Содержание

Введение **17**

Актуальность пентеста кроссплатформенных приложений в ближайшие годы . 19

Особенности безопасности: нативные vs кроссплатформенные приложения (React Native, Python Kivy/BeeWare)..... 21

Общие уязвимости для кроссплатформенных решений 22

Обзор современного стека технологий и инструментов для анализа..... 23

ГЛАВА 1. **Архитектура кроссплатформенных приложений** .. **25**

1.1. Архитектура кроссплатформенных приложений 26

1.2. React Native: JavaScript-мост и нативные модули 28

1.2.1. Что такое React Native? 29

Базовый пример: простое приложение-счётчик..... 30

Что здесь происходит и как это становится нативным приложением? 31

1.2.2. Ключевые компоненты архитектуры 32

Нативный поток (UI Thread) 34

JavaScript-мост (The Bridge) 34

Нативные компоненты (Native Components) 34

Теневой поток и диспетчер представлений (Shadow Thread & Layout) 35

1.2.3. Зачем нужны нативные модули?..... 35

Практический пример – нативный модуль для работы с файловой системой..... 36

1.2.4. Сборка приложения: что попадает на телефон? 40

Исходный код приложения..... 40

Что происходит при сборке и что оказывается внутри APK? ... 40

Пример реального нарушения безопасности в бандле..... 43

Практические рекомендации 43

1.2.5. JavaScript-мост: "сердце" React Native 44
Что происходит внутри моста на каждом этапе? 45
Главные технические особенности моста, влияющие на безопасность 47
Пример уязвимости, связанной с работой моста 47
Инструменты для анализа работы моста 48
Вывод раздела 49
Для безопасности 49

1.3. Python-фреймворки (Kivy, BeeWare): взаимодействие с платформой, использование библиотек 50

Основные риски для Python-приложений 51
1.3.1. Общая идея: Python на мобильных 52
Исходный код приложения (чистый Python) 52
Что происходит при сборке этого кода в мобильное приложение? 53
Ключевое отличие от React Native 55
Пример взаимодействия с платформой через ***pyjnius*** (Android) 55
Пример уязвимости, специфичной для этого подхода 56
Вывод раздела 56
1.3.2. Kivy: кроссплатформенный UI с собственным движком 57
Полный исходный код приложения 57
Как этот код превращается в мобильное приложение? 60
Проблемы безопасности, специфичные для Kivy 62
Пример сборки и запуска 63
Преимущества подхода Kivy 63
Недостатки 64
Вывод раздела 64
1.3.3. BeeWare (Toga): Нативные UI из Python 64
Методика пентеста приложений на BeeWare/Toga 69
Пример скрипта Frida для перехвата ***pickle.loads*** 69
Рекомендации по защите 70
1.3.4. Использование библиотек Python (главная сила и риск) 70
Типичная структура зависимостей в APK-файле 71
Примеры опасного использования библиотек в коде приложения 74
Практические примеры эксплойтов для зависимостей 76

Рекомендации по аудиту библиотек Python 77
1.3.5. Простой итог для понимания....78
Практический ориентир для пентестера 81
1.4. Общие компоненты: WebView, хранилище и сетевые запросы....81
1.4.1. WebView: окно в Интернет внутри приложения 84
Базовый пример интеграции WebView в React Native....85
Пример уязвимой реализации в Kivy (Python) 86
Эксплойты для уязвимого WebView....89
Атаки на WebView и методы пентеста 89
Рекомендации по защите WebView....92
1.4.2. Хранилище: где приложение хранит свои данные 93
Основные типы хранилищ в кроссплатформенных приложениях 93
Методика пентеста хранилищ 99
Практические эксплойты для извлечения данных....104
Рекомендации по защите....105
Практические замечания 107

ГЛАВА 2. **Настройка лаборатории для пентеста**.....**109**

2.1. Настройка лаборатории для пентеста....110
Эмуляторы и реальные устройства для Android и iOS....110
Установка и настройка ключевых инструментов (Frida, MobSF, Charles Proxy/Burp Suite)....111
Подготовка Python-окружения для кастомного анализа....112
2.2. Эмуляторы и реальные устройства для Android и iOS....113
Для Android: реальные устройства с Root-правами....114
Для iOS: симулятор – непригоден для пентеста....114
Для iOS: реальные устройства с Jailbreak....115
2.3. Установка и настройка ключевых инструментов (Frida, MobSF, Charles Proxy/Burp Suite)....115
Frida: инструмент динамической инструментации....116
MobSF: статический анализ в один клик 116
Базовая настройка прокси 117
2.3.1. Frida: инструмент для "живого" анализа приложений 118

Основные возможности Frida 118
Архитектура Frida 119
Установка Frida 119
Базовые примеры использования Frida 119
Примеры скриптов для конкретных задач пентеста 121
Практические примеры для пентеста кроссплатформенных приложений 125
Практические сценарии использования Frida для пентеста 133
2.3.2. MobSF (Mobile Security Framework): автоматический анализатор 134
Основные возможности MobSF 134
Настройка MobSF для кроссплатформенных приложений 136
Анализ React Native приложений с MobSF 137
Практическое использование MobSF для пентеста 151
2.3.3. Charles Proxy / Burp Suite: перехватчики сетевого трафика 152
Charles Proxy: профессиональный HTTP-прокси 152
Burp Suite: комплексный фреймворк для пентеста веб-приложений 156
Практические скрипты для анализа трафика кроссплатформенных приложений 162
Практические советы для пентеста кроссплатформенных приложений 170
2.4. Подготовка Python-окружения для кастомного анализа 171
Создание изолированной среды 171
Установка основных инструментов пентеста 171
Специализированные утилиты для кроссплатформенного анализа 172
Разработка собственных скриптов и автоматизация 172
Управление зависимостями и воспроизводимость 173

ГЛАВА 3. **Атаки на слой взаимодействия (Bridge/WebView) 175**

3.1. Атаки на слой взаимодействия (Bridge/WebView) 177
Инъекции через JavaScript-мост в React Native 177
Анализ и эксплуатация уязвимых нативных модулей 178
Атаки на гибридные компоненты WebView (XSS, обход SSL-пиннинга) 178

Интеграция атак: пример сценария 179

3.2. Инъекции через JavaScript-мост в React Native 180

Механика уязвимости....180
Методология поиска и эксплуатации 180
Особенности и сложности 182
Защита и выводы для пентестера 182

3.3. Анализ и эксплуатация уязвимых нативных модулей....182

Методология анализа нативных модулей....183
Типовые сценарии эксплуатации 184

3.4. Атаки на гибридные компоненты WebView (XSS, обход SSL-пиннинга) ...186

XSS в контексте WebView: от скрипта к нативному коду....186
Обход SSL-пиннинга в WebView....187

Практический итог раздела 189

ГЛАВА 4. **Уязвимости данных и сериализации** **191**

4.1. Уязвимости данных и сериализации 193

Утечки данных через логи, кеш и буфер обмена 193
Небезопасная десериализация в Python-компонентах (Pickle, YAML) .. 194
Анализ локальных хранилищ (AsyncStorage, SQLite, файлы)....194

4.2. Утечки данных через логи, кеш и буфер обмена 195

Логи (Logging) как источник утечек....196
Кеш – забытое хранилище....197
Буфер обмена (Clipboard) – общедоступная область обмена 197
Общая картина риска....198

4.3. Небезопасная десериализация в Python-компонентах (Pickle, YAML)....199

Модуль ***pickle***: встроенная угроза 199
Пример опасного кода в мобильном приложении 199
Библиотека PyYAML и опасный метод yaml.load() 200
Пример опасного YAML....200
Контекст мобильного приложения и источники данных 200
Методика тестирования на уязвимость 201
Защита и выводы....201

4.4. Анализ локальных хранилищ (AsyncStorage, SQLite, файлы) 202

Абстракции фреймворков: AsyncStorage и аналоги 202
Базы данных SQLite: структурированное хранилище с уязвимостями . 203
Файловая система: произвольное хранение данных 203
Типы уязвимых файлов.....204
Методика анализа для пентестера 204
Инструментальный анализ 204
Интеграционный риск.....205

ГЛАВА 5. **Сетевой уровень и криптография** **207**

5.1. Сетевой уровень и криптография 208

5.2. Перехват и анализ сетевого трафика (HTTP/HTTPS, WebSockets) 209

Перехват и анализ сетевого трафика (HTTP/HTTPS, WebSockets) .. 210
Перехват трафика из разных источников 210
Обход SSL-пиннинга в React Native и Python-приложениях 211
Анализ слабых алгоритмов шифрования и небезопасного хранения ключей 212
Интеграция рисков: пример сценария 213

5.3. Обход SSL-пиннинга в React Native и Python-приложениях 213

Понимание реализаций SSL-пиннинга 214
Стратегии и инструменты для обхода 214

5.4. Анализ слабых алгоритмов шифрования и небезопасного хранения ключей 216

Выявление слабых и устаревших криптографических алгоритмов .. 217
Анализ сетевого трафика и полезной нагрузки.....218
Небезопасное хранение и управление криптографическими ключами.....218
Методика динамического анализа с помощью Frida 219

Последствия и выводы раздела 220

ГЛАВА 6. **Защита кода и логики приложения** **223**

6.1. Защита кода и логики приложения.....224

Методы обфускации JavaScript-бандлов в React Native 225

Дублирование и перемешивание кода 225
Защита Python-кода от реверс-инжиниринга (упаковка, обфускация)... 225
Реализация защиты от отладки и анализа (Frida, root/jailbreak детекция) 226
Детекция эмулятора/симулятора 227
Интеграция защиты 228

6.2. Методы обфускации JavaScript-бандлов в React Native 228

Базовый уровень: минификация (Minification) 229
Продвинутая обфускация: трансформация кода 229
Запутывание потока управления (Control Flow Flattening) 229
Строковое шифрование (String Encryption) 229
Преобразование и скрытие доступа к свойствам 230
Внедрение мёртвого (недостижимого) и диверсионного кода 230
Обфускация чисел и вычислений 230
Интеграция в процесс сборки 230
Ограничения и проблемы обфускации 231

6.3. Защита Python-кода от реверс-инжиниринга (упаковка, обфускация) 231

Упаковка (Packaging): создание бинарного исполняемого файла...232
Nuitka – трансляция Python в C 232
Обфускация исходного кода 233
Обфускация на уровне байт-кода 234
Комбинированный подход и практические рекомендации 234

6.4. Реализация защиты от отладки и анализа (Frida, root/jailbreak детекция) ..235

Детекция отладки и инструментации 235
Детекция ***root*** (Android) и ***jailbreak*** (iOS) 236
Детекция эмуляторов и симуляторов 237
Стратегия реакции и противодействия 238

ГЛАВА 7. **Безопасная работа с данными** **241**

7.1. Безопасная работа с данными 243

Шифрование чувствительных данных в Python-приложениях 243
Пример безопасного шифрования с использованием ***cryptography*** ..244
Безопасное хранение ключей, сертификатов и токенов (Keychain/Keystore) 244
Принципы безопасной сериализации данных 245

7.2. Шифрование чувствительных данных в Python-приложениях 247

Выбор правильных криптографических примитивов и библиотек...247
Алгоритмы и параметры 247
Генерация и управление ключом 248
Шифрование с использованием AES-GCM (рекомендуемый способ).. 248
Шифрование с использованием AES-CBC + HMAC (альтернативный способ) 249
Хранение ключей шифрования – центральная проблема 249
Обработка ошибок и побочные каналы 250

7.3. Безопасное хранение ключей, сертификатов и токенов (Keychain/Keystore)..250

Android Keystore System: защищённое хранение ключей 251
iOS Keychain Services: хранение секретов и учётных данных 252
Стратегия хранения ***access*** и ***refresh tokens*** 254
Архитектура "мастер-ключ + производные ключи" 254
Почему одной защиты на устройстве недостаточно 255

7.4. Принципы безопасной сериализации данных.....256

1. Выбор безопасного формата сериализации 257
2. Валидация схемы и содержимого данных.....258
3. Принцип минимальных привилегий при десериализации 259
4. Безопасная кастомная сериализация.....259
5. Межкомпонентное взаимодействие в кроссплатформенных приложениях 260
Итог раздела 261

ГЛАВА 8. **Укрепление сетевого взаимодействия** **263**

8.1. Укрепление сетевого взаимодействия 264

1. Почему одного HTTPS недостаточно 265
2. Проверка подлинности сервера и роль SSL-пиннинга 266
3. Защита от подмены адреса и сетевого маршрута 267
4. Целостность запросов и защита от повторного воспроизведения...267
5. Единый сетевой стек и принцип безопасных значений по умолчанию.....268
6. WebView как отдельная граница доверия 269
7. Кроссплатформенные особенности React Native и Python 270
8. Мониторинг, телеметрия и реакция на ошибки 270

9. Ротация ключей и эксплуатационная устойчивость 271
Практическая модель защищённого сетевого взаимодействия 272
Итог раздела 272

8.2. Правильная реализация SSL-пиннинга....273

1. Что именно следует закреплять....273
2. Основной и резервный отпечатки 274
3. Реализация на Android с OkHttp 275
4. Реализация на iOS с URLSession....276
5. Кроссплатформенные приложения: где должен находиться пиннинг 276
6. Хранение отпечатков и защита клиентской логики....277
7. Область применения и типичные ошибки 278
8. Пиннинг как часть процесса эксплуатации....278
Итог раздела 279

ГЛАВА 9. **Динамический анализ с Frida** **281**

9.1. Динамический анализ с Frida 282

1. Зачем нужен динамический анализ 283
2. Принцип работы Frida 284
3. Хуки как основной инструмент наблюдения....286
4. Frida и кроссплатформенная архитектура....287
5. Анализ сетевого взаимодействия 289
6. Исследование данных в памяти и ***runtime***....290
7. Антиотладочные и антиинструментационные механизмы....291
8. Итеративная стратегия динамического анализа....292
9. Ограничения и правильная интерпретация результатов....293
Итог раздела 293

9.2. Основы Frida: хуки, скрипты, работа с *runtime* 294

Архитектура и принцип работы 294
Ключевые абстракции и API 295
Interceptor – основной механизм перехвата вызовов 295
Платформенные API Frida 296
Типичный рабочий процесс написания и запуска скрипта....296
Пример базового скрипта для Android (Java) 297
Пример базового скрипта для нативной функции (C/C++)....297

Отладка и ***best practices*** 298

9.3. Примеры скриптов для анализа React Native (JavaScript-мост) и Python-приложений 299

Анализ React Native: мониторинг и модификация вызовов нативных модулей 299

Пример для Android: хук нативного модуля, вызываемого из JS 299

Пример для iOS: хук нативного модуля Objective-C 300

Анализ Python-приложений (Kivy/BeeWare): перехват интерпретируемого кода 301

Хук встроенных опасных функций Python (pickle.loads, eval) 301

Анализ сетевых вызовов и шифрования в Python-приложении 302

Универсальный скрипт для первоначальной разведки 303

Практические замечания 303

9.4. Обход анти-отладочных механизмов 304

Типы анти-отладочных механизмов и методы их обхода 304

Стратегия обнаружения и нейтрализации защит 306

Пример комплексного скрипта для начального обхода (Android) 308

Заключение главы 309

ГЛАВА 10. **Автоматизированное сканирование с MobSF .. 311**

10.1. Автоматизированное сканирование с MobSF 312

Роль MobSF в процессе пентеста 313

Что автоматизирует статический анализ 314

Особенности анализа кроссплатформенных приложений 315

От автоматической находки к проверяемой гипотезе 316

Динамический анализ как продолжение статического 316

Ограничения автоматизированного сканирования 317

Место MobSF в дальнейшей методике 317

Вывод раздела 317

10.2. Настройка и работа с Mobile Security Framework (MobSF) 318

Установка и запуск MobSF 318

Базовый рабочий процесс анализа 319

Особенности работы с кроссплатформенными приложениями 321

Практические советы по работе 321

10.3. Анализ отчетов: интерпретация результатов для кроссплатформенных приложений....322

1. Определение технологии и ключевых артефактов....322
2. Приоритизация находок для гибридной архитектуры 322
3. Глубокий анализ специфичных компонентов....324
4. Связывание статических находок с динамическим тестированием .. 324
5. Формирование вывода....325

10.4. Интеграция MobSF в процесс CI/CD 325

Что даёт интеграция в CI/CD 326
Архитектура интеграции 327
Актуальный REST API: базовый цикл 328
Правильные Quality Gates 329
Пример Jenkins Pipeline 330
Особенности кроссплатформенных проектов....334
Базовая линия и снижение шума 334
Производительность и стабильность пайплайна....335
Уведомления и работа с результатами....335
Практическая модель DevSecOps с MobSF 336

ГЛАВА 11. **Кастомный инструментарий на Python**.....**337**

11.1. Кастомный инструментарий на Python....338

Зачем пентестеру собственные инструменты....339
Почему именно Python 340
Какие задачи стоит автоматизировать....341
Архитектура полезной пентест-утилиты 341
Работа с результатами и уровнем уверенности....342
Безопасная обработка недоверенных артефактов....343
Кастомный инструмент как часть общей методики 344
Границы автоматизации....344
Вывод раздела....345

11.2. Написание скриптов для автоматизации анализа (парсинг кода, тесты на проникновение)....345

Парсинг и анализ извлеченного кода....345
Анализ JavaScript-бандла React Native 345
Анализ Python-кода в приложениях Kivy/BeeWare 347

Автоматизация тестов на проникновение 348
Фаззинг нативных модулей React Native через Frida....348
Автоматическое тестирование API эндпоинтов 350
Интеграция в рабочий процесс 352

11.3. Инструменты для анализа трафика и межпроцессного взаимодействия (IPC) 352

Анализ и модификация сетевого трафика 353
Создание кастомного прокси-мидлвара на Python 353
Анализ WebSocket-трафика 353
Работа с TLS и обход SSL-пиннинга на уровне скрипта....354
Анализ межпроцессного взаимодействия (IPC)....354
Анализ ***Intent*** на Android 354
Анализ URL-схем (Custom URL Schemes) на iOS 355
Мониторинг Unix-сокетов и именованных каналов....356
Анализ обмена через Shared Memory или файлы 356
Интеграция с Frida для перехвата IPC....356
Практическое применение....357

11.4. Создание собственных сканеров для специфических уязвимостей 358

Проектирование специализированного сканера: от идеи до реализации 358
Определение цели и сигнатур уязвимости....358
Разработка алгоритма обнаружения 359
Пример 1: сканер для инъекций через React Native Bridge 359
Техническая реализация на Python (с использованием ***androguard***) .360
Пример 2: сканер небезопасной десериализации в Python-приложениях (Kivy)....361
Интеграция и развитие сканеров 364
Вывод раздела....364

Заключение....365

Ключевые векторы атак на кроссплатформенные приложения в ближайшие годы....366

Будущее мобильной безопасности: тренды и прогнозы....367

Рекомендации по построению процесса безопасной разработки (DevSecOps) . 368

Разбор ключевых направлений атак на кроссплатформенные приложения ...369

Атаки на слой взаимодействия (Bridge) и нативные модули 369
Компрометация гибридных компонентов WebView 369
Небезопасная десериализация и выполнение кода в Python-среде ... 370
Утечки данных через побочные и неуправляемые каналы.....370
Слабости в криптографии и сетевой защите 371
Недостатки защиты от анализа и реверс-инжиниринга 371
Вывод раздела.....372

Будущее мобильной безопасности: тренды и прогнозы.....372

Углубление интеграции с облаком и смещение периметра безопасности.....372
Распространение и усложнение Runtime Application Self-Protection (RASP) 373
Методы машинного обучения будут использоваться и в защитных, и в исследовательских инструментах 374
Риски безопасности цепочки поставок (Software Supply Chain) выйдут на первый план.....374
Конвергенция платформ и стирание границ 375
Ужесточение регуляторного давления и роль ***privacy*** 375

Рекомендации по построению процесса безопасной разработки (DevSecOps) . 376

Образование и формирование культуры безопасности (Shift Left Culture).....376
Внедрение автоматизированного статического анализа (SAST) в CI/CD..... 377
Динамический анализ и автоматизированное тестирование (DAST)..377
Защита сборки и артефактов 378
Мониторинг безопасности в рабочей среде и реагирование на инциденты.....378
Регулярный ручной пентест и ***bug bounty*** 379

Список использованных источников информации 381