Секурити тру обскурити – точно всё. Угрозы открытому API и способы защиты от них.

Павел Супрунюк

Технический руководитель направления

Аудита и Консалтинга Group-IB

О чем доклад

1. Чем отличаются подходы к обеспечению практической безопасности OpenAPI и остальных информационных

2. Примеры кейсов из практики проверки безопасности API. Все совпадения – просто совпадения

3. Как защититься – точка зрения практических безопасников

Докладчик – Павел Супрунюк, технический руководитель направления «Аудит и консалтинг» Group-IB

- Занимаюсь тестами на проникновение, безопасностью приложений, red teaming

Отличия в защите OpenAPI от других информационных сервисов

GROUP IB

Взгляд с точки зрения практической безопасности

- С середины 2000-х до настоящего времени банковское ПО эволюционировало от громоздких «толстых клиентов» до унифицированных приложений с единым API.
- Сейчас в большинстве случаев: в неявном для пользователя виде API используются в основе веб-клиентов, мобильных приложений. В явном – в системах автоматизированного обмена с банками (host-to-host, и др.).

Коротко – отличий совсем немного

Технологически – OpenAPI – не что-то инновационное, а еще один интерфейс API. Тестируется схожим образом.

В **организационно-правовом** плане – сложная концепция и большой шаг вперед для индустрии финтеха.

Отличия в защите OpenAPI от других информационных сервисов

Взгляд с точки зрения практической безопасности

Положительные стороны концепции OpenAPI для практической безопасности:

- Психология разработчиков не работает на «скрытие» – obscurity.
- API обычно создаются через профильные фреймворки.
  Это сильно снижает риск атак. Но есть нюанс – бизнес-логика.
- Использование OpenAPI не предполагает использование исключительно браузера как клиента – снижается риск соответствующих атак.
- Разработка профильных отраслевых требований к участникам обмена до начала внедрения сервисов OpenAPI.

Примеры практических кейсов. Раскрытие информации

- То, чего не ожидается в OpenAPI по определению: security through obscurity в API
- Входим в интерфейс ДБО, видим приглашение:

*Здравствуйте, Иван Петрович Б. (+7 9** *** 34-51)*

- Смотрим «под капот», в API:

Запрос мобильного клиента/браузера в ДБО -> GET /api/v1/client_info

Ответ <- {...“first_name”: “Иван”, “last_name”: “**Борисов**”, “reg_series”: “**77 77**”, “reg_number”: “**112233**”, “reg_details”: “**Выдан отделением УФМС….**”, “phone”: “**+79999993451**”, “snils”: “**1234567890**”…}

Примеры практических кейсов. Раскрытие информации

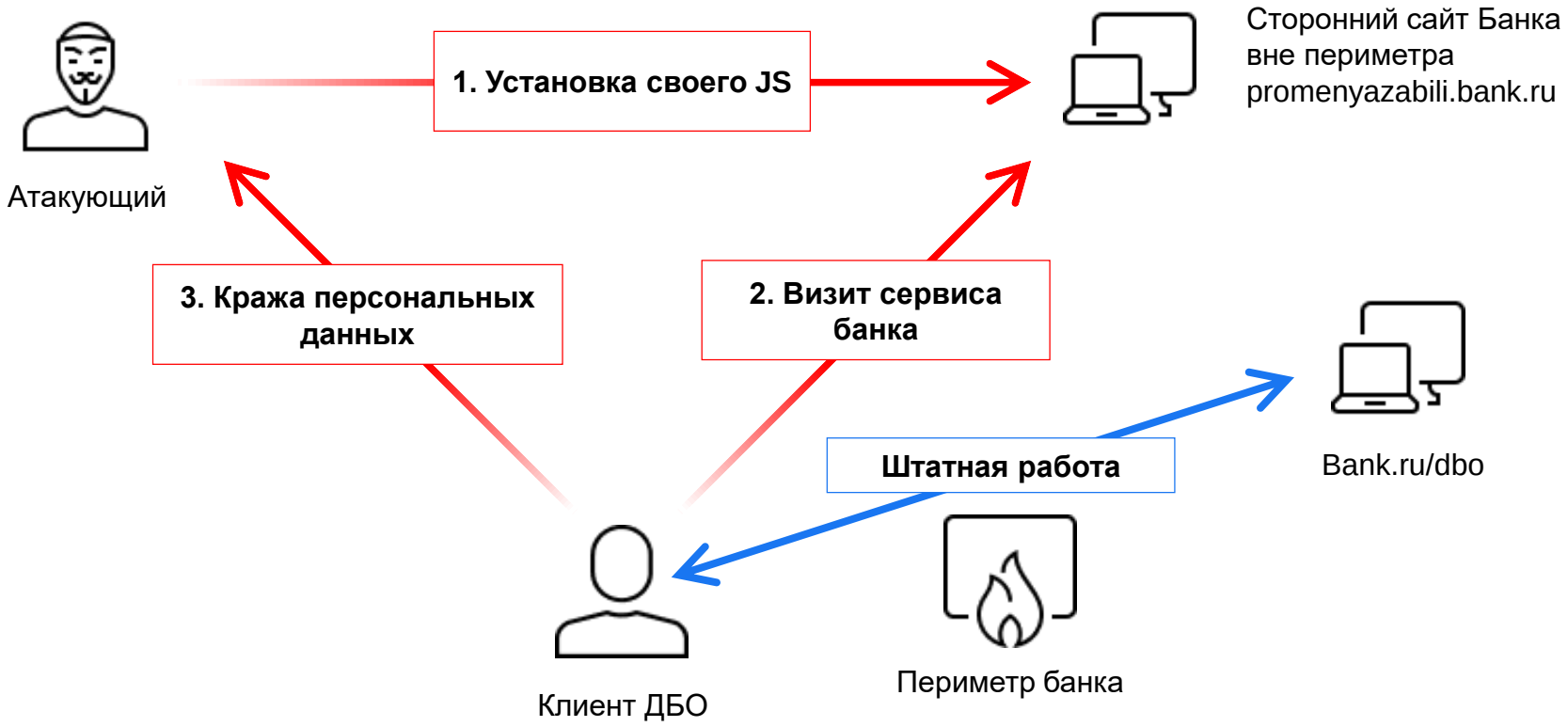
- Далее, находятся уязвимости внедрения JavaScript и неправильной области действия сессии в ДБО.
- В сторонний домен банка внедряется публичный документ с подготовленным JavaScript => массово собираются персональные данные клиентов, случайно зашедших на **другой** официальный сервис банка, **вне периметра**.

! Простейшие утечки через недостаточную авторизацию в API вида GET /api/v3/operation_data/NNNN …… (NNNN+1, NNNN-1, NNNN+2)

или чрезмерное раскрытие данных встречаются в **60-70%** объектов проверки. СЗИ – не помощник, у них просто нет понимания правильного/неправильного запроса.

Примеры практических кейсов. Скрытие информации

То же самое графически



Примеры практических кейсов. Бизнес-логика раз

- Тот самый нюанс. Фреймворки и СЗИ уже слабо помогают.
- Вводная. Есть сервис, есть у него API, есть регистрация пользователя, с последующим скринингом. Пользователь проходит скрининг безопасности, его пускают в систему.
- Запрос на регистрацию от здорового человека:

POST /api/v1/user/register
….
{“name”: “Ivan”, “surname”: “Ivanov”, “email”: “ivan@ivanov.com”, “phone”: “+79999999999”, …}

Примеры практических кейсов. Бизнес-логика раз (продолжение)

- Запрос на регистрацию человека, владеющего исходными кодами и любопытными руками:

POST /api/v1/user/register

….
*{“name”: “Ivan”, “surname”: “Ivanov”, “email”: “ivan@ivanov.com”, “phone”: “+79999999999”, …., **“is_approved”: 1**}*

- Сервер принимает запрос, преобразует в запись в базе данных, автоматически размечает поля по именам.
- Свойство «прошел скрининг» выставляется сразу в момент регистрации. Удобно.

Примеры практических кейсов. Бизнес-логика два

Финансовый сервис (биржа) позволяет торговать чем-то за внутренние деньги сервиса.

Ожидание: клиент спокойно создает заявки, раз в день. Не может работать с двух устройств.

Реальность: клиент очень быстро, используя мобильное API и «настольное» API одновременно, создает заявки на весь свой баланс. И они создаются, т.к. сервис не успевает заблокировать средства еще при первом запросе.

Затем заявки отменяются, у клиента на балансе X2 средств. ~~Процесс повторяется до подозрительного стука в дверь.~~

Примеры практических кейсов. Выводы

Почему так вышло?

Основной ответ:
Нарушен процесс безопасной разработки и внедрения.
При этом с точки зрения мер «все выполнено». СЗИ на месте.

Частые частности:

- Недостаточная квалификация команды разработчиков в части ИБ. Нет понимания, чем руководствоваться при проектировании и разработке.
- Нет мотивации сделать безопасно, есть мотивация «чтобы быстрее заработало, или хотя бы просто заработало».
- Отсутствие тестирования в процессе разработки, кроме функционального.
- Излишние надежды на СЗИ, криптографию и т.д. без сопоставления с реальными рисками.
- Отсутствие проверок внешними силами.

Что делать?

1. Только менять процессы.

2. Повысить квалификацию разработчиков:

- Изучить OWASP – основной фреймворк и чек-лист для тестирования веб-приложений
- Изучить руководства по безопасности для применяемых компонентов
- Добавить в команды специалистов по практической безопасности

3. Скорректировать процесс разработки:

- Применить из моделей разработки ПО «безопасные» части
- Для OpenAPI – применить со старта работ требования новых стандартов Ассоциации ФинТех по OpenAPI, которые включают стандарты по безопасной разработке, управлению уязвимостями, защищенности инфраструктуры

Что делать?

4. Применить СЗИ в соответствии с оценкой рисков и моделями угроз:

- Для API помогут: Антифрод-системы и Web Application Firewall с моделями обучения и поиском аномалий

5. Не отрицать полезность внешней помощи:

- Внешние практические безопасники (≠ обязательному тесту на проникновение!)
- Разработать систему Bug Bounty – все равно будут хактивисты, надо с ними работать

Предотвращаем и расследуем киберпреступления с 2003 года

Павел Супрунюк

Технический руководитель направления
Аудита и Консалтинга Group-IB

suprunyuk@group-ib.com