Харденинг и инвентаризация

Алексей Морозов

Руководитель прикладной безопасности ecom.tech

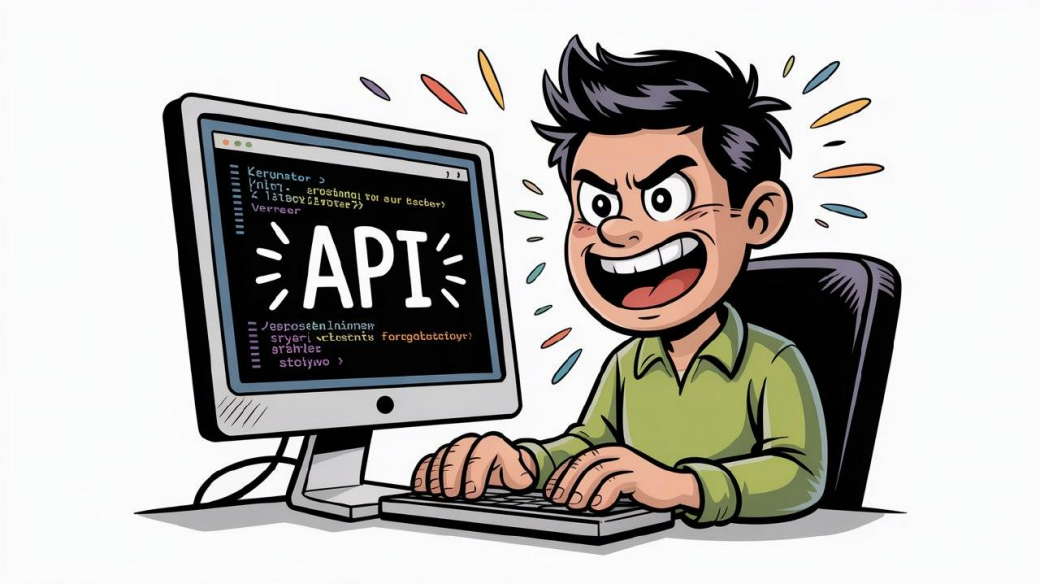
ecom.tech

Обо мне

1) Четырехкратный чемпион Standoff.
2) 12+ лет опыта.
3) Спикер (более 100 конференций).
4) Основатель компании Ghack.
5) Имею CVE и публикации.
6) Серты: OSWE, OSCP, CEH.

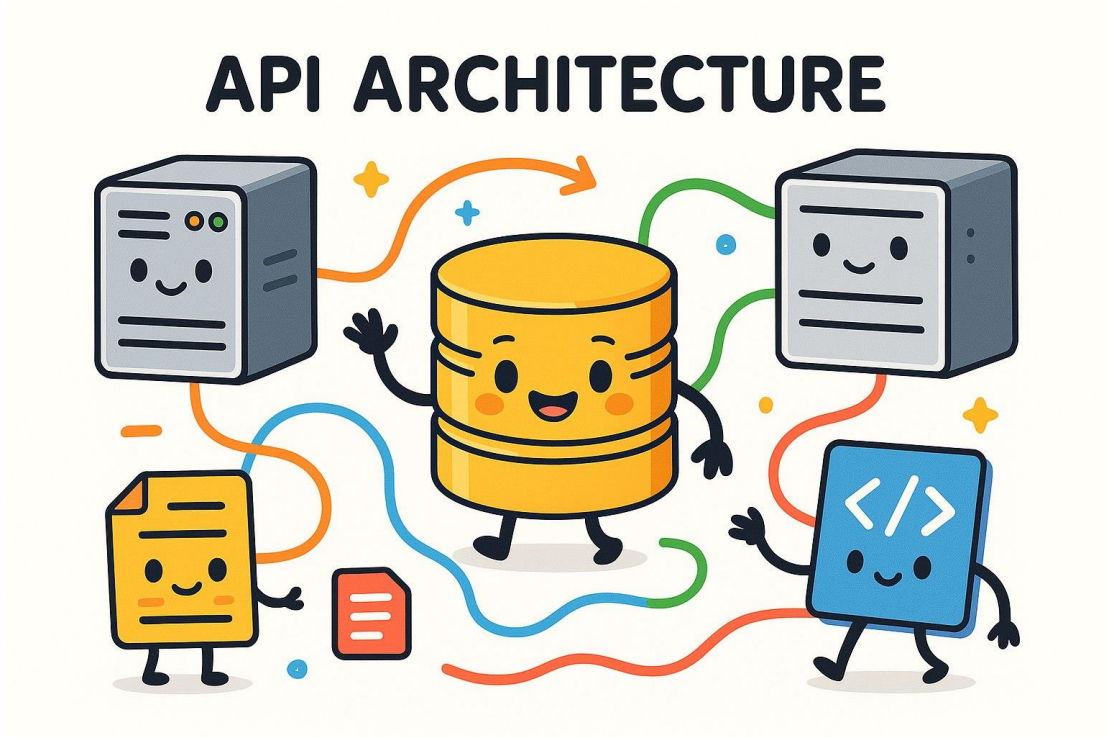
Проблемы безопасности API

- Мы про них просто не знаем.
- Отсутствует ролевая модель.
- IDOR’s.
- Авторизация либо слабая, либо отсутствует.
- Rate-Limit’s.



Архитектура инвентаризации

- Публикуем прямые методы.
- **SWAGGER'ы** в **SSOT**.
- Инвентаризация активов в **BackStage** уровня критичности.
- Спускаем требования к API.
- Регулярный **мониторинг** логов.
- **Классификация** методов.



А если сваггеров нет? (pipeline)

(01) Определить язык и фреймворк проекта.

(02) Написать парсер для определенной модели кода и обучить dataset.

- Endpoints (пути, методы).
- Параметры запросов и ответов.
- Описания (докстринги, комментарии).
- Модели данных (DTO/serializers).

(03) (Собрать разделы [paths], [components/schemas] и т.д. в формате YAML/JSON).

(04) Подключить ИИ агента для генеративного ответа (**НЕ ИСПОЛЬЗОВАТЬ ДЛЯ КЛАССИФИКАЦИИ**).

Пример кода

```
@app.route('/users', methods=['GET'])
def get_users():
    return jsonify(users)

@app.route('/tasks', methods=['GET'])
def get_tasks():
    return jsonify(tasks)

@app.route('/users/<int:user_id>/tasks', methods=['GET'])
def get_user_tasks(user_id):
    user_tasks = [task for task in tasks if task['user_id'] == user_id]
    return jsonify(user_tasks)
```

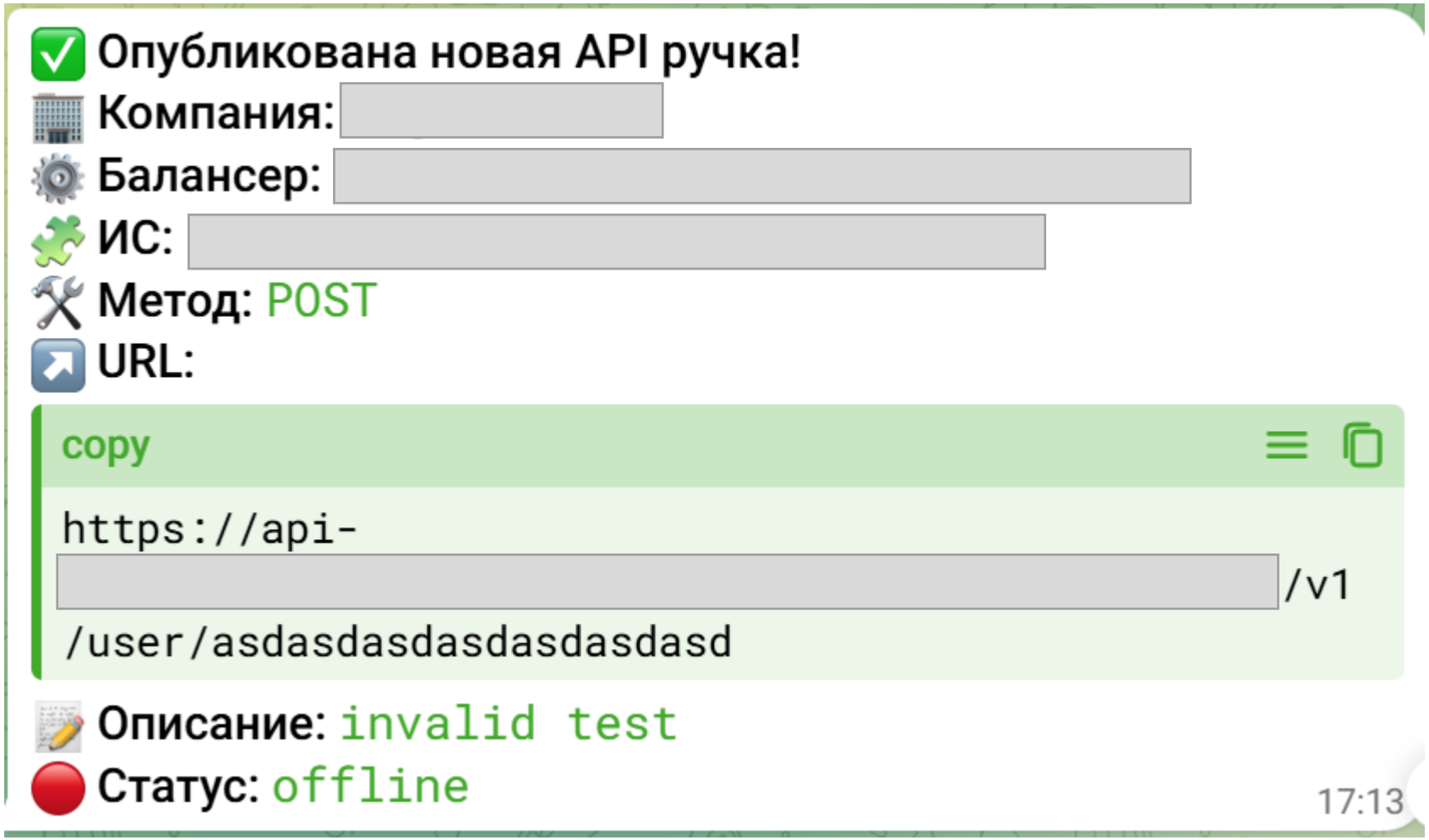
Пример вывода

```
"/users/{user_id}/tasks": {
      "get": {
        "summary": "Получение задач пользователя",
        "description": "Возвращает все задачи, связанные
с пользователем по его идентификатору user_id.",
        "parameters": [
          {
            "name": "user_id",
            "in": "path",
            "required": true,
            "schema": {
              "type": "integer"
            },
            "description": "Уникальный идентификатор пользователя"
          }
        ],
```

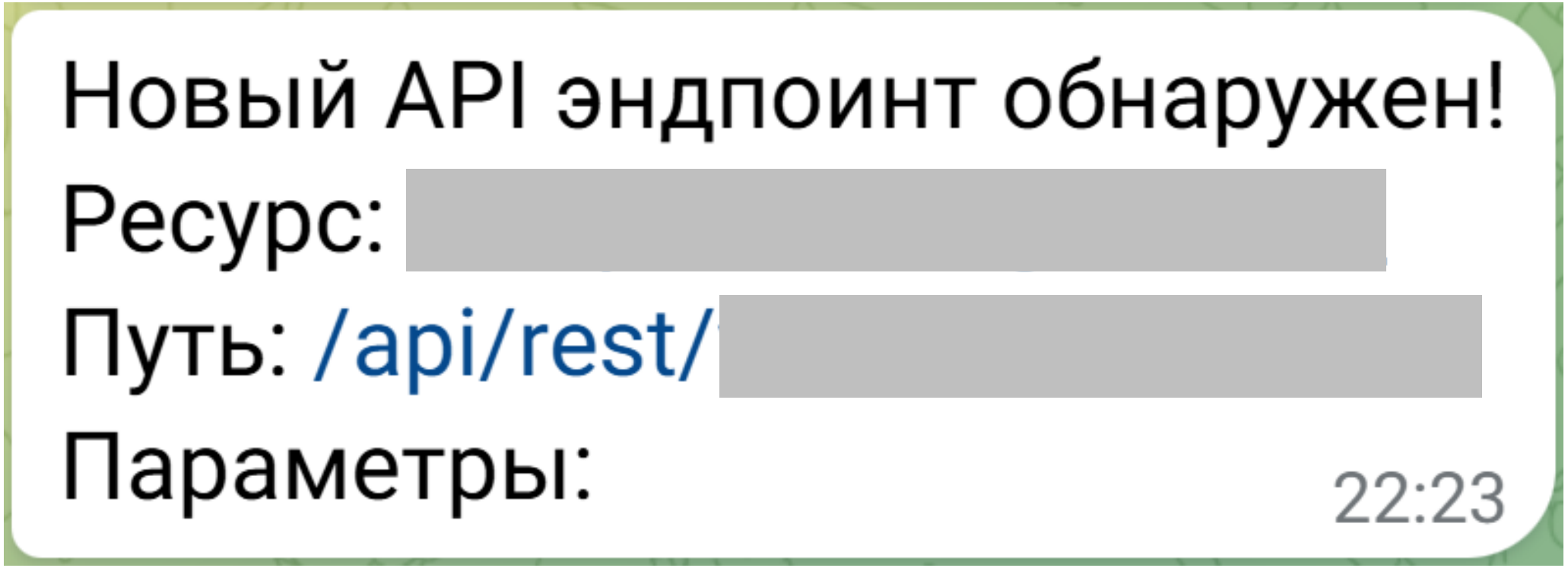
Мониторинг

- Забирать регулярно логи (раз в час).
- Сверять с реестром публикаций. <- **Контроль тут**
- Дедуплицировать.
- Нотифицировать.

Мониторинг


✅ Опубликована новая API ручка!
Компания:
Балансер:
ИС:
Метод: POST
URL:
copy
https://api-
/v1
/user/asdasdasdasdasdasdasd
Описание: invalid test
Статус: offline
17:13

Мониторинг


Новый API эндпоинт обнаружен!
Ресурс:
Путь: /api/rest/
Параметры:
22:23

Фабрика для API-Gateway

- Запросы проходят авторизацию на уровне Gateway.
- OAuth 2.0 / OIDC (валидация access token, подписи, времени жизни).
- Для каждого метода требуется ролевка.
- Все идентификаторы в формате UUID (RBAC/ABAC).
- Форматы: Rest, GraphQL, SOAP.

Паттерн матч ролевки

```
routes:
  - path: /api/users/list
    methods: [GET]
    allowed_roles: [admin, manager]
  - path: /api/profile
    methods: [GET, POST]
    allowed_roles: [admin, user, manager]
  - path: /api/report
    methods: [GET]
    allowed_roles: [manager]

authorization:
  type: oauth
  role_claim: "role"
```

Паттерн матч ролевки

1. Gateway получает запрос от клиента.
2. Извлекает токен.
3. Декодирует токен, определяя роль пользователя (`role`).
4. Сравнивает роль с разрешёнными для эндпоинта ролями.
5. Если роль не разрешена — возвращает 403 Forbidden без проксирования запроса к API.

Линтеры

```
susPatterns := []*regexp.Regexp{
  regexp.MustCompile(\(Query(s*".+"s*))\),
  regexp.MustCompile(\Varss*(.*?)s*[s*".+"s*]\),
  regexp.MustCompile(\FormValue(s*".+"s*)\),
  regexp.MustCompile(\Argss*[s*".+"s*]\),
  regexp.MustCompile(\(json.Unmarshal|Decode)(.*[Ii][Dd].*\),
 }
```

Уязвимый код

```
func (r *queryResolver) GetUser(...) (*User, error) {
    id := args["id"].(int) return db.FindUserByID(id)
}
```

Уязвимый код

```
[IDOR] Подозрительные данные: id := args["id"].(int)
Используйте вместо числовых идентификаторов формат UUID!
```

Классификация API. Структура

- Критичность актива.
- Наличие критичных ролевок.
- Эвристический анализ по названию методов и параметров.
- Приватный/Публичный.

Дамасем

Собираем dataset:

- Имя
- Параметры
- Описание
- Критичность Актива
- Ролевки
- Доступность
- Rate-Limit
- IsAuth
- Score

Feature Engineering

- **Название ручки, описание и параметры:**
  - bag-of-words
  - n-граммы
  - tf-idf
- **Актив:**
  - one-hot encoding
- **Приватный/Публичный**
  - бинарный флаг
- **Ролевка**
  - one-hot encoding (guest, user, admin и т.д)

Алгоритмы

- **Авторизация** — Бинарная классификация, Логистическая регрессия.
- **Разметка типа данных** — Случайный лес, Градиентный бустинг, KNN.
- **Rate-Limit** - Бинарная классификация, Логистическая регрессия.
- **Критичность** — Градиентный бустинг. Формула.

Пример

/user/create, name,email,password; MC; публичный; guest

Выход:

- Авторизация: Не требуется.
- Тип данных: персональные.
- Rate-limit: Требуется.
- Критичность: 5 (Medium).

Пример

/stats/get, <none>; BO; публичный; guest

Выход:

- Авторизация: Не требуется.
- Тип данных: общедоступные.
- Rate-limit: Не требуется.
- Критичность: 2 (LOW).

Итоги

1) Безопасность API начинается с создания правил игры.
2) Нельзя защитить, то чего не видишь.
3) За кадром остались еще дасты, фаззеры и прочие клевые штуки.

Бонус:
Не используйте большие LLM для задач классификации.

Остаёмся на связи

Алексей Морозов

ecom.tech

@SooLFaa