mnkcjsadLLL&passwd=32GGhbu76jkhytT5t

2) POST /api/v1/main, с телом запроса: mainscreen=main&role=adminhbu76jkh!!!;%№

3) POST /api/v1/payment, с телом запроса: sum=8372.23&shop446BF_id=7b4c8b91

4) POST /api/v1/balance, с телом запроса: b@lanceFOR=10000&currency=usd

5) GET /api/status?payment4433Fx1_id=8387fd22

Для указанных запросов (3-5) были разработаны дополнительные функции на стороне сервера приложений имитирующие различные дефекты ИБ. Для запроса номер 3 был разработан функционал, который имитирует наличие дефекта ИБ типа RCE (Remote Code

Execution) [19]: если для параметра shop446BF_id передать значение "*__import__('os').system('rm -rf /')*", то данная системная команда будет выполнена на сервере функцией eval(). Для запроса номер 4 имитацией является дефект ИБ с типом Path Traversal (обход пути) [20]: если для параметра b@lanceFOR передать значение "../../../../etc/passwd", то в ответе от сервера будет получен файл passwd, который содержит информацию о пользователях системы. Для запроса номер 5 имитацией является дефект ИБ с типом SQL-инъекция [21]: если для параметра payment4433Fx1 передать значение "*1' UNION SELECT version() --*", то в ответе от сервера вернется версия SQL- базы данных. Синтаксически вышеприведенные названия параметров были подобраны таким образом, чтобы методы простого перебора занимали много ресурсов и времени. Это позволит сделать выводы об эффективности исследуемых методов. Аналогичным сложным образом назначены значения для запроса 1, который имитирует дефект ИБ с типом Broken Access Control (BAC) [22], когда, например, логин (*login*) и пароль (*passwd*) от входа в систему был явным образом указан в исходном коде мобильного приложения (разработчики могут оставить эту пару в комментариях). Перебор вышеуказанных значений невозможен за разумное время. Для запроса 2, который имитирует нарушение ролевой модели (RBAC) [22], если передать указанное значение для параметра *role*, то в ответе сервера вернется контент с функциональными возможностями администратора. Все перечисленные дефекты ИБ входят в перечень OWASP-Top10 [23].

Для качественной оценки были исследованы еще два приложения, опубликованные в открытом доступе: Android-InsecureBankv2 [12] для платформы Android и iGoat-Swift [24] для платформы iOS. Для приложения Android-InsecureBankv2 характерны 4 дефекта ИБ, в соответствии с описанием в их публичных репозиториях, связанные с серверными точками: Parameter Manipulation, Username Enumeration issue, Hardcoded secrets, Developer Backdoors. Определены следующие запросы, которые могут быть использованы для анализа:

POST /login, с телом username=alice&password=secret123
POST /getaccounts, с телом username=alice&password=secret123
POST /changepassword, с телом username=alice&newpassword=newSecret456
POST /dotransfer, с телом username= alice&password=secret123&from_acc=12345& to_acc=67890&amount=100
POST /devlogin, с телом username=dev_user

Для приложения iGoat-Swift определены следующие запросы, которые могут быть использованы для анализа, при этом только запрос */igoat/token* мог использоваться для оценки при поиске дефектов ИБ:

POST /igoat/user
POST /igoat/exercise/certificatePinning
GET /igoat/token?username=<username>&password=<password>
GET /igoat/exercise/certificatePinning

Выбор последних двух приложений для исследования не случаен, они создавались авторами для сообществ ИБ, как заведомо уязвимые для тренировок и повышения компетенцией ИБ у разработчиков мобильных платформ. Дефекты ИБ заранее известны, что позволяет оценить итоговое покрытие проверками.

Существует множество других тренировочных уязвимых приложений, например, InjuredAndroid [25], OVAA (Oversecured Vulnerable Android App) [26], AndroGoat [27]. Однако, они не содержат функций взаимодействия с сервером и не рассматриваются как кандидаты для анализа.

Для объектов исследования, в соответствии со схемой (1), были выполнены все этапы, позволяющие определить тестовое множество HTTP-запросов к серверу. Затем это множество было преобразовано в соответствующий формат шаблонов для сканирований инструментами nuclei [28] и OWASP ZAP [29]: фактически был получен список запросов к серверу, последовательно выполняемый инструментами с автоматической подстановкой полезных нагрузок (сигнатур дефектов ИБ) и элементами фаззинга (мутациями). Пример такого шаблона для nuclei в формате YAML:

```yaml
id: owasp-top10-check
info:
  name: OWASP Top 10
  author: name
  description: |
    Сканирование query-параметров
  severity: high
  tags:
    - owasp-top10
    - owasp-a03
    - sql-injection
    - version-leak
    - api
requests:
  - method: GET
    path:
      - "{{BaseURL}}/api/status?payment4433Fx1_id={{payment_id}}"
    payloads:
      payment_id: payment-id-payloads.txt  # Файл с полезными нагрузками
    attack-type: sniper
    headers:
      User-Agent: Mozilla/5.0 (compatible; Nuclei SQL Scan)
      Accept: */*
    # Подгрузка признаков для срабатывания
    matchers-file: matchers.yaml
    extractors:
      - type: regex
        name: sql_version
        regex:
          - "(?i)(MySQL|MariaDB|PostgreSQL|Microsoft SQL Server|Oracle|SQLite).*?([0-9]+\.[0-9]+(?:\.[0-9]+)?)"
        part: body
      # Код HTTP
      - type: status
        name: http_code
    # Настройки выполнения
    rate-limit: 1
    max-retries: 2
    timeout: 10
    threads: 3
    delay: 1000
```

Рис. 5 Пример шаблона для сканирований запроса GET /api/status.
Fig. 5. Example template for request GET /api/status.

Для определения структур запросов перед формированием шаблонов использовался инструмент [30], который осуществляет поиск и анализ специальных последовательностей API-функций, а также директорий в серверной части приложений, что позволяет сформировать множество тестируемых объектов. Благодаря выявленным серверным точкам и структурам становится возможным создать шаблоны для тестирований. Для фиксации обработки запроса и его корректности производилось логирование в серверной части и фиксация ответов сервера. Критерии оценки:

1) отношение количества найденных к общему количеству всех известных дефектов ИБ;
2) отношение количества найденных структур запросов к серверным точкам к общему количеству всех известных API-функций, отвечающих за обработку запросов;

Результаты применения полученных шаблонов для инструментов приведены в табл. 3.

Результаты применения инструментов Nuclei и OWASP ZAP без предварительного определения шаблонов и поверхности атаки (методом «черного ящика») представлены в Табл. 4 (запуск в режиме «по умолчанию»).

Как видно из результатов в таблицах 3 и 4, тестирование WEB-сервисов для МП без превентивного анализа серверных точек неэффективно, поскольку дефекты ИБ не были найдены вовсе. В ходе экспериментального исследования применимости методов

установлено, что точная семантическая локализация позиций идентифицированных строковых значений в структуре HTTP-запроса представляет собой сложную задачу ввиду многообразия структур и многоуровневой логики, однако, удалось достичь существенно высокого уровня результативности.

Табл. 3. Результаты поиска и обнаружения указанных дефектов ИБ серверной части.
Table 3. Results of searching and detecting the specified security defects in the server part.

Объект исследования	**Nuclei** (найдено/всего дефектов)	**OWASP ZAP** (найдено/всего дефектов)	**Процент выявленных дефектов ИБ**
1	5/5	5/5	100 %
2	5/5	5/5	100 %
3	4/4	4/4	100%
4	1/1	1/1	100%

Таблица 4. Результаты поиска и обнаружения указанных дефектов ИБ серверной части без предварительного анализа МП.
Table 4. Results of searching and detecting the specified server-side security defects without prior analysis of the mobile applications.

Объект исследования	**Nuclei** (найдено/всего дефектов)	**OWASP ZAP** (найдено/всего дефектов)	**Процент выявленных дефектов ИБ**
1	0/5	0/5	0 %
2	0/5	0/5	0 %
3	0/4	0/4	0 %
4	0/1	0/1	0 %

Заключение

В ходе проведенного исследования был рассмотрен подход к анализу исходного кода МП, направленный на выявление механизмов взаимодействия с серверными точками. Предложен метод автоматизированного извлечения структур запросов к серверным API исключительно на основе анализа клиентского кода мобильных приложений. Это позволяет определять поверхность атаки без доступа к серверной инфраструктуре и документации, что принципиально отличает данный подход от методов, требующих анализа серверной составляющей или перехвата сетевого трафика. Разработан унифицированный метод представления точек взаимодействия, инвариантный относительно мобильной платформы (Android/iOS) и языка реализации (Java, Kotlin, Swift, Objective-C). Предложен механизм трансляции результатов анализа кода в формализованные шаблоны для инструментов динамического тестирования (DAST) и фаззинга. Данная концепция устраняет разрыв между этапами анализа и сокращает ручные трудозатраты на конфигурирование сканеров. С теоретической точки зрения исследование вносит вклад в развитие методологии анализа защищенности систем организаций, формализуя процедуру раннего выявления потенциальных векторов атак.

В результате экспериментальной апробации проанализированы 4 мобильных приложения, в которых выявлены все точки взаимодействия с серверными API. Экспериментально подтверждена полнота обнаружения на уровне 100% для явно определённых API-функий. По сравнению с методами динамического анализа, требующими инструментации среды выполнения и наличия работающего серверного окружения, предложенный подход позволяет выполнять первичное картирование поверхности атаки на основе только кода МП. Успешно решена ключевая задача – точное определение поверхности атаки мобильных

приложений без необходимости анализа серверной составляющей. Экспериментально подтверждено существенное повышение точности определения поверхности атаки и обеспечение превентивного выявления дефектов ИБ на ранних стадиях разработки.

Этот подход может быть интегрирован в процессы комплексного анализа защищённости WEB-приложений. Сформированные в рамках любых приложений шаблоны могут быть связаны версионированием с соответствующими WEB-сервисами в организациях. Они могут быть встроены в жизненный цикл безопасной разработки или применены в методологии автоматизации процессов разработки, которая включает непрерывную интеграцию (Continuous Integration, CI) и непрерывную доставку (Continuous Delivery/Deployment, CD).

Полученные результаты закладывают основу для создания автоматизированных инструментов, способных на базе анализа кода генерировать целевые шаблоны для последующего динамического анализа (DAST) и фаззинга, реализуя тем самым замкнутый цикл автоматизированного тестирования безопасности.

Таким образом, предложенный подход подтвердил свою работоспособность и может служить основой для создания масштабируемых решений в области автоматизированного анализа безопасности мобильных приложений, обеспечивая:

- повышение точности определения поверхности атаки;
- сокращение временных затрат на тестирование ИБ;
- интеграцию практик безопасности в жизненный цикл разработки ПО с унификацией процессов анализа для гетерогенных мобильных экосистем.

Направления дальнейших исследований включают расширение метода на анализ обфусцированного кода, интеграцию с системами машинного обучения для классификации выявленных серверных точек по уровню критичности, а также разработку механизмов автоматической генерации тестовых сценариев на основе семантики обнаруженных API.

Список литературы

[1]. ГОСТ Р 56939–2024. Защита информации. Разработка безопасного программного обеспечения. Общие требования. Введ. 2024-01-01. Доступ из справ.-правовой системы «КонсультантПлюс» по ссылке: https://docs.cntd.ru/document/1310017763, дата обращения: 17.01.2026.

[2]. Сафин Л.К., Чернов А. В., Александров Я. А., Трошина К. Н. Исследование информационной защищенности мобильных приложений. Вопросы кибербезопасности, 2015, № 4 (12), с. 28-37.

[3]. Груммет В.А., Лисовин О.А., Фешина Е.В., Куштанок С.А. Способы защиты мобильного приложения под Android. Наука XXI века: проблемы, перспективы и актуальные вопросы развития общества, образования и науки: материалы междунар. межвуз. науч.-практ. конф., пос. Яблоновский, 25 сент. 2020 г. Яблоновский: ФГБУ «РЭА» Минэнерго России, 2020, с. 53-57.

[4]. Nan Y., Zhang L., Yang J., Wang S. Identifying user-input privacy in mobile applications at a large scale. IEEE Transactions on Information Forensics and Security, 2017, vol. 12, no. 3, pp. 647-661. DOI: 10.1109/TIFS.2016.2631949.

[5]. Zhao Q., Zuo C., Dolan-Gavitt B., Pellegrino G., Lin Z. Automatic uncovering of hidden behaviors from input validation in mobile apps. Proceedings of the IEEE Symposium on Security and Privacy, 2020, pp. 1106-1120. DOI: 10.1109/SP40000.2020.00072.

[6]. Qin J., Zhang H., Guo J., Wang S., Wen Q., Shi Y. Vulnerability detection on Android apps inspired by case study on vulnerability related with web functions. IEEE Access, 2020, vol. 8, pp. 1–11. DOI: 10.1109/ACCESS.2020.2998043.

[7]. Network and Distributed System Security Symposium (NDSS 2023). Available at: https://dblp.org/db/conf/ndss/ndss2023.html, accessed 17.01.2026.

[8]. netTrace. Available at: https://github.com/twabbott/netTrace, accessed 17.01.2026.

[9]. ACM Conference on Computer and Communications Security (CCS 2022). Available at: https://dblp.org/db/conf/ccs/ccs2022.html, accessed 17.01.2026.

[10]. Ломоносовские чтения: тезисы докладов науч. конф. Москва, 24 марта – 4 апр. 2025 г. Москва: МАКС Пресс, 2025. 200 с. ISBN 978-5-317-07378-7. DOI: 10.29003/m4389.978-5-317-07378-7.

[11]. OWASP Mobile Application Security Testing Guide (MASTG). Available at: https://mas.owasp.org/MASTG/, accessed 17.01.2026.
[12]. Android InsecureBank v2. Available at: https://github.com/dineshshetty/Android-InsecureBankv2.
[13]. JADX: Dex to Java decompiler. Available at: https://github.com/skylot/jadx, accessed 17.01.2026.
[14]. RFC Editor. Available at: https://www.ietf.org/process/rfcs/, accessed 17.01.2026.
[15]. APK Analyzer. Available at: https://developer.android.com/tools/apkanalyzer?hl=ru, accessed 17.01.2026.
[16]. Dex2jar. Available at: https://github.com/pxb1988/dex2jar, accessed 17.01.2026.
[17]. NATCH. Available at: https://www.ispras.ru/technologies/natch/, accessed 17.01.2026.
[18]. Panda Reverse Engineering. Available at: https://panda.re/, accessed 17.01.2026.
[19]. CWE-94: Improper Control of Generation of Code. Available at: https://cwe.mitre.org/data/definitions/94.html, accessed 17.01.2026.
[20]. CWE-35: Path Traversal. Available at: https://cwe.mitre.org/data/definitions/35.html, accessed 17.01.2026.
[21]. CWE-89: SQL Injection. Available at: https://cwe.mitre.org/data/definitions/89.html, accessed 17.01.2026.
[22]. CWE-284: Improper Access Control. Available at: https://cwe.mitre.org/data/definitions/284.html, accessed 17.01.2026.
[23]. OWASP Top Ten. Available at: https://owasp.org/www-project-top-ten/, accessed 17.01.2026.
[24]. iGoat-Swift. Available at: https://github.com/OWASP/iGoat-Swift, accessed 17.01.2026.
[25]. InjuredAndroid: Android security playground. Available at: https://github.com/B3nac/InjuredAndroid, accessed 17.01.2026.
[26]. OVAA (Oversecured Vulnerable Android App). Available at: https://github.com/oversecured/ovaa, accessed 17.01.2026.
[27]. AndroGoat: vulnerable Android application. Available at: https://github.com/satishpatnayak/AndroGoat, accessed 17.01.2026.
[28]. Nuclei: vulnerability scanner. Available at: https://github.com/projectdiscovery/nuclei, accessed 17.01.2026.
[29]. OWASP ZAP: Zed Attack Proxy. Available at: https://www.zaproxy.org/, accessed 17.01.2026.
[30]. Юрьев А.С. Свидетельство о государственной регистрации программы для ЭВМ № 2025664231 Российская Федерация. PointTargetSearch: заявл. 15.05.2025: опубл. 03.06.2025. EDN TCNACE.

Информация об авторах / Information about authors

Шамиль Фаимович КУРМАНГАЛЕЕВ – кандидат физико-математических наук. Сфера научных интересов: Статический анализ бинарного кода, динамический анализ, фаззинг, обфускация, символьное выполнение.

Shamil Faimovich KURMANGALEEV – Cand. Sci. (Phys.-Math.), Senior Researcher. Research interests: Static analysis of binary code, dynamic analysis, fuzzing, obfuscation, symbolic execution

Артемий Сергеевич ЮРЬЕВ – аспирант ИСП РАН. Научные интересы: информационная безопасность, фаззинг информационных систем, анализ защищенности, тестирование на проникновение, динамическое сканирование, безопасная разработка.

Artemiy Sergeevich YUREV – postgraduate student of ISP RAS. Research interests: information security, fuzzing of information systems, security analysis, penetration testing, DAST, SSDLC.

Никита Александрович ДАЦУК – аспирант МГТУ им. Н.Э. Баумана. Научные интересы: информационная безопасность, тестирование на проникновение, криптография, динамический анализ приложений, безопасная разработка, проектирование микросхем.

Nikita Alexandrovich DATSUK – postgraduate student of BMSTU. Research interests: information security, penetration testing, cryptography, dynamic application analysis, secure development, chip design.