Безопасность backend приложений

Java Junior, 2021

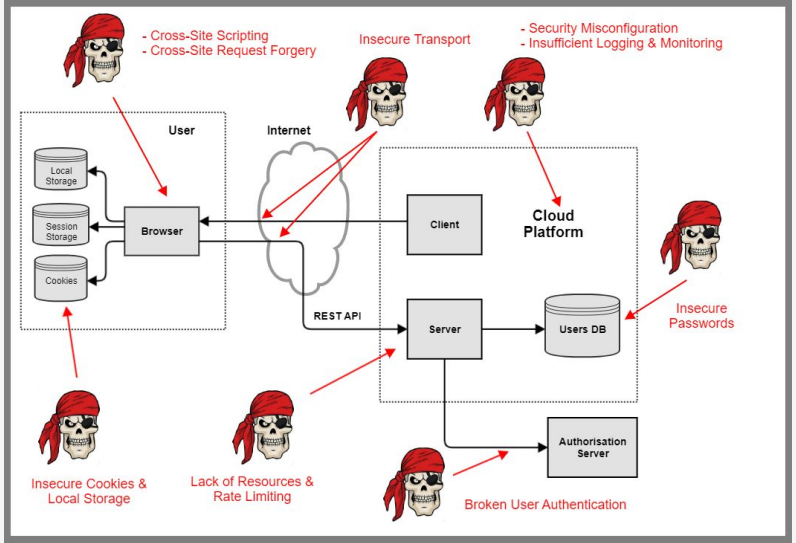
Введение

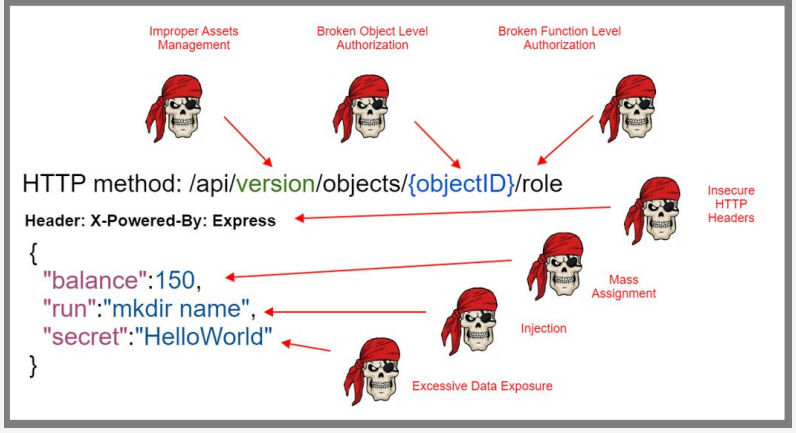
- Достаточно ли сделать работающий сервис?
- CDN Akamai: 83% всего web-траффика приходится на вызовы API
- Ростелеком: более 70% вэб-приложений содержат критические уязвимости

- Зачем и от кого нужно защищаться?
- Почему 100% безопасности не существует?
- Безопасность – ответственность каждого участника команды

Последние большие факапы

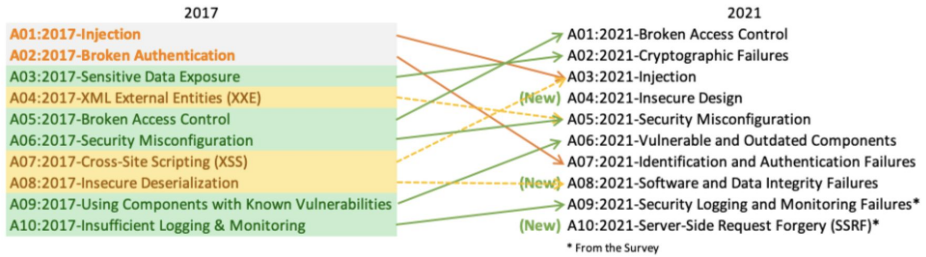
- Meltdown, spectre
- log4shell


- Cross-Site Scripting
- Cross-Site Request Forgery
Insecure Transport
- Security Misconfiguration
- Insufficient Logging & Monitoring
User
Internet
Local Storage
Session Storage
Cookies
Browser
Client
Cloud Platform
REST API
Server
Users DB
Insecure Passwords
Authorisation Server
Insecure Cookies & Local Storage
Lack of Resources & Rate Limiting
Broken User Authentication


Improper Assets Management
Broken Object Level Authorization
Broken Function Level Authorization
HTTP method: /api/version/objects/{objectID}/role
Header: X-Powered-By: Express
Insecure HTTP Headers
{
"balance":150,
"run":"mkdir name",
"secret":"HelloWorld"
}
Mass Assignment
Injection
Excessive Data Exposure

Помощь профессионалов

- OWASP (Open Web Application Security Project)
- API Security Top 10



A01 Broken Access Control

- 94% всех приложений
- Недостаточный контроль доступа к объектам
- Небезопасные прямые ссылки на объекты

A01 Broken Access Control

Получить одного пользователя с userID:

GET /users/{userID} Получаем данные другого пользователя

Удалить пользователя с userID :

DELETE /users/{userID} Удаляем другого пользователя

Некоторые понятия

- **Аутентификация**
- **Авторизация**

Некоторые понятия

- **Аутентификация** – проверка подлинности (ты тот, кем представляешься)

- **Авторизация**

Некоторые понятия

- **Аутентификация** – проверка подлинности (ты тот, кем представляешься)

- **Авторизация** – проверка прав на действия или ресурсы (тебе разрешено делать то, что ты пытаешься сделать)

A01: Как бороться

- Проверяем права доступа при каждом запросе
- Не доверяем ID, полученным от клиента. Проверяем что залогиненный пользователь имеет доступ к запрошенным объектам
- Внешние ID объектов должны быть сложными для подбора, например в виде UUID, а не простая последовательность 1, 2, 3

A01: Как бороться

Модели контроля доступа, Access Control Cheat Sheet

Role-Based Access Control (RBAC) – с помощью роли
Attribute-Based Access Control (ABAC) – на основании атрибутов пользователя
Organization-Based Access Control (OrBAC) – на основании организации
Discretionary Access Control (DAC) – по личности или группе
Mandatory Access Control (MAC) – по чувствительности данных
Permission Based Access Control – по разрешениям

A02 Cryptographic Failures

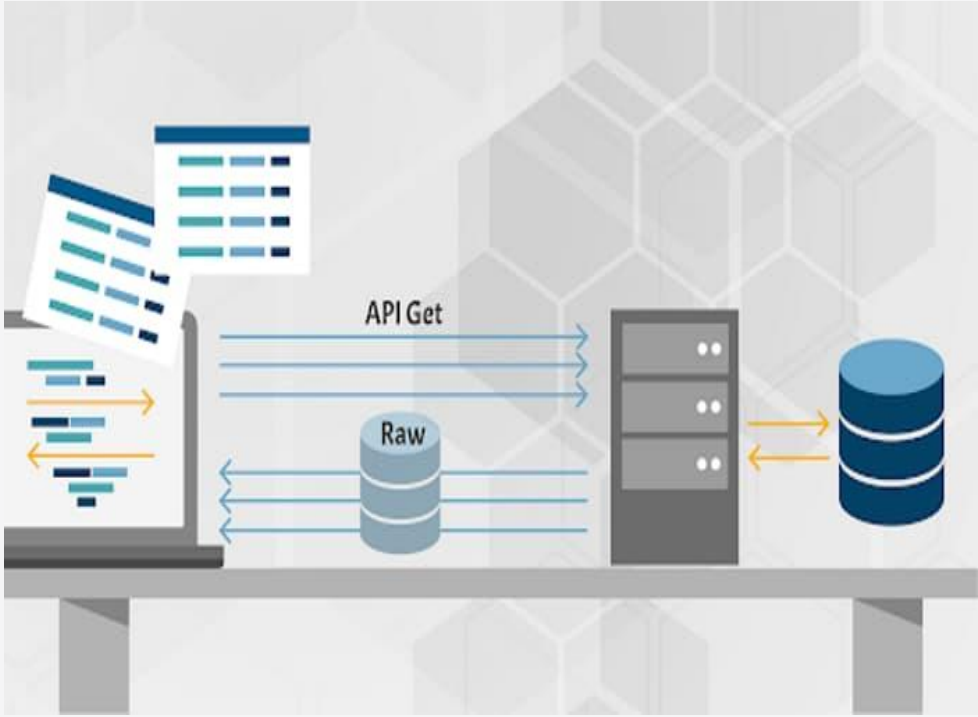
Получаем информацию о пользователе:

GET /users/1

Ответ может содержать избыточные данные:


```
{"userName":"Alex",
 "age":25,
 "secretAnswer":"HelloWorld"}
```


Если фильтрация данных на стороне клиента, можем получить лишние данные



A02: Cryptographic Failures

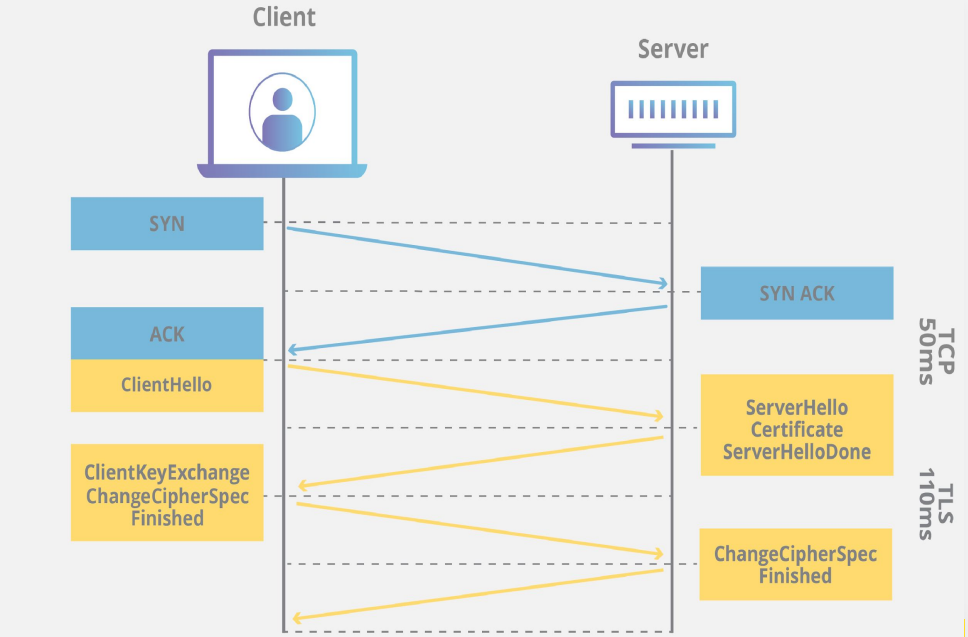
- Использование небезопасного протокола
- Утечка чувствительных данных
- Хранение паролей AS IS

A02: Как бороться

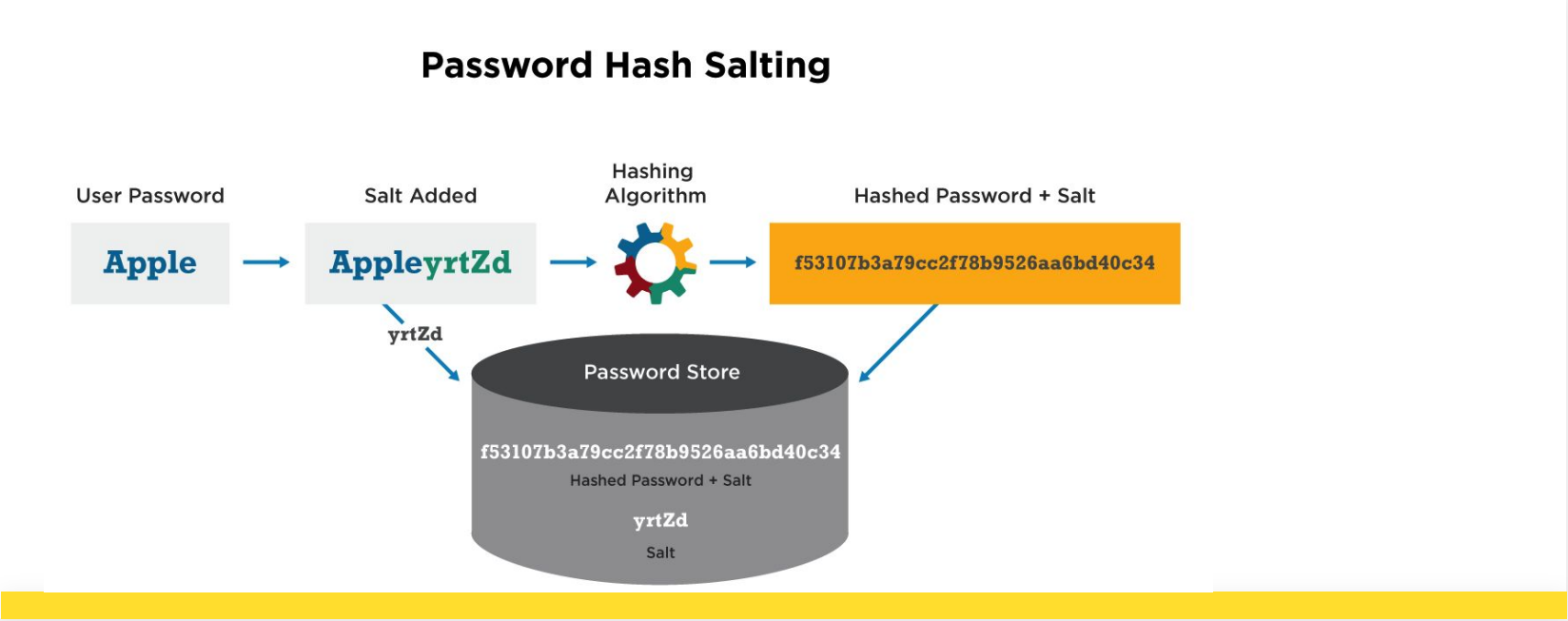
- Используем DTO для выдачи данных на фронт
- Добавляем в DTO только необходимые поля
- Реализуем фильтрацию данных на сервере
- TLS
- Для хранения паролей используем salted hashing functions

TLS handshake

- Согласование версии протокола
- Сервер в ответе отправляет свой сертификат и способ шифрования
- Клиент проверяет сертификат у CA, если сертификат валидный – отправляет случайную строку, закодированную публичным ключом из полученного сертификата
- Сервер декодирует строку своим приватным ключом
- Клиент и сервер генерируют **session key** (один и тот же), используя отправленную строку
- Handshake завершен, session key - ключ симметричного шифрования



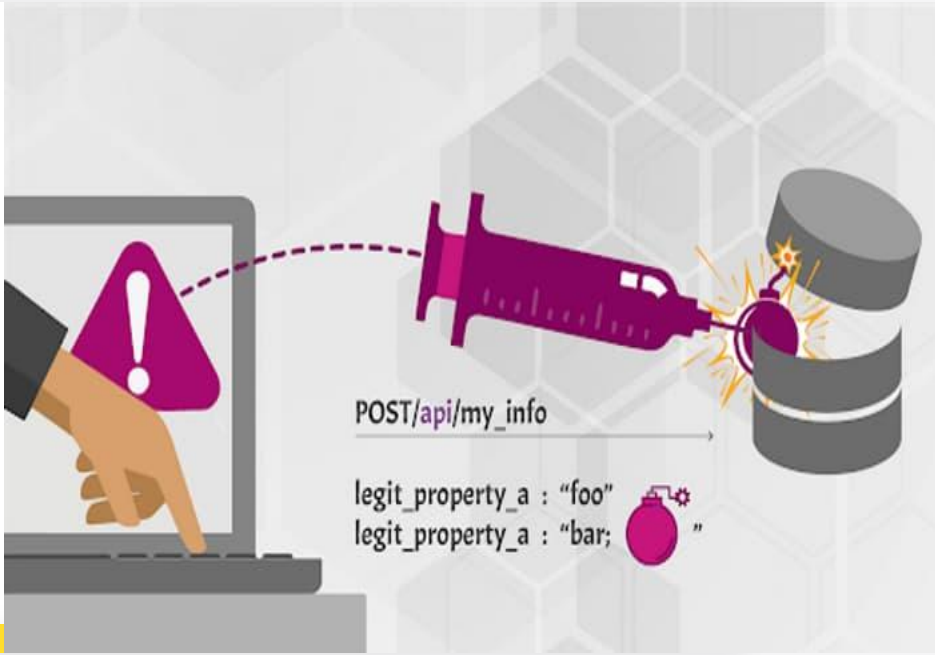
Salted hashing



A03 Injection

Ввод передается напрямую интерпретатору:

- SQL
- NoSQL
- LDAP
- Команды ОС
- XML Parser
- ORM



SQL injection

GET http://.../items/items.asp?itemid={idemId}

```
SELECT item_name, item_description FROM items
WHERE item_number = #{itemId}
```

GET http://.../items/items.asp?itemid=999

```
SELECT item_name, item_description FROM items
WHERE item_number = 999
```

SQL injection

GET http://.../items/items.asp?itemid=999 OR 1=1

```
SELECT item_name, item_description FROM items
WHERE item_number = 999 OR 1=1
```

GET http://.../items/items.asp?itemid=999; DROP TABLE users

```
SELECT item_name, item_description FROM items
WHERE item_number = 999; DROP TABLE users
```

SQL Injection – как бороться

Использовать параметризированные запросы:

- Prepared Statement
- ORM (последняя версия)
- Code Review
- Whitelist server side validations

OS command Injection

POST /run

```
{"mkdir":"name"}
```

Если сервер выполняет команды без проверки, то злоумышленник может послать следующую команду с большой вероятностью вывода сервера из строя:

POST /run

```
{"mkdir":"name && format C:/"}
```

OS command Injection – как бороться

- Не передавать пользовательский ввод в команды
- Делать проверки, фильтрацию, очистку данных

A04 Insecure design

- Восстановление пароля с помощью контрольного вопроса
- Логические ошибки (больше одного действия в “одни руки”)
- Нет защиты от ботов (ограничение на использование API)

A04: Как бороться

- Нанять AppSec специалистов ☺
- Унифицировать security дизайн паттерны
- Интеграционное и unit тестирование
- Ограничение потребляемых клиентом ресурсов - **Rate Limiting**

A05 Security Misconfiguration

- Используются настройки приложений по умолчанию
  (доступы, аккаунты и пароли, порты)
- В проде используются настройки для разработки и отладки
- Административные панели/endpoint’ы выставлены без ограничений
- Stacktrace’ы в сообщениях об ошибках

A05: Как бороться

- Целенаправленно ищем проблемы в конфигурации
- Отключаем ненужные функции
- Ограничиваем административный доступ
- Стандартизируем сообщения об ошибках
- Открываем только необходимые сетевые порты и пути на балансировщиках

A06 Vulnerable/Outdated Components

- Не обновляются системы/библиотеки
- Зависимости содержат уязвимости

A06: Как бороться

- Регулярно обновляем ОС и ПО
- Используем сканер уязвимостей в CI/CD

A07 Identification/Authentication Failures

- Незащищенные/внутренние API
- Ненадежный/нестандартный механизм аутентификации
- Отсутствие двухфакторной аутентификации
- Слабые ключи API/пароли
- Подверженность brute force атакам
- Пароли/ключи в URL
- Нет валидации access-token

A07 Identification/Authentication Failures

Виды аутентификации

A07 Identification/Authentication Failures

Виды аутентификации

- API key
- Basic Authentication
- Cookie-Based Authentication
- Token-Based Authentication

A07 Identification/Authentication Failures

API Key

```
POST https://my-api.com/my-service?key=API_KEY
```

A07 Identification/Authentication Failures

Basic auth

Используется HTTP заголовок 'Authorization’:

- Ключевое слово Basic
- Пробел
- base64 закодированная строка username:password

Authorization: "Basic dXNlcm5hbWU6cGFzc3dvcmQ="

Механизм поддерживается браузерами

A07 Identification/Authentication Failures

Cookie-Based Authentication

В ответ на запрос аутентификации сервер посылает заголовок Set-Cookie, который содержит имя и значение cookie, а также дополнительные атрибуты: expires, domain, path, secure, httponly

Set-Cookie: JSESSIONID=123456789; Path=/; HttpOnly

После этого клиент автоматически будет посылать заголовок Cookie при каждом запросе

Cookie: JSESSIONID=123456789

Требуется хранить сессии на сервере

A07 Identification/Authentication Failures

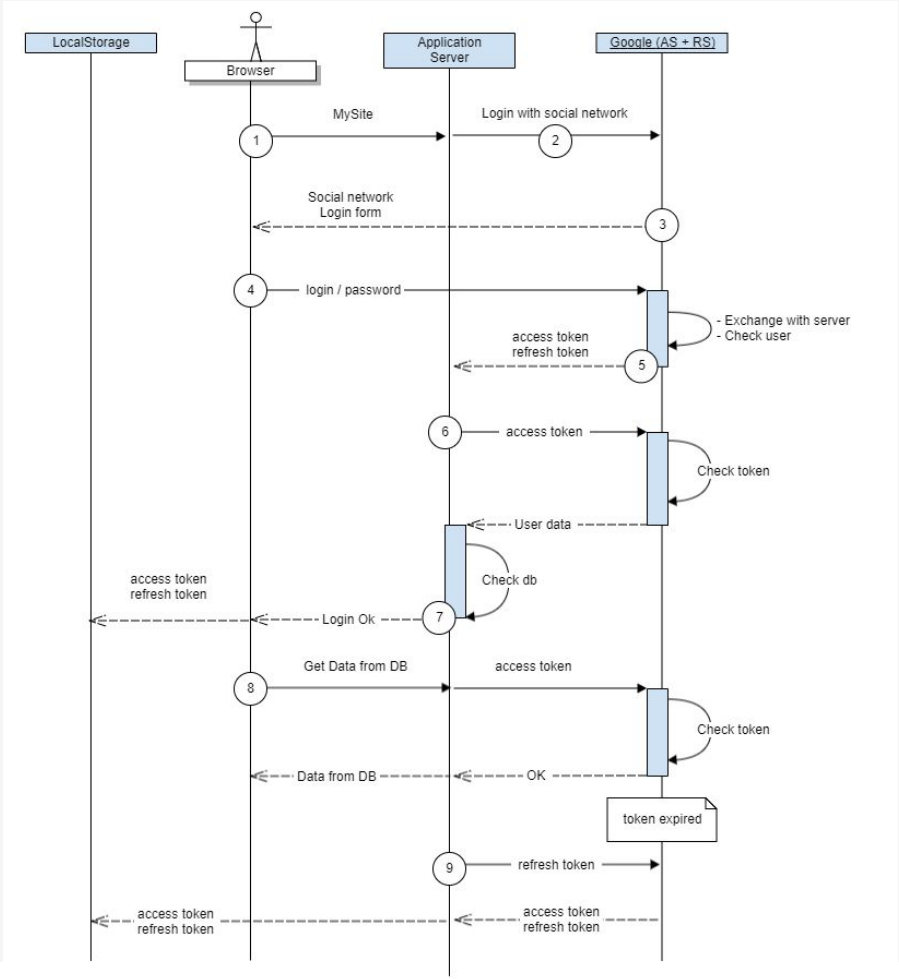
Token-Based Authentication

В ответ на запрос аутентификации сервер посылает заголовок Authorization, который содержит ключевое слово Bearer и токен

Authorization: Bearer eyJhbGciOiJSUzI1NiIsImtpZCI6IjI4Y

Может использоваться как часть OAuth 2.0 или OpenID Connect протоколов, так и сервер сам может сформировать токен

Для безопасного использования должен использоваться протокол, который обеспечивает шифрование данных, HTTP заголовков и URL, например HTTPS


LocalStorage
Browser
Application Server
Google (AS + RS)
MySite
1
Login with social network
2
Social network Login form
3
4
login / password
- Exchange with server
- Check user
access token
refresh token
5
6
access token
Check token
User data
Check db
access token
refresh token
Login Ok
7
Get Data from DB
8
access token
Check token
Data from DB
OK
token expired
9
refresh token
access token
refresh token
access token
refresh token

A07: Как бороться

- Мультифакторная аутентификация
- Проверять слабые пароли
- Защиту формы логина и восстановления от account enumeration attacks
- Capcha & Rate-limiting

A08 Software and Data Integrity Failures

- Автообновление версий ПО (OS, libraries, plugins etc)
- Уязвимость десериализации

A08 Software and Data Integrity Failures

```
public class BadThing implements Serializable {
    private static final long serialVersionUID = 0L;

    Object looselyDefinedThing;
    String methodName;

    private void readObject(ObjectInputStream ois) throws ClassNotFoundException, IOException {
        ois.defaultReadObject();
        try {
            Method method = looselyDefinedThing.getClass().getMethod(methodName);
            method.invoke(looselyDefinedThing);
        } catch (Exception e) {
            // handle error...
        }
    }

    // ...
}
```

```
BadThing badThing = new BadThing();
badThing.looselyDefinedThing = new MyCustomAttackObject();
badThing.methodName = "methodThatTriggersAttack";

Method method = looselyDefinedThing.getClass().getMethod(methodName);
method.invoke(methodName);
```

A08: Как бороться

- Использовать цифровые подписи и контрольные суммы, чтобы убедиться что источник ПО не скомпрометирован
- Code review новых зависимостей
- Пакетный менеджер может загружать только из проверенных репозиториев
- Нет автообновления стороннего ПО

A09 Insufficient Logging & Monitoring

Недостатки журналирования и мониторинга:

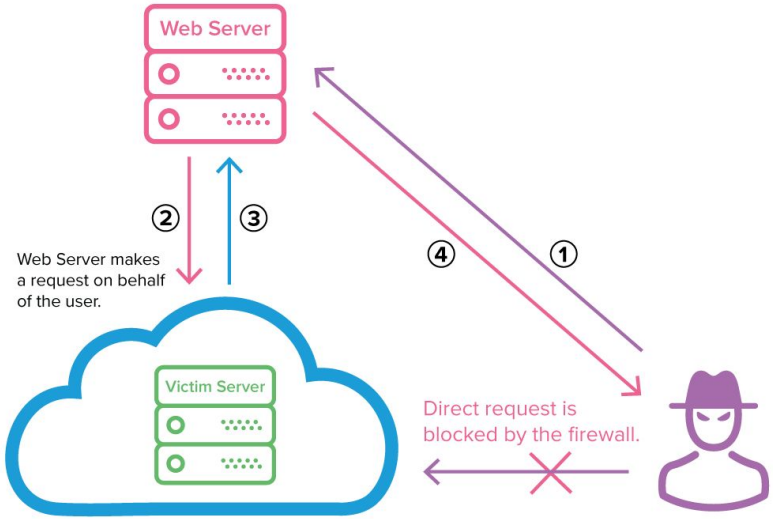
- Логов и метрик нет
- Логи не целостные
- Логи недостаточно подробные

A09: Как бороться

- Логировать все неудачные попытки аутентификации, отказы в доступе, ошибки валидации входных данных
- Обеспечить целостность логов, чтобы предотвратить возможность их подделки
- Мониторить не только приложения и вызовы API, но и инфраструктуру, сетевую активность, загрузку вычислительных ресурсов
- Обеспечить оперативное оповещение о нарушениях штатной работы системы

A10 Server-Side Request Forgery (SSRF)

- Не валидируются параметры запроса клиента, которые формируют URL внутренних вызовов
- Нет whitelist’а на используемые домены
- Клиенту возвращаются грязные данные



SSRF

GET /?url=http://images.com/image.jpg

- GET /?url=http://localhost/server-status - Apache HTTP
- GET /?url=http://169.254.169.254/latest/meta-data/ - AWS
- GET /?url=file:///etc/passwd