# Актуальные проблемы API

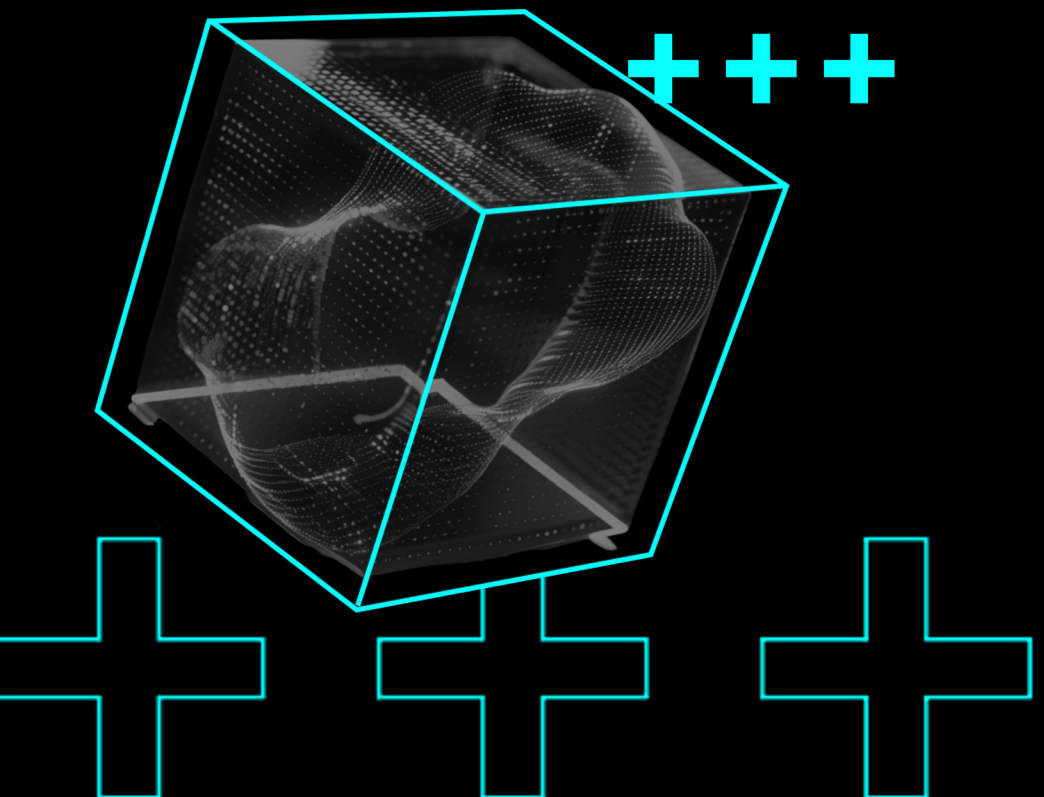

**Костючек Артем**

Старший эксперт в отделе тестирования кибербезопасности Альфа-Банк

# Костючек Артём


- Превращаю request'ы в багрепорты
- Старший эксперт в отделе тестирования кибербезопасности

# Введение



# План доклада:

01 OWASP API Top 10

02 Best Practices

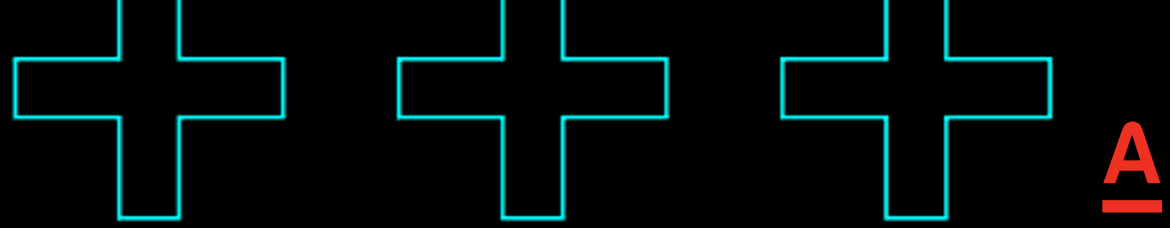
03 Обсуждение

**Цель доклада:** разобраться, почему API — лакомый кусок для атак, и как не дать злоумышленнику его съесть

# Почему API-безопасность — это важно?

API — новый фронт атак. Сегодня почти каждое приложение использует API: мобильные клиенты, SPA, микросервисы, интеграции.

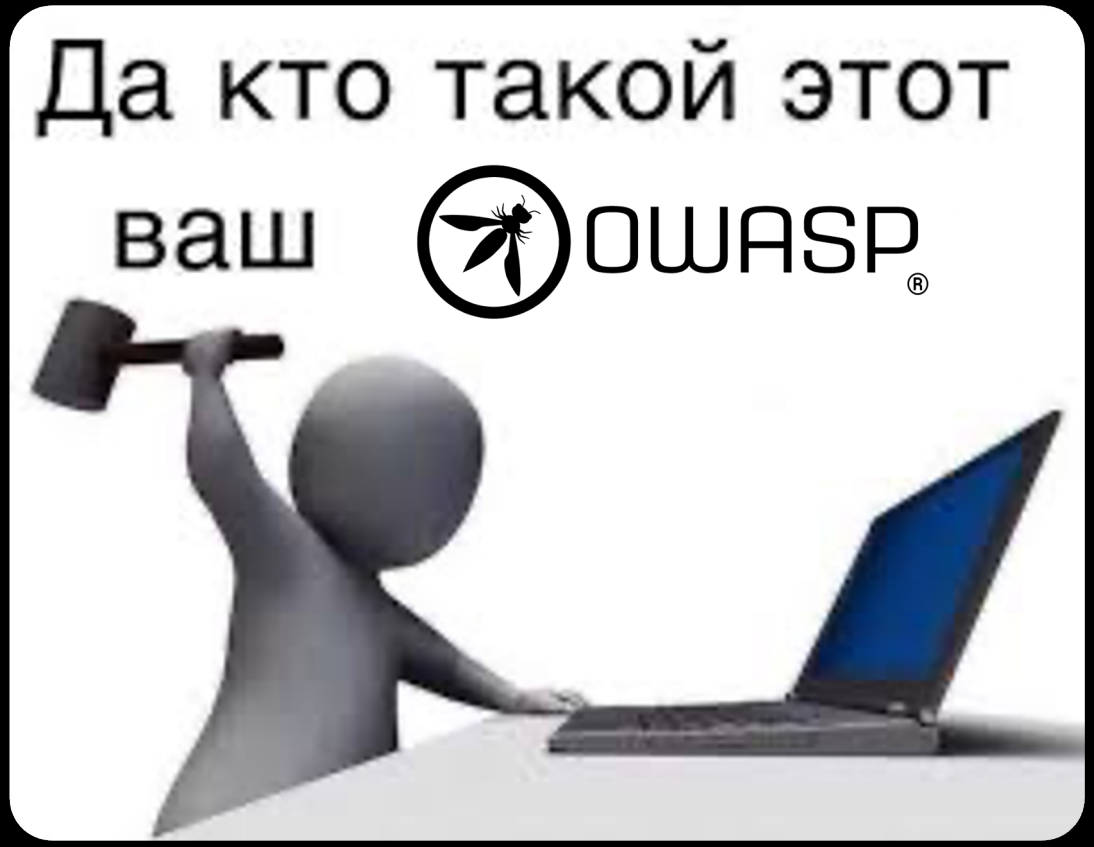
API — двери в логику приложения. Часто через них доступна бизнес-логика, которую можно ломать, не взламывая «оболочку».



# OWASP API Top 10

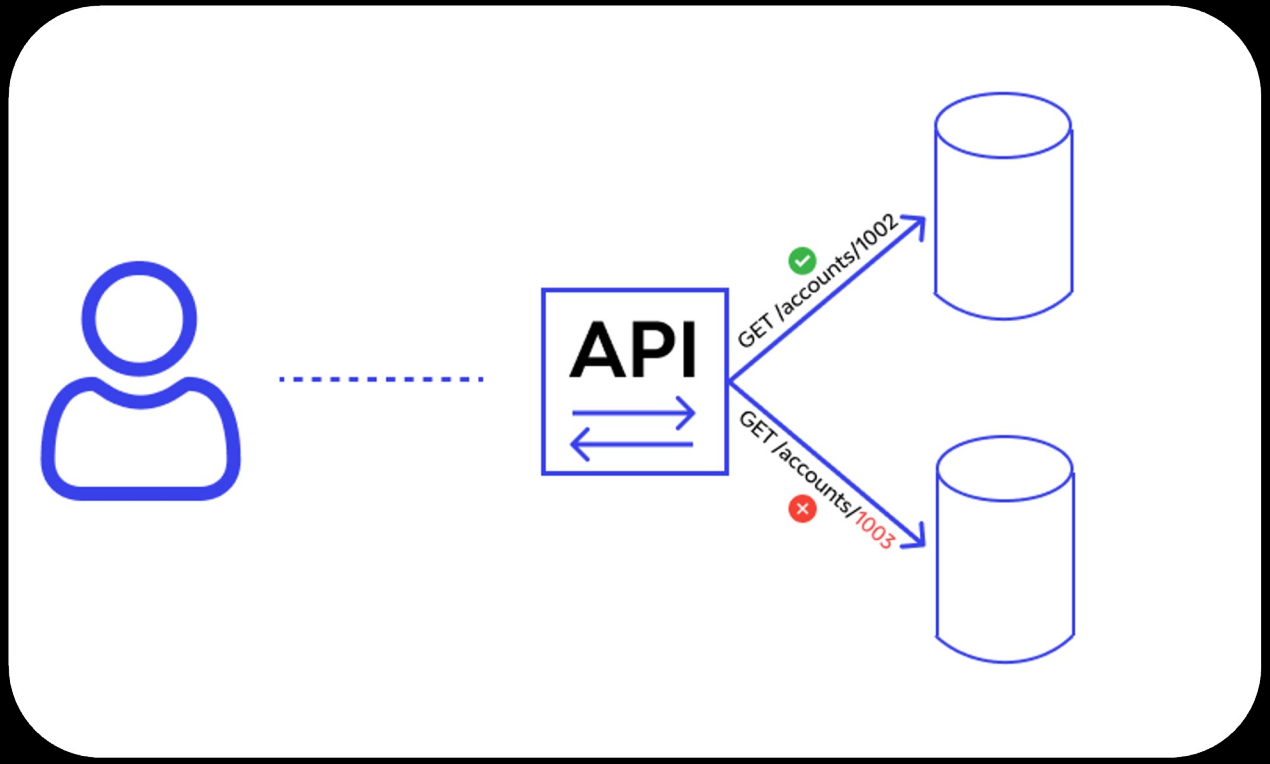
# Что такое OWASP?

OWASP (Open Worldwide Application Security Project) — это открытое сообщество, которое создает свободно доступные материалы по безопасности приложений.



# API1:2023 — Broken Object Level Authorization

Атака возникает, когда API не проверяет, имеет ли пользователь право на доступ к определённому объекту.



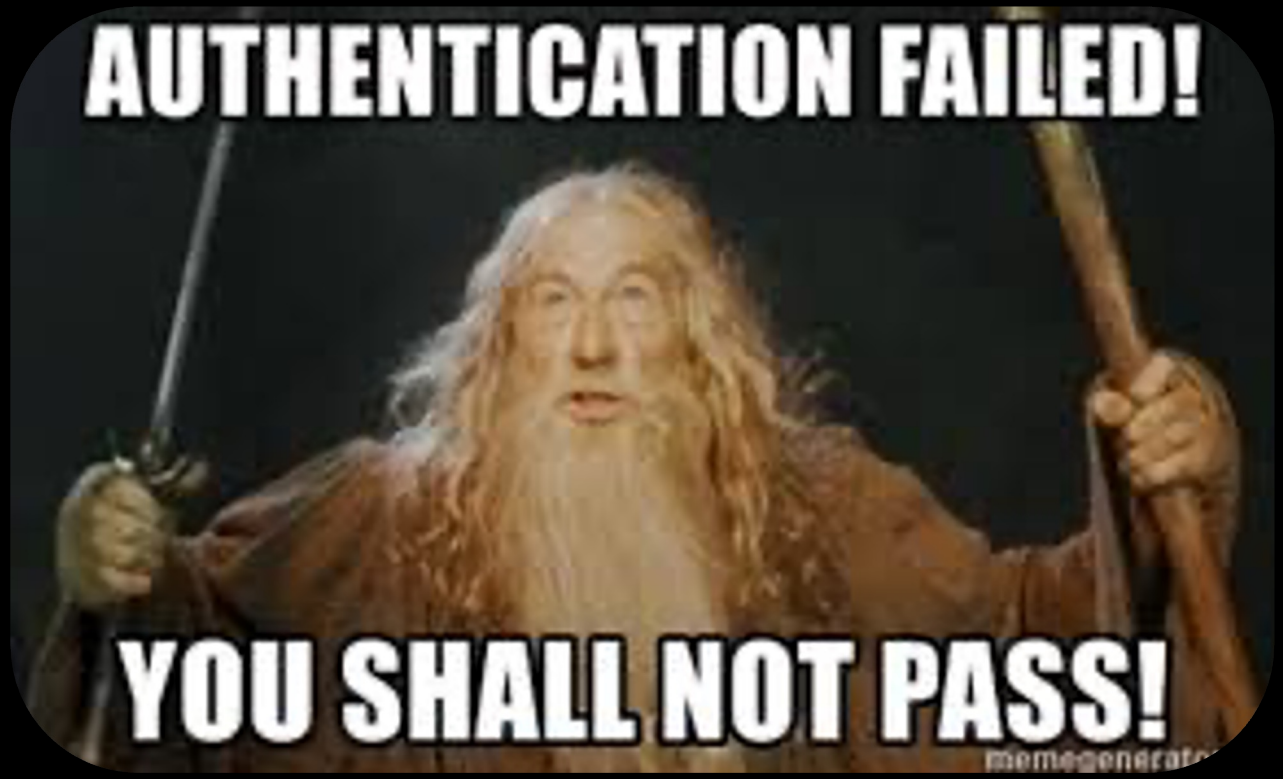
# API1:2023 — Broken Object Level Authorization

🎯 **Best practices для эксперта по безопасности:**

- Проверка, реализована ли авторизация **на уровне объекта**.
- Требовать использование **идентификаторов из токена**, а не из URL.
- Анализировать маршруты на возможность **предсказуемого доступа** (/user/42).
- Убедиться в наличии unit/integration тестов на авторизацию.

# API2:2023 — Broken Authentication

Неправильная реализация аутентификации, позволяющая злоумышленнику получить доступ к API без валидных учетных данных или выдать себя за другого пользователя.



# API2:2023 — Broken Authentication

🎯 **Best practices для эксперта по безопасности:**

- Проверить, какие механизмы аутентификации используются (OAuth2, OpenID).
- Убедиться, что токены подписаны, ограничены по времени и отзываемы.
- Оценить устойчивость к брутфорсу и enumeration.
- Проверить поведение при утрате/истечении токена.

# API3:2023 — Broken Object Property Level Authorization

Пользователь получает доступ к свойствам объекта, которые не должен видеть или редактировать, потому что отсутствует гранулярная проверка прав на уровне полей объекта.

```
{
  "userId": 123,
  "email": "attacker@evil.com",
  "isAdmin": true
}
```

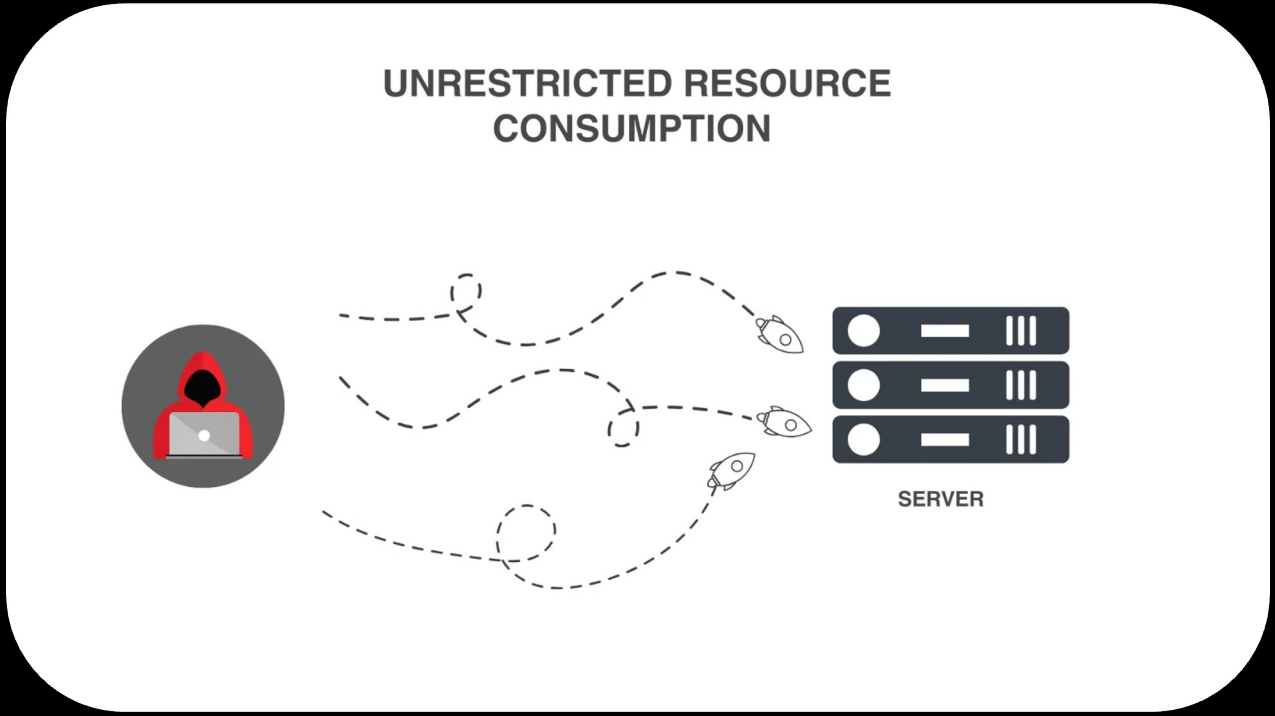
# API3:2023 — Broken Object Property Level Authorization

🎯 Best practices для эксперта по безопасности:

- Оценить, как контролируется доступ к отдельным полям объектов.
- Проверить, фильтруются ли чувствительные поля до сериализации.
- Убедиться в использовании white-list подхода при обновлении данных.
- Проанализировать бизнес-логику на предмет преобразования ролей/привилегий.

# API4:2023 — Unrestricted Resource Consumption

API не ограничивает количество или объём запрашиваемых ресурсов — это позволяет злоумышленнику перегружать сервер, истощать ресурсы или получать избыточные данные, что приводит к DoS или деградации производительности.



# API4:2023 — Unrestricted Resource Consumption

🎯 Best practices для эксперта по безопасности:

- Проверить, заданы ли лимиты по ресурсам (время, память, размер запроса).
- Есть ли ограничения по пагинации, глубине рекурсии и т. п.
- Убедиться, что API не позволяет массовые операции без контроля.
- Оценить механизмы rate limiting и защиты от DoS.

# API5:2023 — Broken Function Level Authorization

API не проверяет права доступа на уровне действий, позволяя пользователю выполнять функции, которые ему недоступны, например

```
POST /api/admin/create-user HTTP/1.1
Host: vulnerable-api.com
Authorization: Bearer eyJhbGciOi...

Content-Type: application/json

{
  "username": "evil_user",
  "password": "P@ssw0rd!",
  "role": "admin"
}
```

# API5:2023 — Broken Function Level Authorization

🎯 Best practices для эксперта по безопасности:

- Проверить, разграничены ли endpoint’ы по ролям/уровням доступа.
- Убедиться, что запрещён вызов "административных" функций обычным пользователям.
- Провести role/permission mapping для всех операций.
- Оценить работу контроллеров/гейткиперов на сервере.

# API6:2023 - Unrestricted Access to Sensitive Business Flows

API не ограничивает доступ к важным бизнес-функциям (например, покупкам, скидкам, бронированиям), злоумышленник может злоупотреблять этими процессами

```
POST /api/checkout/apply-coupon
Authorization: Bearer eyJhbGci...
Content-Type: application/json

{
  "coupon": "SPRING2025"
}
```

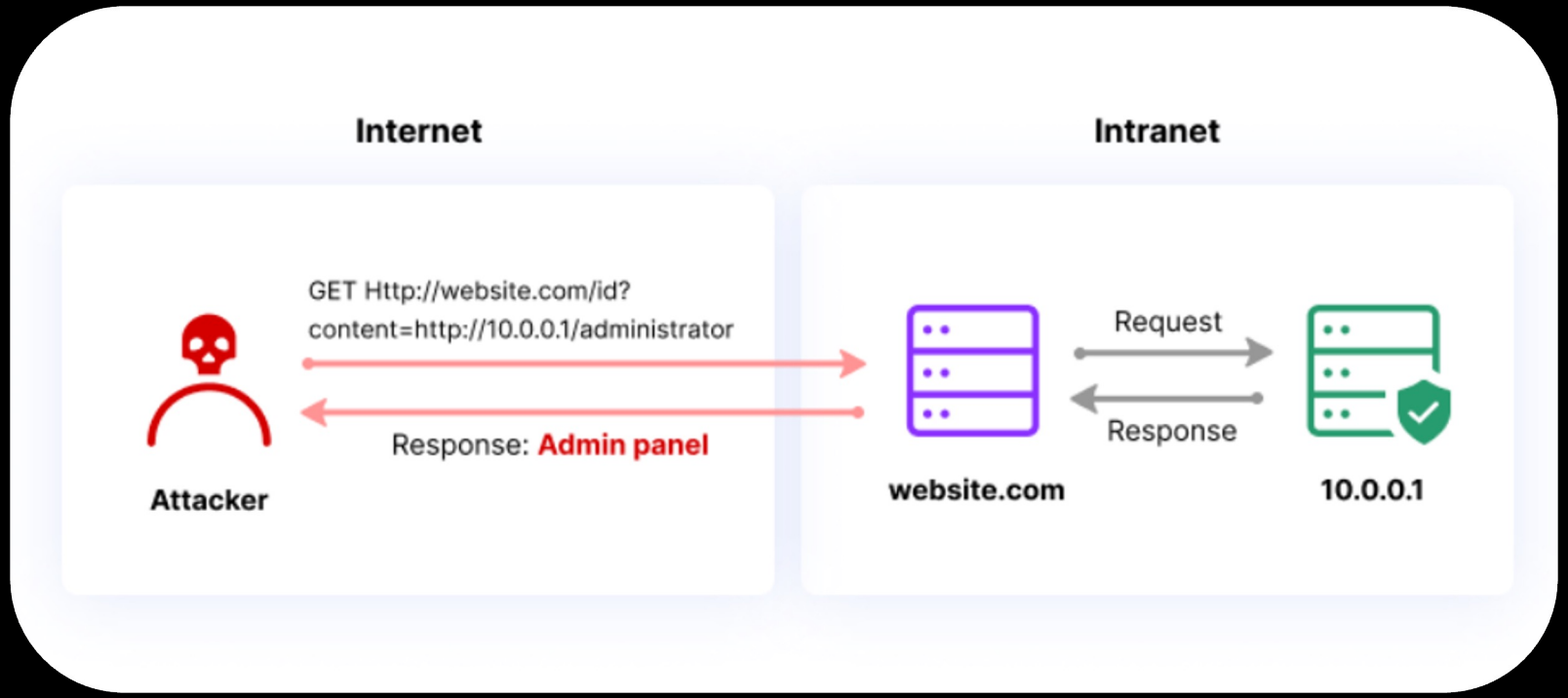
# API6:2023 - Unrestricted Access to Sensitive Business Flows

🎯 Best practices для эксперта по безопасности:

- Проанализировать, можно ли автоматизировать важные действия .
- Проверить наличие CAPTCHA, антибот-защиты.
- Проверить, ограничено ли число операций в единицу времени.
- Убедиться, что нет обхода клиентской логики (бизнес-проверки только на фронте).

# API7:2023 — Server Side Request Forgery

API позволяет пользователю задавать URL-адреса или иные внешние ресурсы без должной фильтрации, злоумышленник может заставить сервер отправлять запросы от своего имени



# API7:2023 — Server Side Request Forgery

🎯 Best practices для эксперта по безопасности:

- Проверить, отправляются ли сервером исходящие запросы по входным данным.
- Убедиться в наличии валидаторов URL (домен, IP, порт, протокол).
- Оценить изоляцию контейнеров, сетевых зон.
- Проверить, можно ли обходить защиту через редиректы/DNS-рекурсию.

# API8:2023 — Security Misconfiguration

API настроены небезопасно. Это может быть ненастроенная авторизация, включённый дебаг, открытые эндпоинты, слишком подробные ошибки и т.д.

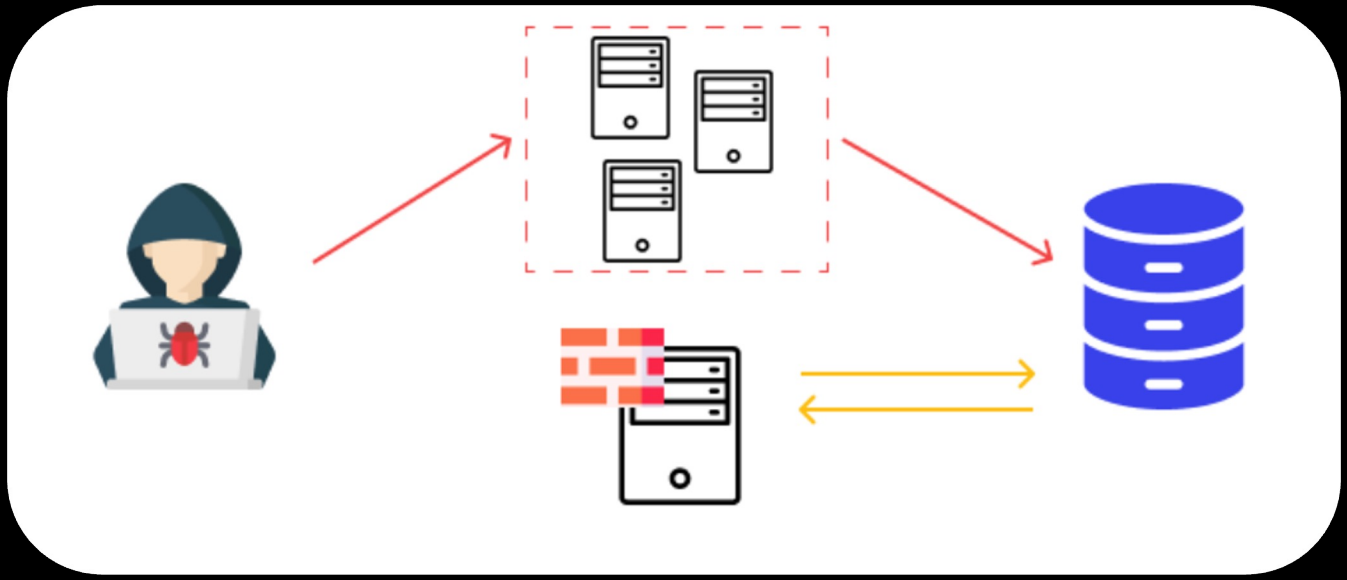
# API8:2023 — Security Misconfiguration

🎯 **Best practices для эксперта по безопасности:**

- Убедиться, что отключены отладочные интерфейсы на проде.
- Проверить настройки CORS, CSP, заголовки безопасности.
- Проверить наличие открытых портов/эндпоинтов, не используемых в проде.
- Оценить, какие конфиги/секреты попадают в деплой.

# API9:2023 — Improper Inventory Management

Когда организация не ведёт учёт всех своих API или не отслеживает их версионирование, это приводит к использованию устаревших, неподдерживаемых или внутренних API в продакшене.



# API9:2023 — Improper Inventory Management

🎯 Best practices для эксперта по безопасности:

- Проверить наличие устаревших/дублирующих API в проде.
- Убедиться в отслеживании версий и удалении legacy endpoint’ов.
- Требовать централизованную документацию API-интерфейсов.
- Убедиться, что нет "серых" API, используемых без документирования.

# API10:2023 — Unsafe Consumption of APIs

Приложение доверяет сторонним или внутренним API без должной проверки.

```
// Получаем цену из внешнего API и используем её напрямую
const userPrice = await axios.get("https://thirdparty.com/api/price?id=123");
processPayment(userPrice.data);
```

# API10:2023 — Unsafe Consumption of APIs

🎯 **Best practices для эксперта по безопасности:**

- Проверить, как обрабатываются входные данные от сторонних API.
- Убедиться в наличии валидации, санитации, фильтрации.
- Проверить fallback/обработку ошибок внешнего API.
- Оценить риск инъекций, XSS, подмены данных через партнёрские системы.

# Немножко подытожим



- 🔓 Недостаточная аутентификация и авторизация
- 🫧 Отсутствие фильтров на уровне объектов и функций
- ⚙️ Перегрузка ресурсов и неправильная конфигурация
- 🕵️ Неучтённые эндпоинты и бизнес-логика
- 🌍 Нет проверки данных от сторонних API и клиентов

# Best Practices

# Пример запроса и ответа ❌

```
POST /login
Host: shop.example.com
Content-Type: application/json

{
  "username": "admin",
  "password": "admin"
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json
Set-Cookie: session=abc123; HttpOnly

{
  "message": "Welcome back, admin!",
  "debug": {
    "auth_step": "password_ok",
    "user_id": 1
  }
}
```

# Пример запроса и ответа ❌

POST /login
Host: shop.example.com
Content-Type: application/json

```
{
 "username": "admin",
 "password": "admin"
}
```

HTTP/1.1 200 OK
Content-Type: application/json
Set-Cookie: session=abc123; HttpOnly

```
{
 "message": "Welcome back, admin!",
 "debug": {
  "auth_step": "password_ok",
  "user_id": 1
 }
}
```

**API2:2023 — Broken Authentication**
**API4:2023 — Unrestricted Resource Consumption**
**API6:2023 — Unrestricted Access to Sensitive Business Flows**
**API8:2023 — Security Misconfiguration**

# Пример запроса и ответа ✅

```
POST /api/v1/auth/login
Host: shop.example.com
Content-Type: application/json

{
  "email": "user@example.com",
  "password": "C0rrectHors3BatterySt4ple"
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "expiresIn": 3600
}
```

# Пример запроса и ответа 2 ❌

```
POST /add-to-cart HTTP/1.1
Host: shop.example.com
Content-Type: application/x-www-form-urlencoded
Cookie: session=abc123

cartId=999&productId=666&quantity=1000&description="<h1>Best Product!</h1>
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{"cart": {
        "id": 999,
        "items": [{
                "productId": 666,
        "description": "<h1>Best Product!</h1>"
                "quantity": 1000}]},
}
```

# Пример запроса и ответа 2 ❌

POST /add-to-cart HTTP/1.1
Host: shop.example.com
Content-Type: application/x-www-form-urlencoded
Cookie: session=abc123

cartId=999&productId=666&quantity=1000&description="<h1>Best Product!</h1>

HTTP/1.1 200 OK
Content-Type: application/json

```
{"cart": {
        "id": 999,
        "items": [{
                "productId": 666,
        "description": "<h1>Best Product!</h1>"
                "quantity": 1000}]},
}
```

**API1:2023 — Broken Object Level Authorization**
**API4:2023 — Unrestricted Resource Consumption**
**API6:2023 — Unrestricted Access to Sensitive Business Flows**

# Пример запроса и ответа 2 ✅

```
POST /api/v1/cart/items HTTP/1.1
Host: shop.example.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR...

Content-Type: application/json

{
  "productId": "987654321",
  "quantity": 2
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "message": "Item added to cart",
  "items": [
    {
      "productId": "abc123",
      "quantity": 2
    }
  ]
}
```

# Пример запроса и ответа 3 ❌

```
POST /apply-coupon HTTP/1.1
Host: shop.example.com
Content-Type: application/json
Authorization: Bearer fake-token

{
  "userId": 7,
  "coupon": “TEST-CODE"
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{"status": "Coupon applied successfully",
  "discount": 100,
  "user": {
    "id": 7,
    "role": "admin"
  },
}
```

# Пример запроса и ответа 3 ❌

POST /apply-coupon HTTP/1.1
Host: shop.example.com
Content-Type: application/json
Authorization: Bearer fake-token

```
{
  "userId": 7,
  "coupon": “TEST-CODE"
}
```

HTTP/1.1 200 OK
Content-Type: application/json

```
{"status": "Coupon applied successfully",
  "discount": 100,
  "user": {
    "id": 7,
    "role": "admin"
  },
}
```

**API1:2023 — Broken Object Level Authorization**
**API4:2023 — Unrestricted Resource Consumption**
**API6:2023 — Unrestricted Access to Sensitive Business Flows**
**API8:2023 — Security Misconfiguration**

# Пример запроса и ответа 3 ✅

```
POST /api/v1/cart/discounts HTTP/1.1
Host: shop.example.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR...

Content-Type: application/json

{
  "couponCode": "SPRING24"
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "message": "Coupon applied",
  "discount": {
    "type": "percentage",
    "value": 24
  },
  "newTotal": 1350
}
```

# Пример запроса 4 ❌

```
POST /purchase HTTP/1.1
Host: shop.example.com
Content-Type: application/json
Authorization: Bearer fake-token

{
  "cartId": 123,
  "price": 9.99,
  "promoCode": “TEST-CODE“,
  "address": "Moscow Lenin st. 1321"
}
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "success",
  "total_paid": 9.99,
  "cartId": 123,
  "items": [
    {
      "name": "Premium headphones",
      "price": 149.99
    }
  ],
  "payment": {
      "transactionId": "txn_123456",
      "card_last4": "1111"
  }
}
```

# Пример запроса 4 ❌

POST /purchase HTTP/1.1
Host: shop.example.com
Content-Type: application/json
Authorization: Bearer fake-token

```
{
  "cartId": 123,
  "price": 9.99,
  "promoCode": “TEST-CODE“,
  "address": "Moscow Lenin st. 1321"
}
```

**API1:2023 — Broken Object Level Authorization**
**API4:2023 — Unrestricted Resource Consumption**
**API6:2023 — Unrestricted Access to Sensitive Business Flows**
**API8:2023 — Security Misconfiguration**

HTTP/1.1 200 OK
Content-Type: application/json

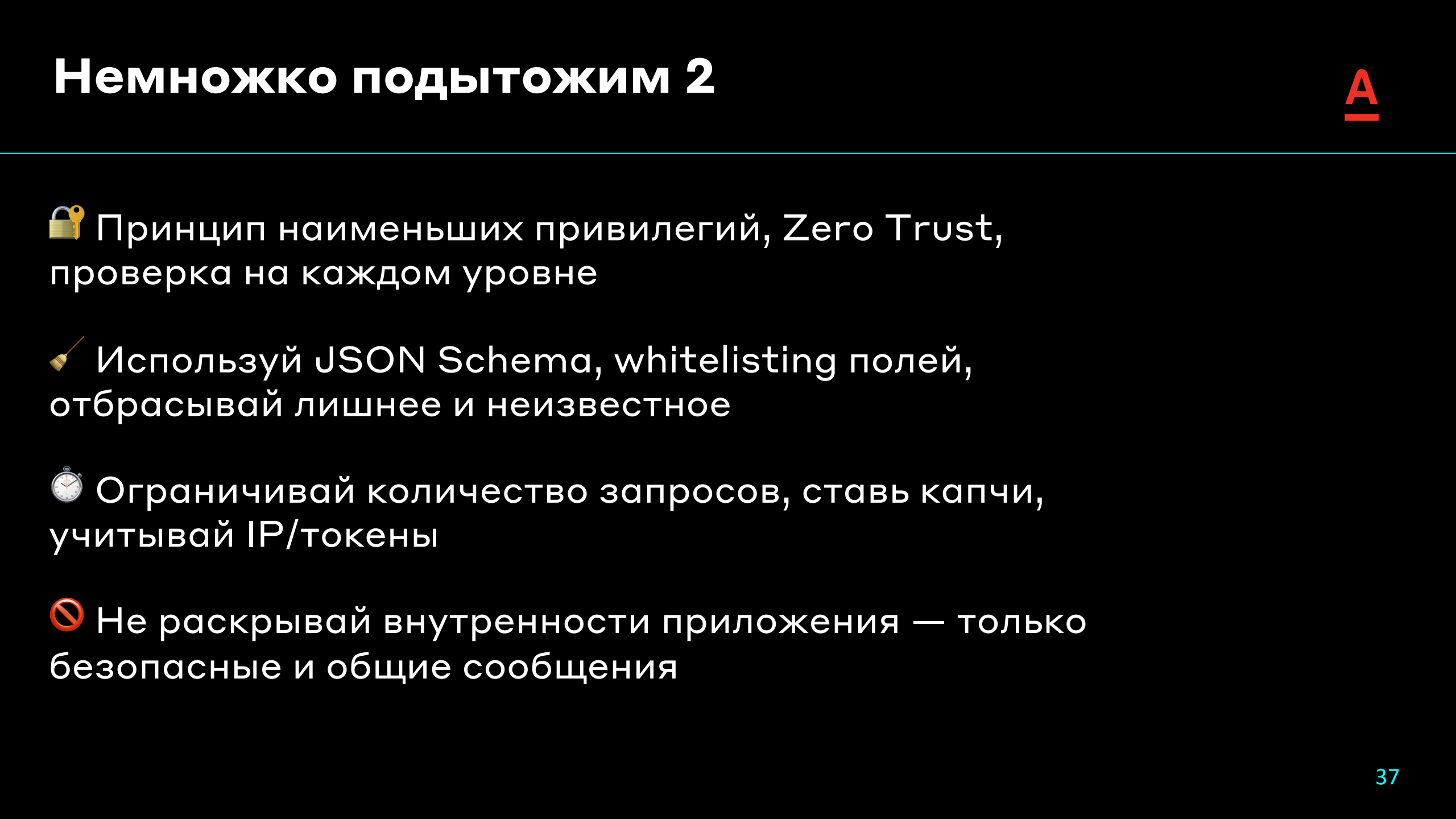
```
{
  "status": "success",
  "total_paid": 9.99,
  "cartId": 123,
  "items": [
    {
      "name": "Premium headphones",
      "price": 149.99
    }
  ],
  "payment": {
    "transactionId": "txn_123456",
    "card_last4": "1111"
  }
}
```

# Пример запроса 4 ✅

```
POST /api/v1/checkout HTTP/1.1
Host: shop.example.com
Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR...

Content-Type: application/json

{
  "paymentMethodId": "pm_1ABC...",
  "shippingAddress": {
    "line1": "123 Main St",
    "city": "Springfield",
    "zip": "12345"
  }
}
```

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "message": "Order placed successfully",
  "orderId": "ord_8921cd",
  "estimatedDelivery": "2025-04-18"
}
```

# Немножко подытожим 2



🔐 Принцип наименьших привилегий, Zero Trust, проверка на каждом уровне

🧹 Используй JSON Schema, whitelisting полей, отбрасывай лишнее и неизвестное

⏱️ Ограничивай количество запросов, ставь капчи, учитывай IP/токены

🚫 Не раскрывай внутренности приложения — только безопасные и общие сообщения

# Время вопросов

# Обратная связь