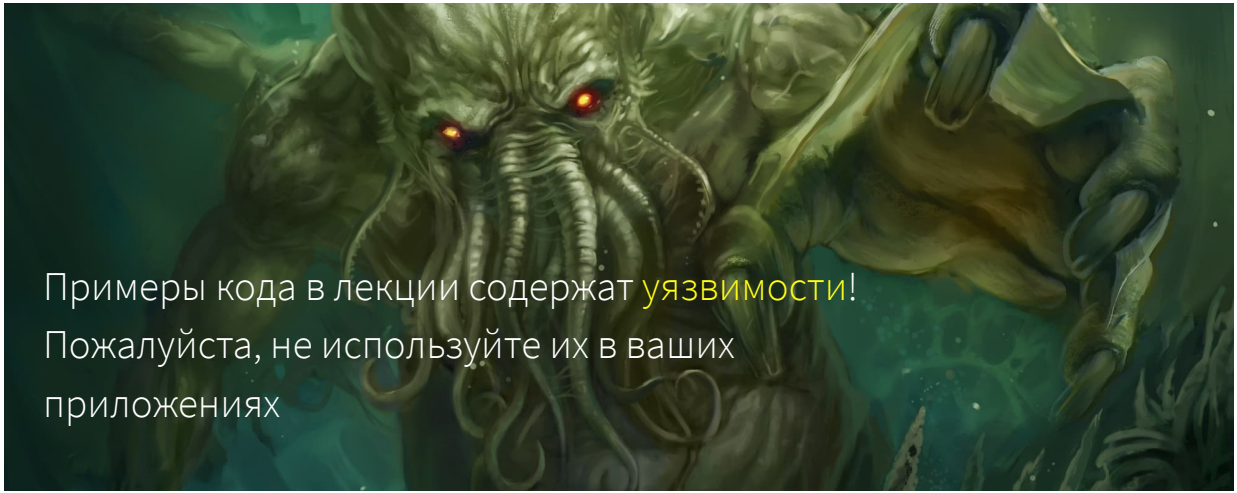
Безопасность веб-приложений

Семён Махаев

Дисклеймер



Убежать

Вперёд!

Почему безопасность — это важно?


Яндекс
Яндекс
Яндекс

Угроза

Потенциально возможное событие,
которое посредством работы
с компонентами сервиса, может нанести
ущерб

Уязвимость

Свойство сервиса, использование которого злоумышленником приводит к реализации угрозы

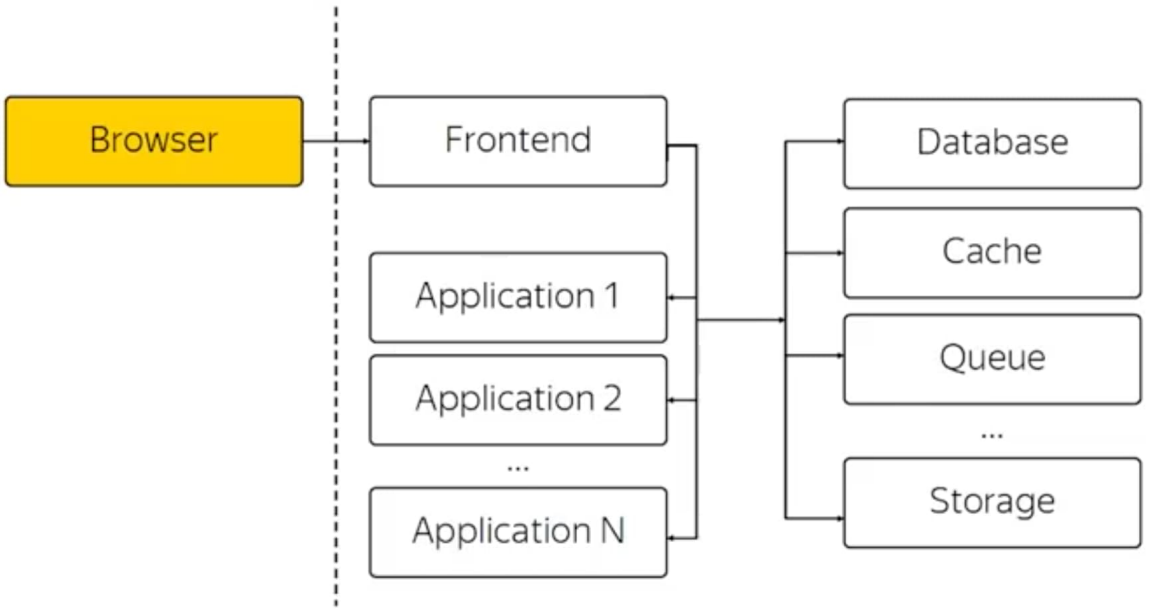
Атака

Реализация угрозы, путём использования уязвимостей

OWASP

The Open Web Application Security Project

Что мы защищаем?


Browser
Frontend
Application 1
Application 2
...
Application N
Database
Cache
Queue
...
Storage

Server-side

Client-side

Серверная часть

Логические уязвимости

Ошибки эксплуатации

Ошибки реализации

Логические уязвимости

Отсутствие разграничений прав доступа

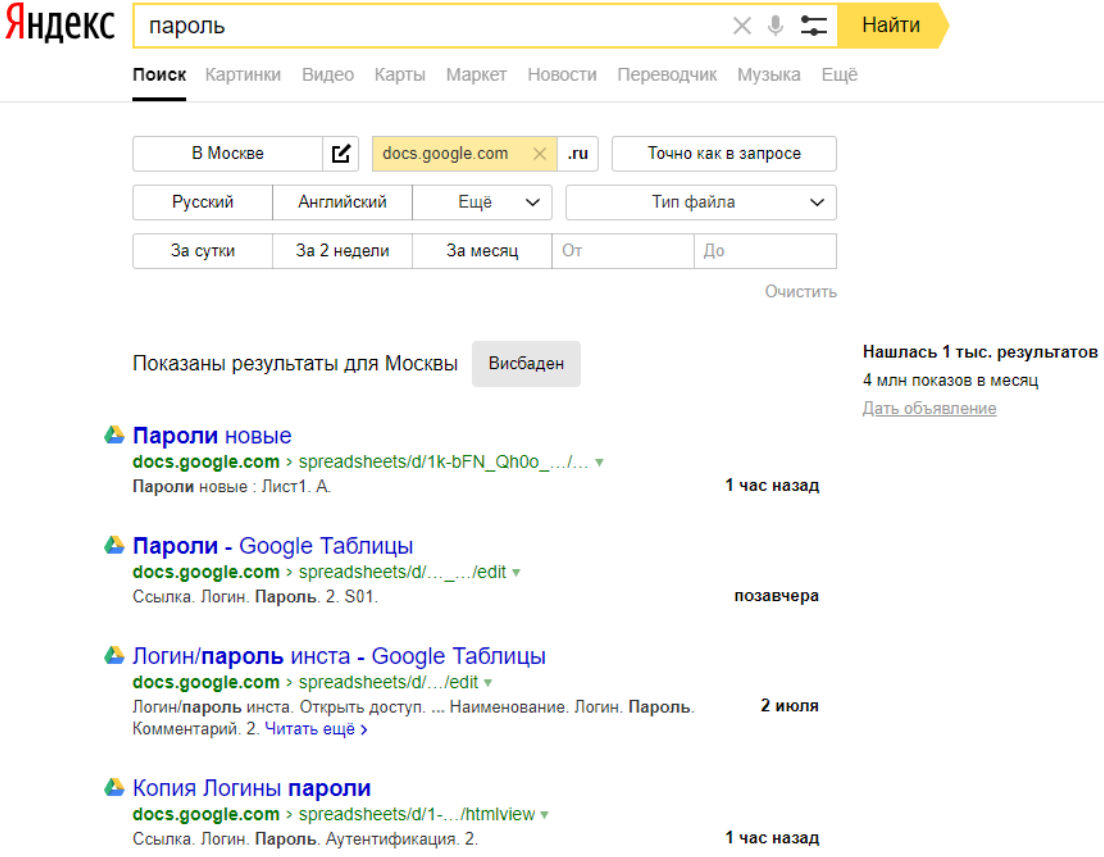
Избыточная небезопасная функциональность

Секретный урл

```
GET https://my-awesome-site.ru/admin
```

```
GET https://my-awesome-site.ru/aliSFla1434snf
```

В обоих случаях такая функциональность может быть скомпрометирована


Яндекс
пароль
Найти
Поиск
Картинки
Видео
Карты
Маркет
Новости
Переводчик
Музыка
Ещё
В Москве
docs.google.com
.ru
Точно как в запросе
Русский
Английский
Ещё
Тип файла
За сутки
За 2 недели
За месяц
От
До
Очистить
Показаны результаты для Москвы
Висбаден
Нашлась 1 тыс. результатов
4 млн показов в месяц
Дать объявление
Пароли новые
docs.google.com › spreadsheets/d/1k-bFN_Qh0o_…/…
Пароли новые : Лист1. A.
1 час назад
Пароли - Google Таблицы
docs.google.com › spreadsheets/d/…_…/edit
Ссылка. Логин. Пароль. 2. S01.
позавчера
Логин/пароль инста - Google Таблицы
docs.google.com › spreadsheets/d/…/edit
Логин/пароль инста. Открыть доступ. ... Наименование. Логин. Пароль. Комментарий. 2. Читать ещё ›
2 июля
Копия Логины пароли
docs.google.com › spreadsheets/d/1-…/htmlview
Ссылка. Логин. Пароль. Аутентификация. 2.
1 час назад

Логические уязвимости

Отсутствие разграничений прав доступа

Избыточная небезопасная функциональность

Нарушение логики работы приложения

Функциональность оплаты

Выбираем способ оплаты

```
GET https://online-shop.ru/payment/step/1
```

Производим оплату

```
GET https://online-shop.ru/payment/step/2
```

Подтверждаем по СМС

```
GET https://online-shop.ru/payment/step/3
```

Получаем товар

```
GET https://online-shop.ru/payment/step/4
```

Insecure direct object reference

```
GET https://disk.yandex.ru/client/1028379420

GET https://disk.yandex.ru/client/123
```

Insecure direct object reference

```
GET https://disk.yandex.ru/client/1028379420
```

```
GET https://disk.yandex.ru/client/123
```

Есть возможность получить данные другого пользователя

Предпосылки

Сложности реализации

Плохо продуманная архитектура

Человеческий фактор

Ошибки реализации

Инъекции и небезопасная сериализация

Открытые редиректы

Слабая криптография

Бинарные уязвимости

Инъекции

Уязвимости, направленные на возможность внедрения управляющего кода в язык запросов

SQL инъекции 👑

```
const { id } = req.query;

const sql = `SELECT * FROM adventures WHERE id='${id}';`;
```

```
GET /adventures?id=123
```

```
SELECT * FROM adventures WHERE id='123';
```

SQL инъекции 👑

```
GET /adventures?id=123
```

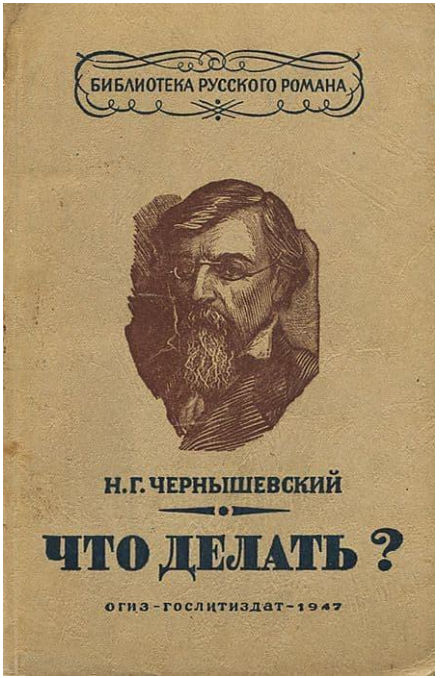
SQL инъекции 👑

```
GET /adventures?id=123' or '1'='1
```

```
SELECT * FROM adventures WHERE id='123' or '1'='1';
```

```
GET /adventures?id=123%27%20or%20%271%27=%271
```

SQL инъекции 👑



SQL инъекции 👑

Санитайзить / эскейпить данные

Использовать ORM

Использовать prepared statements

Эскейпинг

```
const { name } = req.body;
const escaped = name.replace('\'', '\'\'');


const sql = `
    SELECT * FROM adventures WHERE name='${escaped}';
`;
```

Работает только в частном случае

Prepared statements

```
PREPARE findAdventure (text) AS
    SELECT * FROM adventures WHERE name = $1;

EXECUTE findAdventure('magic');

const { name } = req.body;

sequelize
    .query('SELECT * FROM adventures WHERE name = :name', {
        replacements: { name },
        type: sequelize.QueryTypes.SELECT
    })
    .then(adventures => res.send(adventures));
```

Command injection 🐽

```
import { exec } from 'child_process';


const { repo, name } = req.body;

exec(`git clone ${repo}`);
exec(`cd ${name}`);
exec('npm install');
exec('npm test');

POST /hook

{
    repo: 'https://github.com/urfu-2018/telltail.git',
```

Command injection 🐽

```
import { exec } from 'child_process';


const { repo, name } = req.body;

exec(`git clone ${repo}`); // 🤔
exec(`cd ${name}`); // 🤔
exec('npm install');
exec('npm test');

POST /hook

{
    repo: 'https://github.com/urfu-2018/telltail.git',
```

Command injection

```
POST /hook



{
    repo: 'https://github.com/urfu-2018/telltail.git',
    name: 'telltail && sudo rm -rf /app/'
}

exec('cd telltail && sudo rm -rf /app/')
```

Command injection

```
POST /hook
```

```
{
    repo: 'https://github.com/urfu-2018/telltail.git',
    name: 'telltail && sudo rm -rf /app/'
}
```

```
exec('cd telltail && sudo rm -rf /app/')
```

Command injection

Разбивать сервис на микросервисы

Не передавать пользовательский ввод в параметры

Не запускать новые процессы из пользовательских запросов

Изолировать запускаемый код

Возможно, вам это не нужно

Command injection 🐽

```
import { exec } from 'child_process';


const { repo, name } = req.body;

const repoRegex = /^https?:\/\/[\w\d.\/\-]+\.git$/;
const nameRegex = /^[^{}()<>&*|=?;[\]$\-~!.%\/:+'`#]+$/;

if (!repo.match(repoRegex) || !name.match(nameRegex)) {
    return;
}

exec(`git clone ${repo}`);
exec(`cd ${name}`); // ...
```

CRLF Injection

Проксируем запрос в бэкенд

```
const { pathname } = req;

const data = await got(`https://${apiHostname}/${pathname}`);
```

Путь на фронте преобразуется в путь на бэке

На форнте: `GET /handle`

На бэке: `GET /backend/handle`

CRLF Injection

```
GET /handle%20HTTP/1.0%0D%0AHost%3A%20test%0D%0A%0D%0A HTTP/1.0\r\n
Host: frontend\r\n
\r\n
```

CRLF Injection

```
GET /handle%20HTTP/1.0%0D%0AHost%3A%20test%0D%0A%0D%0A HTTP/1.0\r\n
Host: frontend\r\n
\r\n
```

Запрос к бэкенду:

```
GET /backend/handle HTTP/1.0\r\n
Host: test\r\n
\r\n
HTTP/1.0\r\n
Host: backend\r\n
```

CRLF Injection

```
GET /handle%20HTTP/1.0%0D%0AHost%3A%20test%0D%0A%0D%0A HTTP/1.0\r\n
Host: frontend\r\n
\r\n
```

Запрос к бэкенду:

```
GET /backend/handle HTTP/1.0\r\n
Host: test\r\n
\r\n
HTTP/1.0\r\n
Host: backend\r\n
```

Пользовательский ввод попадает в ответ сервера

HTTP Header Splitting

```
GET /api/v1/vulnerable HTTP/1.0\r\n
Host: frontend\r\n

Referer: test%0D%0ASet-Cookie:%20name=Alice%0D%0A\r\n
\r\n

GET /application/vulnerable HTTP/1.0\r\n
Host: backend\r\n
Referer: test\r\n
Set-Cookie: name=Alice\r\n
\r\n
Set-Cookie: name=Bob\r\n
\r\n
```

HTTP Header Splitting

```
GET /api/v1/vulnerable HTTP/1.0\r\n
Host: frontend\r\n

Referer: test%0D%0ASet-Cookie:%20name=Alice%0D%0A\r\n
\r\n

GET /application/vulnerable HTTP/1.0\r\n
Host: backend\r\n
Referer: test\r\n
Set-Cookie: name=Alice\r\n
\r\n
Set-Cookie: name=Bob\r\n
\r\n
```

HTTP Header Splitting

```
GET /api/v1/vulnerable HTTP/1.0\r\n
Host: frontend\r\n

Referer: test%0D%0ASet-Cookie:%20name=Alice%0D%0A\r\n
\r\n

GET /application/vulnerable HTTP/1.0\r\n
Host: backend\r\n
Referer: test\r\n
Set-Cookie: name=Alice\r\n
\r\n
Set-Cookie: name=Bob\r\n
\r\n
```

Инъекция в HTTP-заголовке Referer

HTTP parameter contamination

```
GET /handle?payload=1%26login=admin
```

```
GET /handle?payload=1&login=admin&login=user
```

HTTP parameter contamination

```
GET /handle?payload=1%26login=admin
```

```
GET /handle?payload=1&login=admin&login=user
```

Инъекция в GET-параметре payload

Path traversal

```
GET /handler?filename=image.jpg
```

```
GET /handler?filename=../../../../../etc/passwd
```

Нужно разграничивать права доступа к директориям в файловой системе

Race conditions (гонки)

```
const { code } = req.body;


const isValid = await PromoCode.validate(code);

if (!isValid) {
    return;
}

await PromoCode.activate(code);
await PromoCode.markAsUsed(code);
```

Race conditions (гонки)

```
const { code } = req.body;


const isValid = await PromoCode.validate(code);

if (!isValid) {
    return;
}

await PromoCode.activate(code);
await PromoCode.markAsUsed(code); // ⏰
```

Нужно использовать транзакции

Раскрытие информации

Заголовки запроса в ответе сервера

Ссылки на внутреннюю документацию / описание внутренних структур

Язык и фреймворк

Отладочные данные в production

Traceback в production

Небезопасное логирование

Криптография

Криптография !== Безопасность

«Последний рубеж»

Не хранить пароли в открытом виде

При хешировании использовать соль

Denial of service

```
const { text } = req.body;


const regex = /^https?:\/\/\w+@\w+\.\w+$/;

const isValidEmail = Boolean(text.match(regex));

const statusCode = isValidEmail ? 200 : 400;

res.sendStatus(statusCode);
```

Denial of service

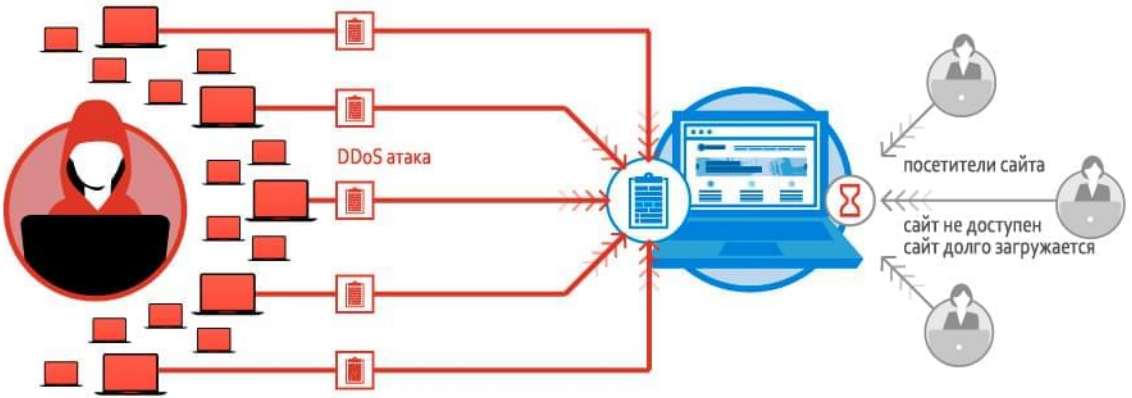
```
const { text } = req.body;


const regex = /^https?:\/\/\w+@\w+\.\w+$/;

const isValidEmail = Boolean(text.match(regex)); // ⚠️

const statusCode = isValidEmail ? 200 : 400;

res.sendStatus(statusCode);
```

Denial of service

```
const isValidEmail = Boolean(text.match(regex));
```

```
while (true); do
    curl $URL -d '{ text: $(cat "Война и мир.txt") }';
done
```

Distributed Denial of service

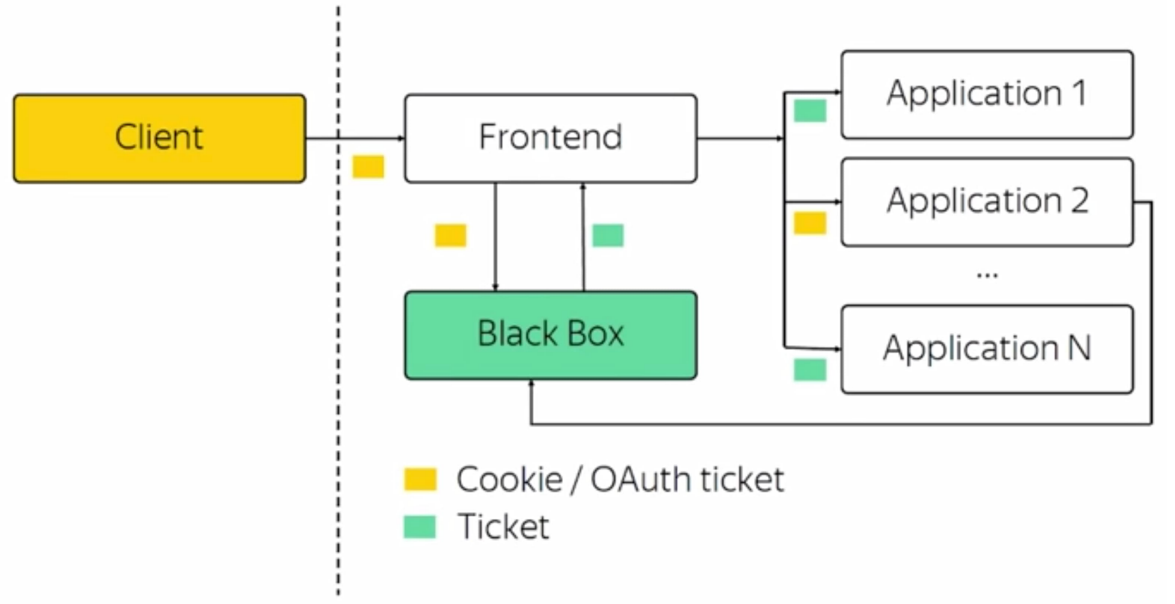


Ошибки архитектуры

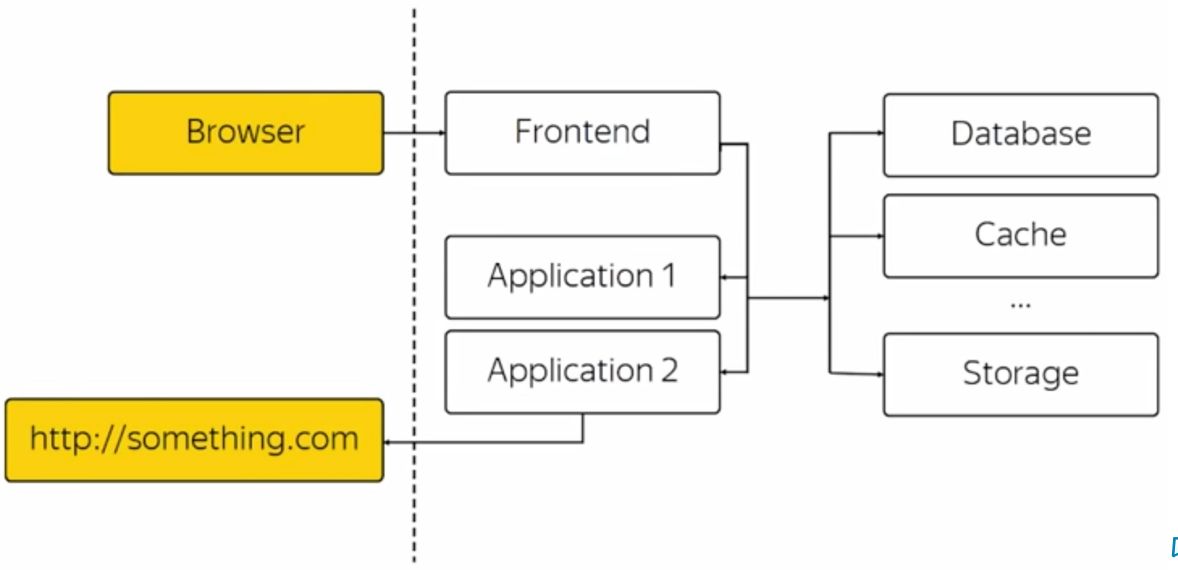
Некорректная авторизация и аутентификация

Атака SSRF

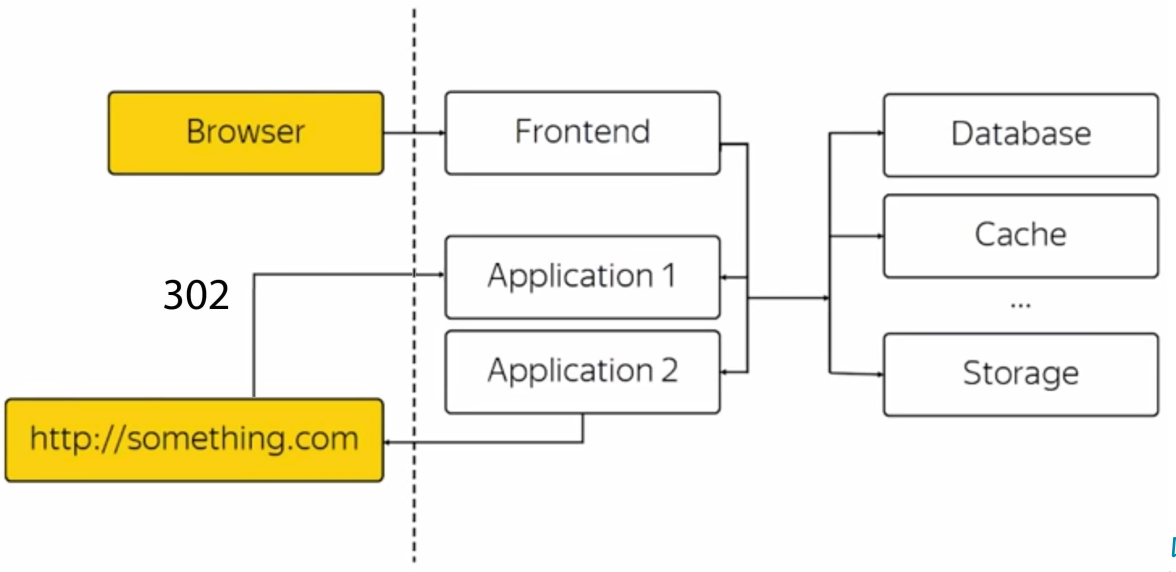
Аутентификация и авторизация



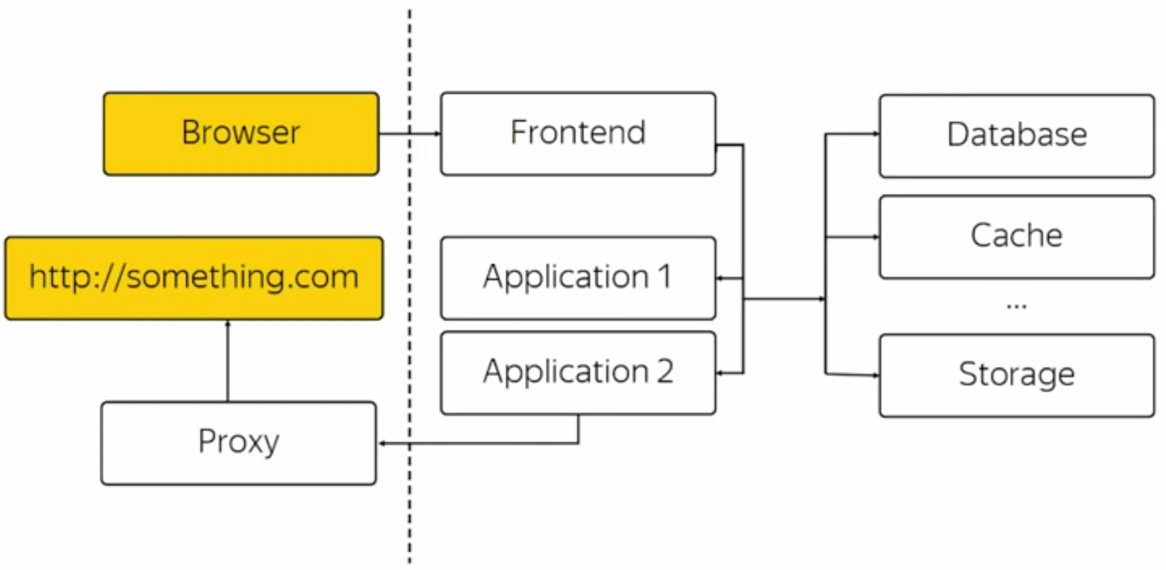
Атака Server Side Request Forgery



Атака Server Side Request Forgery



Атака Server Side Request Forgery



В Яндексе

Security Review

Автоматическое сканирование

Безопасные общие компоненты

Аутентификация и авторизация

Секреты и токены

Client side

Same Origin Policy

http://a.yandex.ru/dir1

http://a.yandex.ru/dir1/dir2

https://a.yandex.ru/dir1

http://a.yandex.ru:8080/dir1

http://b.yandex.ru/dir1

Cookie

```
Set-Cookie: id=1111; Path=/admin/
Set-Cookie: id=2222; Path=/; Domain=my.example.com
Set-Cookie: id=3333; Path=/; Http-Only; secure
```

Cookie

```
Set-Cookie: id=1111; Path=/admin/
Set-Cookie: id=2222; Path=/; Domain=my.example.com
Set-Cookie: id=3333; Path=/; Http-Only; secure
```

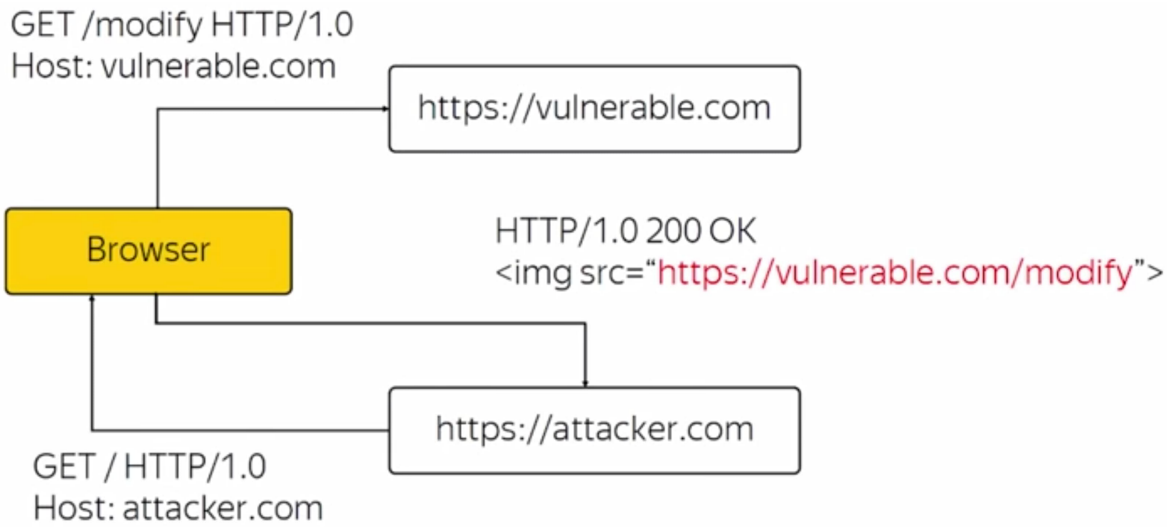
Http-Only - кука не доступна из JS

Secure - кука передаётся только по HTTPS

Cross-Origin Resource Sharing

```
GET /handler/ HTTP/1.1

Host: myawesomedomain.com
Origin: https://attacker.com
Cookie: session=123


HTTP/1.0 200 OK
Access-Control-Allow-Origin: https://attacker.com
Access-Control-Allow-Credentials: true
```

Cross site request forgery



Cross site request forgery

```
POST /modify HTTP/1.0
Host: vulnerable.com
X-CSRF-Token: keyboard-cat
```

Cross site request forgery

```
POST /modify HTTP/1.0
Host: vulnerable.com
X-CSRF-Token: keyboard-cat // 🦄 🌈
```

Cross-site request scripting (XSS)

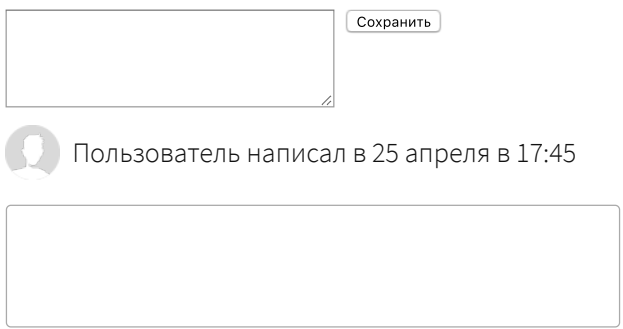
Атака, при которой вредоносный код внедряется в код вашего сайта и может быть запущен

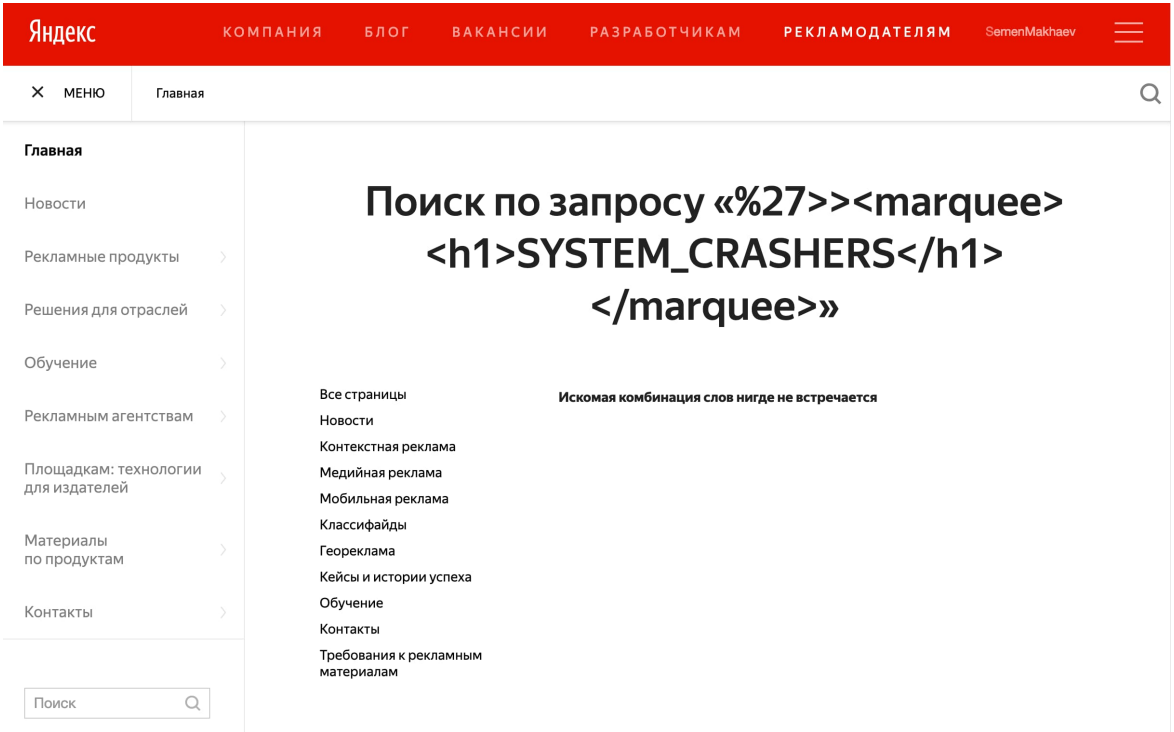
Cross-site request scripting (XSS)

Reflected

Stored

```
<textarea id="textarea" type="text"></textarea>
<button id="button">Сохранить</button>
<div id="result"></div>
<script>
    button.addEventListener('click', function () {
        result.innerHTML = textarea.value;
    });
<script>
```




Яндекс
КОМПАНИЯ
БЛОГ
ВАКАНСИИ
РАЗРАБОТЧИКАМ
РЕКЛАМОДАТЕЛЯМ
SemenMakhaev
МЕНЮ
Главная
Главная
Новости
Рекламные продукты
Решения для отраслей
Обучение
Рекламным агентствам
Площадкам: технологии для издателей
Материалы по продуктам
Контакты
Поиск
Поиск по запросу «%27>><marquee><h1>SYSTEM_CRASHERS</h1></marquee>»
Все страницы
Новости
Контекстная реклама
Медийная реклама
Мобильная реклама
Классифайды
Геореклама
Кейсы и истории успеха
Обучение
Контакты
Требования к рекламным материалам
Искомая комбинация слов нигде не встречается

Cross-Site Scripting

Санитайзинг / эскейпинг данных на выходе

Бескуковый домен для пользовательских скриптов

Заголовок X-XSS-Protection

Content Security Policy

Санитайзинг

```
import sanitizeHtml from 'sanitize-html';


const evilHtml = `
    <p style="background-image: url('attacker.com')">
        Hello, world!
    </p>
    <script>alert(document.cookie);</script>
`;

const kindHtml = sanitizeHtml(evilHtml, {
    allowedTags: ['a', 'p', 'div'],
    allowedAttributes: { a: ['href'] }
});
```

<XSS>’’;!—«<hr/>&{()}

Content Security Policy

```
Content-Security-Policy:
    default-src 'self';
    img-src *;
    script-src: trusted.com;
```

Content Security Policy

```
Content-Security-Policy:
    media-src: https://yandex.ru
```

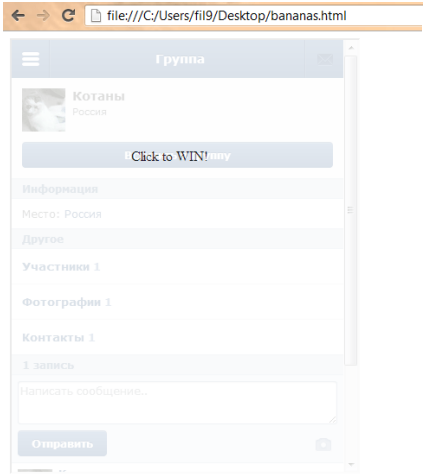
Content Security Policy

```
Content-Security-Policy:
    media-src: https://yandex.ru
```

Список директив CSP

* обращаем внимание на поддержку браузерами

ClickJacking




```
<iframe src="https://vk.com">
</iframe>


<a href="http://attacker.com">
    Click to WIN!
</a>
```

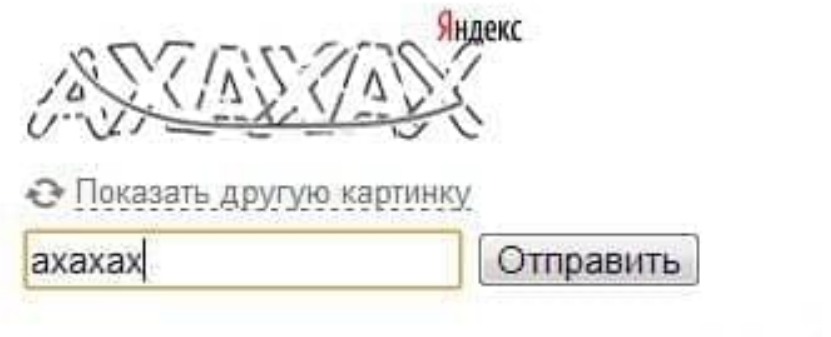
ClickJacking

```
X-Frame-Options: ALLOW-FROM http://trusted.com
X-Frame-Options: SAMEORIGIN
X-Frame-Options: DENY
```

* X-Frame-Options: ALLOW-FROM не работает в Google Chrome

** Но есть директива CSP frame-ancestors

Многократная отправка формы



Что делать, чтобы было безопаснее

Унификация сервисов

Не доверяйте пользовательским данным

Автоматизация (анализ кода, сканирования)

Общие безопасные компоненты

```
npm audit
```

Ссылки

Гайд OWASP v4

Лекции ШИБ

HTML5 Security Cheatsheet

Helmet.js

Acunetix 💰

Burb Suite

??