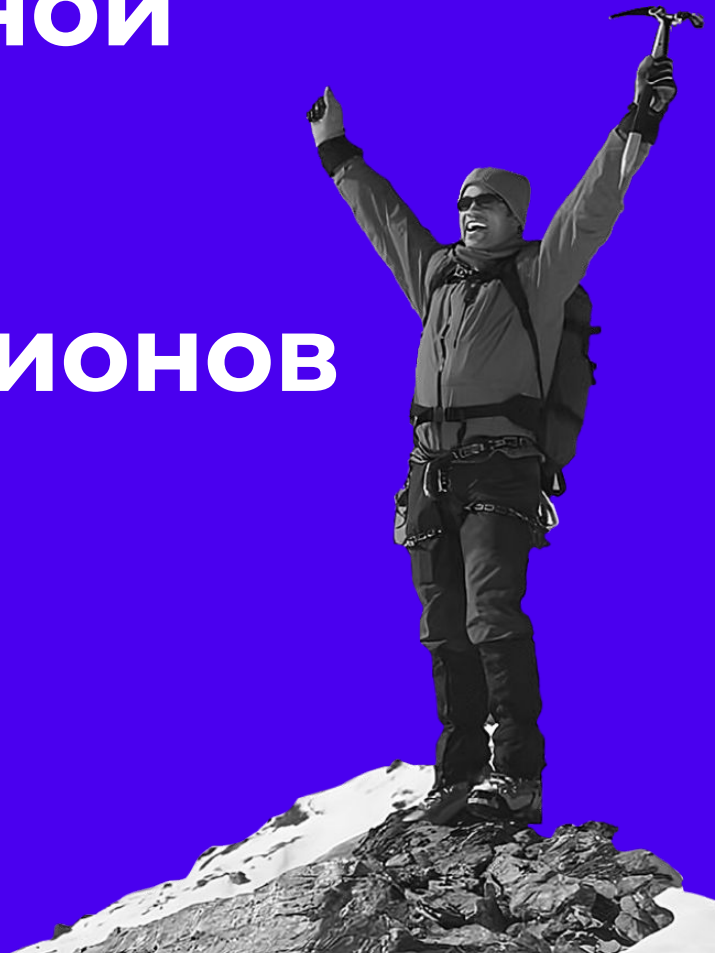
Квиз по безопасной разработке для настоящих секьюрити-чемпионов

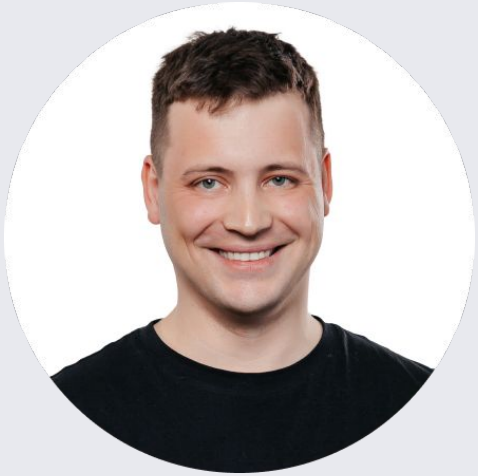


2024

startx.team

Start X

Артемий Богданов, Chief Hacking Officer Start X



Более 10 лет опыта в практической безопасности.

Выполнял проекты по аудиту защищенности веб-приложений, тестированию на проникновение различных информационных систем.

Разрабатывал задания для профессионального обучения тестированию на проникновение. Призер конкурсов PHDays, NeoQUEST и других CTF.

Имеет награды и подтверждения о нахождении уязвимостей в публичных приложениях и системах Uber, Yahoo, Vkontakte, Odnoklassniki, Mail.RU, Starbucks и других.

Кто тут настоящий секьюрити-чемпион?

На квизе вы разберете вопросы «со звездочкой» из теста Security Champion от Start X.

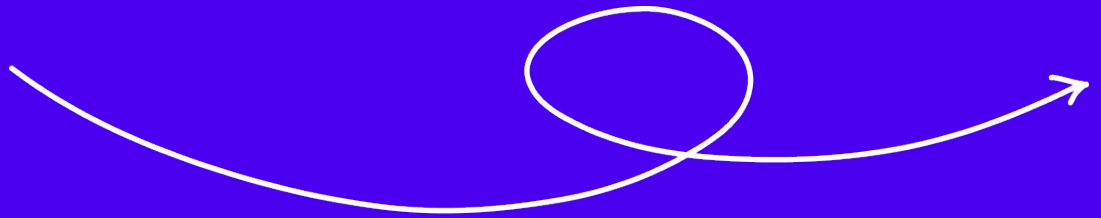
Вы проверите пять важных навыков безопасной разработки:

- Предотвращение RCE
- Вывод информации в браузер (XSS)
- Работа с SQL
- DevOps
- Предотвращение CRLF

Перейти на MyQuiz.ru

Start X

Блок 1:
Предотвращение RCE



Start X

Предотвращение RCE

Менеджер проекта недавно вернулся с конференции.

На совещании с разработчиками он рассказал о лучших практиках, которые узнал на мероприятии. К рассказу приложил флешку с кодом и попросил вас изучить опыт экспертов и передать его коллегам.

Фрагмент кода с флешки был помечен комментарием «проверить безопасность».

Как считаете, все ли в порядке с этим кодом? Укажите возможные проблемы, если они имеются.

Предотвращение RCE

БЛОК 1

ВОПРОС 1/3

```
var express = require('express');
var cookieParser = require('cookie-parser');
var escape = require('escape-html');
var serialize = require('node-serialize');
var app = express();
app.use(cookieParser())

app.get('/', function(req, res) {
 if (req.cookies.profile) {
   var str = new Buffer(req.cookies.profile, 'base64').toString();
   var obj = serialize.unserialize(str);
   if (obj.username) {
     res.send("Hello " + escape(obj.username));
   }
 } else {
     res.cookie('profile', "eyJ1c2VybmFtZSI6Ikl2YW4iLCJjb3VudHJ5IjoiUnVzc2lhIiwiY2l0e
       maxAge: 900000,
       httpOnly: true
     });
 }
 res.send("Hello World");
});
app.listen(3000);
```

Предотвращение RCE

БЛОК 1

ВОПРОС 1/3

- Код написан грамотно, проблем безопасности нет.
- Библиотека **cookie-parser** имеет известные уязвимости. Пользователь может составить специальный запрос и выполнить произвольный код на сервере. Нужно предварительно проверить значение **cookie**, перед его передачей в **cookie-parser**.
- При декодировании данных из **base64** можно выйти за рамки контекста данных и выполнить произвольный код на сервере.
- При первом входе приложение проставляет **cookie**, который можно декодировать, увидеть данные пользователя и подменить их. Можно подменить текущего пользователя или провести атаку **типа XSS**. Выполнить произвольный код на сервере не получится — мы просто парсим данные, а не передаем их на исполнение.
- При десериализации данных можно внедрить произвольный код, который будет выполнен на сервере.

Предотвращение RCE

Небезопасная десериализация (insecure deserialization)

node-serialize

Serialize a object including it's function into a JSON.

build passing

SECURITY WARNING

This module provides a way to unserialize strings into executable JavaScript code, so that it may lead security vulnerabilities if the original strings can be modified by untrusted third-parties (aka hackers). For instance, the following attack example provided by ajinabraham shows how to achieve arbitrary code injection with an IIFE:

```
var serialize = require('node-serialize');
var x = '{"rce":"_$$ND_FUNC$$_function (){console.log(\'exploited\')}()"}'
serialize.unserialize(x);
```

To avoid the security issues, at least one of the following methods should be taken:

1. Make sure to send serialized strings internally, isolating them from potential hackers. For example, only sending the strings from backend to fronend and always using HTTPS instead of HTTP.
2. Introduce public-key cryptosystems (e.g. RSA) to ensure the strings not being tampered with.

Предотвращение RCE

БЛОК 1

ВОПРОС 2/3

В наследство от бывшего коллеги, переехавшего в Сколково, вам досталось Flask-приложение, которое нужно доработать.

Безопасно ли в модуле реализована обработка данных из формы? Посмотрите на фрагмент модуля и выберите правильные утверждения.

Предотвращение RCE


```python
from flask import Flask, abort, request, render_template_string
import jinja2, re, hashlib

app = Flask(__name__)
app.secret_key = b'SECRET_KEY'

@app.route('/message_check', methods=['POST'])
def no_filter():

        message = request.form.get('message')

        template = '''
        <!DOCTYPE html>
        <html>
            <head>
                <title>Checking...</title>
            </head>
            <body>
                    <p>Check your message here:</p>
                <p>''' + message + '''</p>
            </body>
        </html>'''

        return render_template_string(template)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80, debug=False)
```

Предотвращение RCE

БЛОК 1

ВОПРОС 2/3

- Используется уязвимая библиотека **Flask**. Нужно переписать приложение на использование другого фреймворка.
- Приложение уязвимо к **XSS**. Уязвимый параметр — **message**, других уязвимостей нет.
- Пользовательский ввод не передается на исполнение — здесь нет **RCE**.
- Нельзя использовать пользовательский ввод в формировании шаблона, это может привести к проблемам.
- Код написан корректно, проблем нет.

Предотвращение RCE

БЛОК 1

Инъекция шаблонов на стороне сервера (SSTI, Server Side Template Injection)

Request

Pretty Raw Hex \n

```
POST /message_check HTTP/1.1
Host: 127.0.0.1
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.5481.78 Safari/537.36
Connection: close
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 110

message={{ config.__class__.from_envvar.__globals__.import_string("os").popen("cat /etc/passwd").read() }}
```

Response

Pretty Raw Hex Render \n

```
HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.10.8
Date: Sun, 19 Mar 2023 21:21:57 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 8409
Connection: close


<!DOCTYPE html>
<html>
  <head>
    <title>
      Checking...
    </title>
  </head>
  <body>
    <p>
      Check your message here:
    </p>
    <p>
      ##
      # User Database
      #
      # Note that this file is consulted directly only when the system is running
      # in single-user mode.  At other times this information is provided by
      # Open Directory.
      #
      # See the opendirectoryd(8) man page for additional information about
      # Open Directory.
      ##
      nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
      root:*:0:0:System Administrator:/var/root:/bin/sh
      daemon:*:1:1:System Services:/var/root:/usr/bin/false
      _uucp:*:4:4:Unix to Unix Copy Protocol:/var/spool/uucp:/usr/sbin/uucico
      _taskgated:*:13:13:Task Gate Daemon:/var/empty:/usr/bin/false
      _networkd:*:24:24:Network Services:/var/networkd:/usr/bin/false
      _installassistant:*:25:25:Install Assistant:/var/empty:/usr/bin/false
```

Предотвращение RCE

БЛОК 1

ВОПРОС 3/3

Вам на ревью попался вот такой фрагмент кода.

Посмотрите и ответьте — безопасен ли код?

```php
<?php

if(in_array($_GET['action'],['start_process','stop_process','clear_cache'])){
        passthru($_REQUEST['action']);
}
```

Предотвращение RCE

БЛОК 1

ВОПРОС 3/3

- Нет, не безопасен. Пользователь может передать команду в **post** параметре.
- Да, безопасен. Пользовательский ввод проверяется по белому списку.
- Нет, не безопасен. Пользователь может передать массив в параметре **action**, это позволит обойти проверку, после чего в **passthru** попадут потенциально опасные данные.
- Да, безопасен. **Passthru** — встроенная функция языка **PHP**, поэтому не является опасной, в нее можно передавать все, что угодно, это не приведет к рискам безопасности.

Предотвращение RCE

БЛОК 1

Инъекция команд операционной системы (OS Command Injection)

```php
<?php

if(in_array($_GET['action'],['start_process','stop_process','clear_cache'])){
        passthru($_REQUEST['action']);
}
```

passthru — Execute an external program and display raw output

The passthru() function is similar to the exec() function in that it executes a command. This function should be used in place of exec() or system() when the output from the Unix command is binary data which needs to be passed directly back to the browser. A common use for this is to execute something like the pbmplus utilities that can output an image stream directly. By setting the Content-type to image/gif and then calling a pbmplus program to output a gif, you can create PHP scripts that output images directly.

Предотвращение RCE

БЛОК 1

Инъекция команд операционной системы (OS Command Injection)

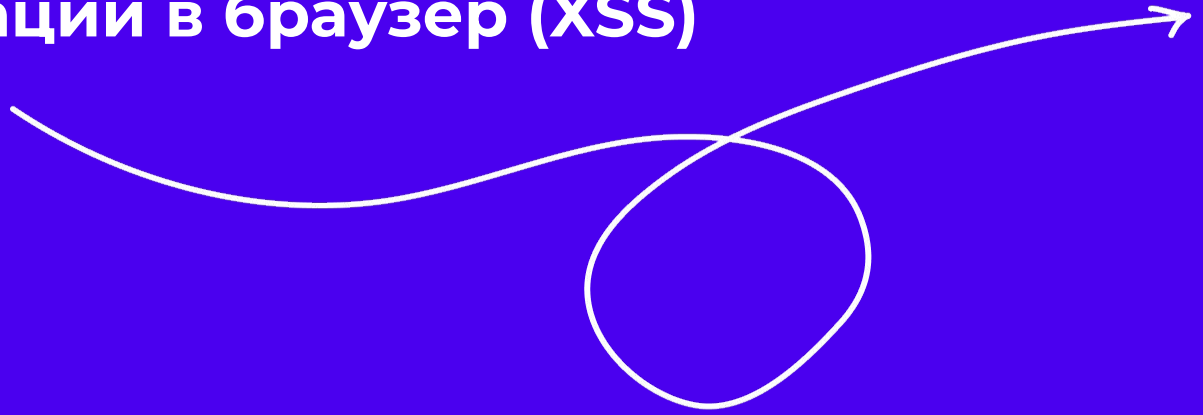
```php
<?php

if(in_array($_GET['action'],['start_process','stop_process','clear_cache'])){
        passthru $_REQUEST['action'];
}
```

POST request: action=cat /etc/passwd

Предотвращение RCE

1. Insecure Deserialization
2. Server Side Template Injection
3. OS Command Injection

Блок 2:
Вывод информации в браузер (XSS)



Start X

Вывод информации в браузер (XSS)

БЛОК 2

```
<a href="Данные"></a>
```

```
<script> Данные </script>
```

```
<!-- … ДАННЫЕ … -->
```

```
<body> … ДАННЫЕ … </body>
```

```
<div onmouseover="x='Данные'"></div>
```

```
<div name="Данные"> </div>
```

```
<div ДАННЫЕ="test"> </div>
```

```
<ДАННЫЕ href="/link/" />
```

```
<style>
selector { property : Данные; }
</style>
```

```
<script>x='Данные'</script>
```

Вывод информации в браузер (XSS)

БЛОК 2

ВОПРОС 1

Ваш новый разработчик решил обезопасить вывод на страницу значения из формы.

Как считаете, получилось у него решить задачу?

```
<body onload="document.body.innerHTML = '<?= htmlspecialchars($_GET['name'], ENT_QUOTES)?>';">
    <form>
        <input type=text name=name>
    </form>
</body>
```

Вывод информации в браузер (XSS)

БЛОК 2

ВОПРОС 1

- Используется преобразование спецсимволов в **html-сущности**, поэтому код не будет уязвимым.
- Нужно заменить **document.body.innerHTML** на **document.body.innerText**, пользовательский ввод предварительно обрабатывается **htmlspecialchars** — это решит проблему.
- Значение **innerHTML** нужно передавать в двойных кавычках, а вместо **htmlspecialchars** — использовать функцию **json_encode**.
- Ни один из ответов не подходит. Код небезопасен.

Вывод информации в браузер (XSS)

DEMO

Блок 3:
Работа с SQL

Start X

Работа с SQL

БЛОК 3

ВОПРОС 1/2

Вы пишете код для сайта с различными статьями. Пользователь ищет статьи по меткам. Данные, которые пользователь ввел в поле поиска, можно получить с помощью функции **getUserInput()**.

Является ли этот код безопасным?

```
1 String input = getUserInput();
2 String query = "SELECT * FROM articles WHERE  label= '"+ input + "'";
3 Statement statement = connection.createStatement();
4 ResultSet resultSet = statement.executeQuery(query);
```

Работа с SQL

- Нет, не является. В **SQL-запросе** присутствует оператор «*», вместо которого можно подставить что угодно.
- Да, является. Пользовательский ввод указан в кавычках.
- Нет, не является. Не производится экранирование или передача параметра через подготовленное выражение.
- Да, является. Мы используем подготовленные выражения, которые полностью исключают возможность **SQL-инъекции**.

Работа с SQL

DEMO

Работа с SQL

БЛОК 3

ВОПРОС 2/2

Один из ваших коллег написал веб-приложение, в котором данные в БД добавляются с использованием подготовленных выражений.

Безопасен ли код коллеги с точки зрения уязвимости для **SQL-инъекций**?

Работа с SQL

БЛОК 3

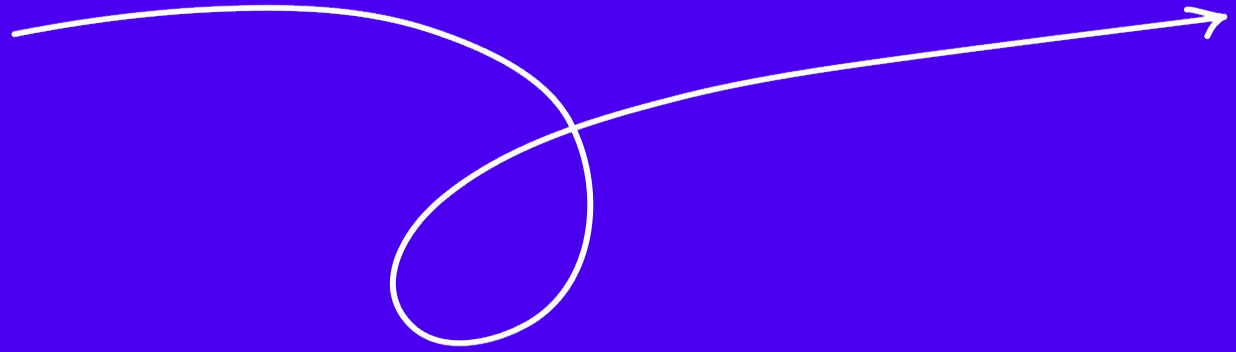
ВОПРОС 2/2

```
public static void main(String[] args) throws SQLException, ClassNotFoundException {
    Connection conn = null;
    String name = getName();
    int price = getPrice();
    String filter = getFilter();
    String filterStr = getFilterStr();

    PreparedStatement prepStmt = null;
    try {
        Class.forName(JDBC_DRIVER);
        conn = DriverManager.getConnection(DB_URL, USER, PASSWORD);
        String sql = "INSERT INTO Products (ProductName, Price) Values (?, ?)";
        if (filter != null) {
            sql = sql + " WHERE " + filter + "=?";
                prepStmt = conn.prepareStatement(sql);
                        prepStmt.setString(3, filterStr);
        } else {
                prepStmt = conn.prepareStatement(sql);
        }
        prepStmt.setString(1, name);
        prepStmt.setInt(2, price);
        prepStmt.executeUpdate();
    } finally {
        if (prepStmt != null) {
            prepStmt.close(); }
        if (conn != null) {
            conn.close(); }
    }
}
```

Работа с SQL

- Используются подготовленные выражения, а значит код безопасен.
- Значение параметров не заключены в одинарные кавычки, поэтому злоумышленник может проэксплуатировать **SQL injection**.
- Оба варианта не верны.

Блок 4:
DevOps



Start X

DevOps

БЛОК 4

ВОПРОС 1

Ваш коллега показал пример **location** из конфигурации веб-сервера **nginx**, который для облегчения нагрузки кэширует данные и отдает пользователю кэшированную страницу.

Какие нюансы вы видите в таком решении?

```
keys_zone=my_cache:100m;

location /api {
proxy_pass http://backend:3000
proxy_pass_header Set-Cookie;
proxy_pass_header Cookie;

proxy_ignore_headers Expires Cache-Control Set-Cookie;

proxy_cache_path /var/www/cache
proxy_cache my_cache;
proxy_cache_methods POST GET;
proxy_cache_key $scheme://$host$uri$is_args$query_string;
proxy_cache_valid 240m;
}
```

DevOps

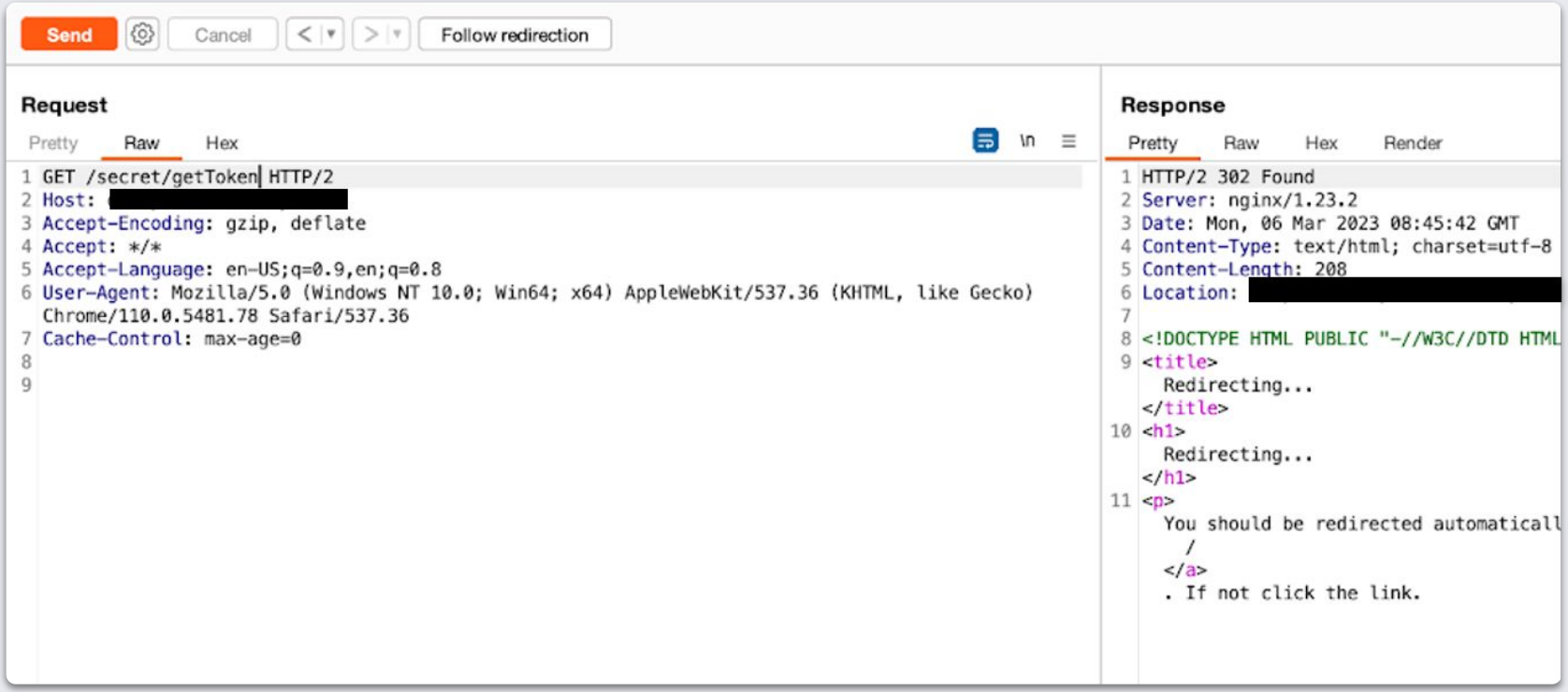
БЛОК 4

ВОПРОС 1

- Кэшируются ответы **backend** на запросы **GET** и **POST** — это позволит использовать **cache poisoning.**
- Кэшируются ответы **backend** по **host+url**, что может привести к раскрытию секретов.
- Дублируются **args** из-за директив **$is_args** и **$query_string**, что приводит к проблеме **parameter pollution.**
- Все вышеперечисленное верно.

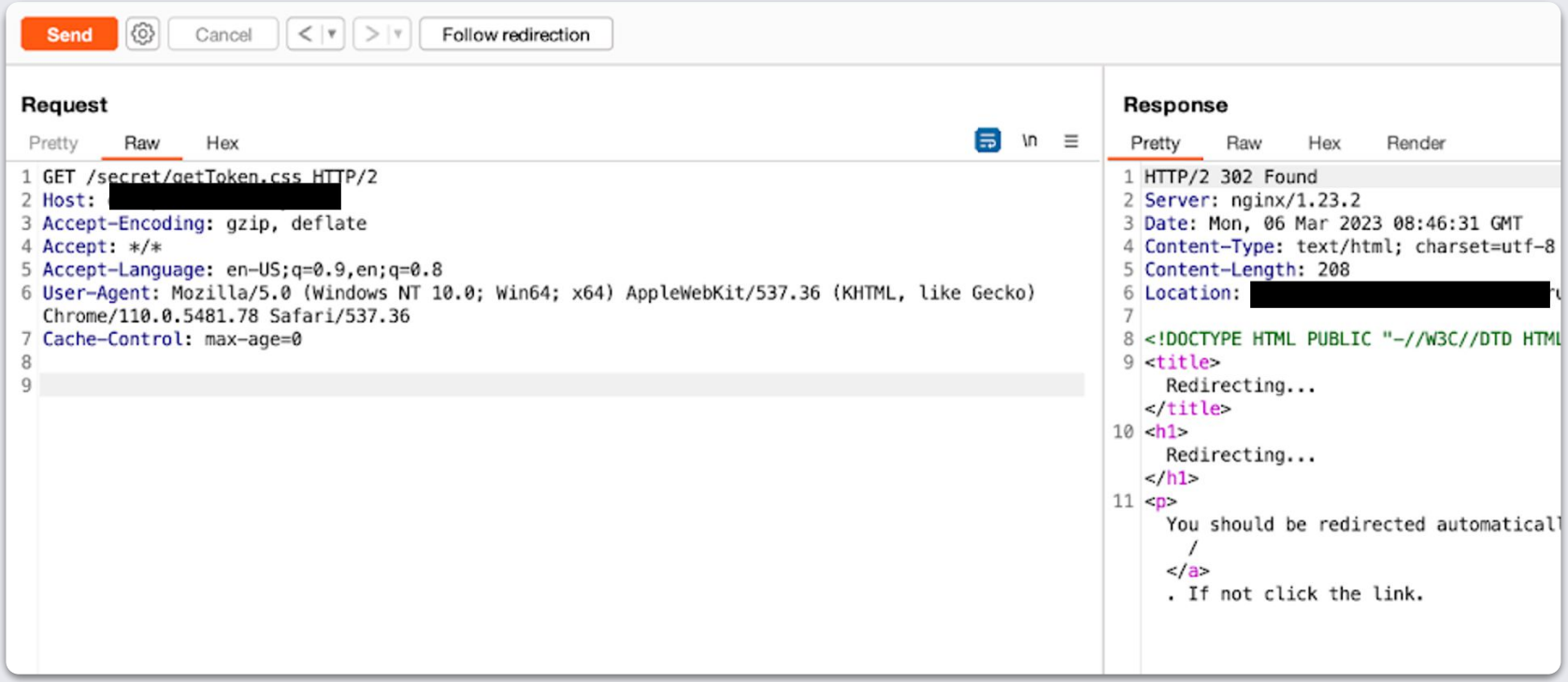
DevOps

БЛОК 4



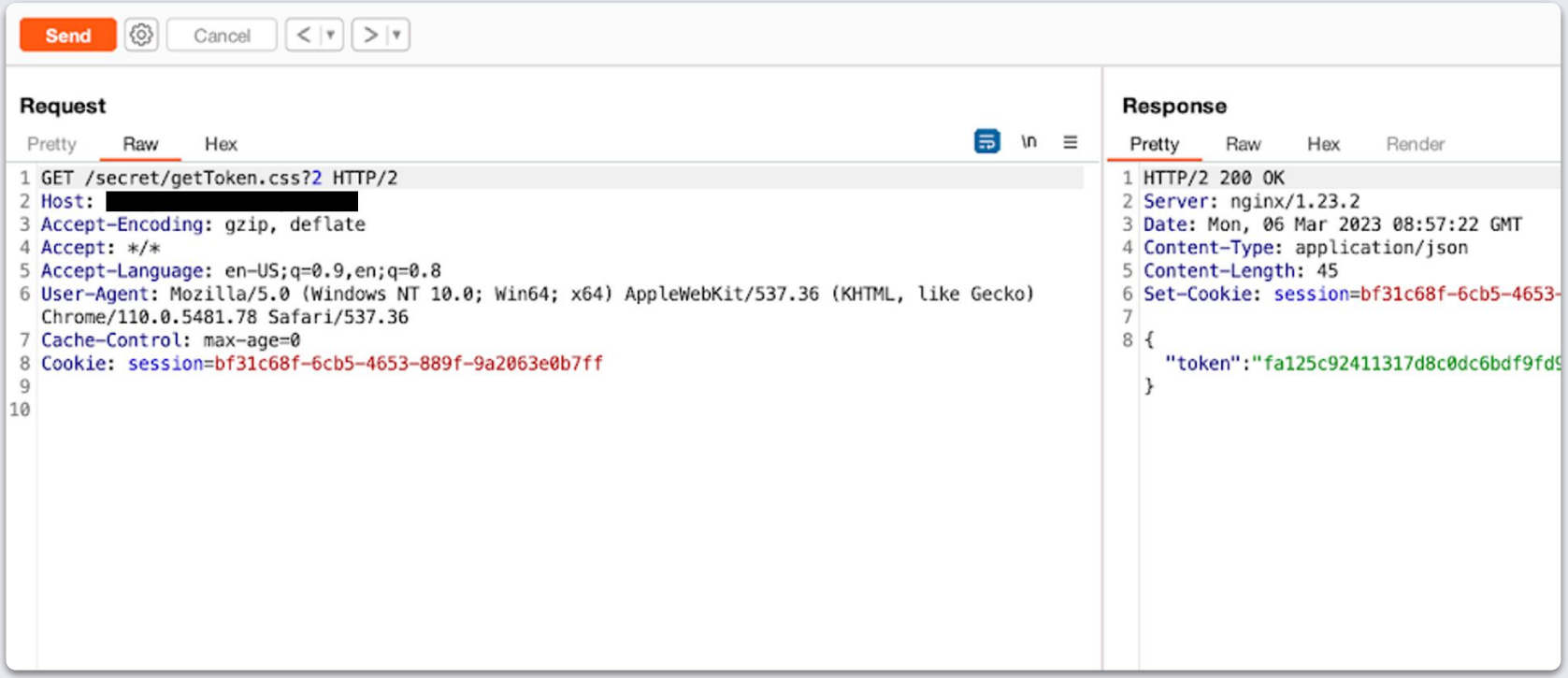
DevOps

БЛОК 4



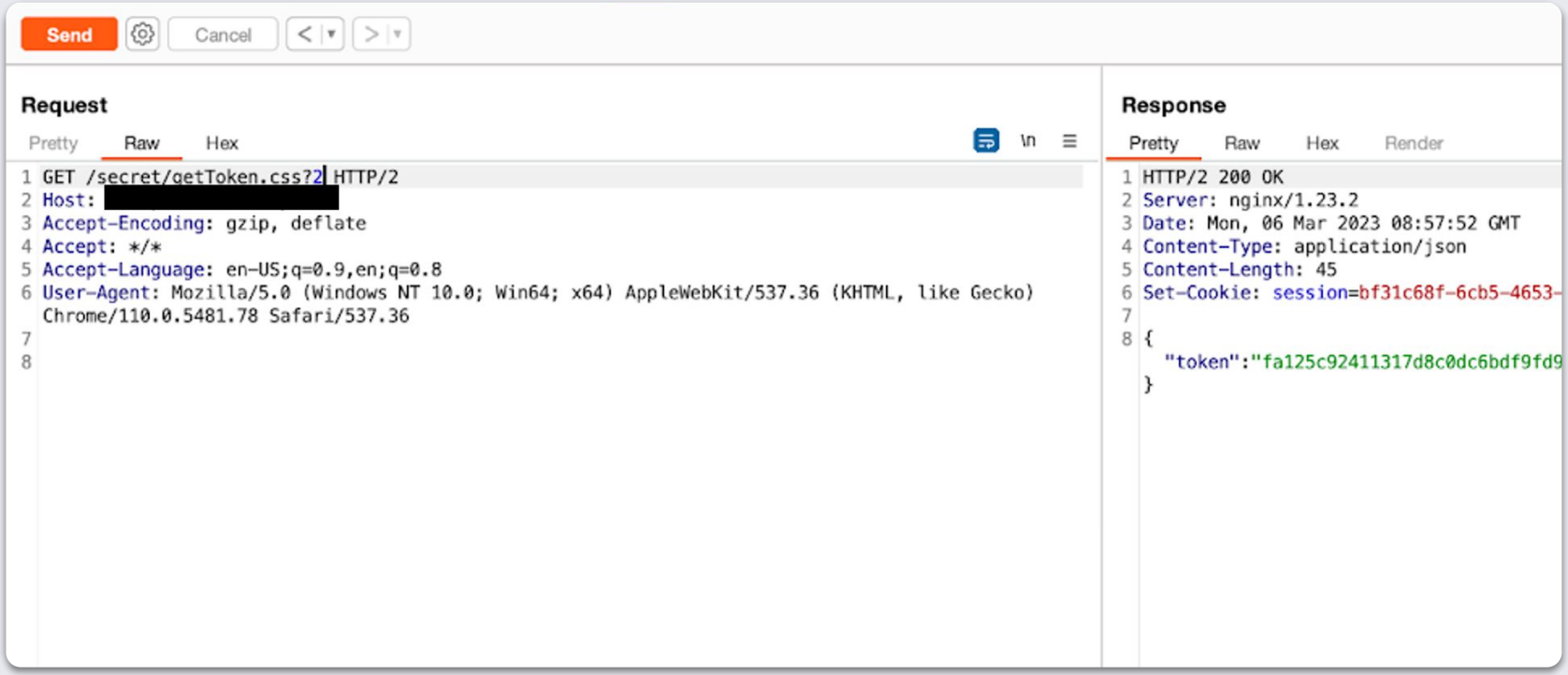
DevOps

БЛОК 4

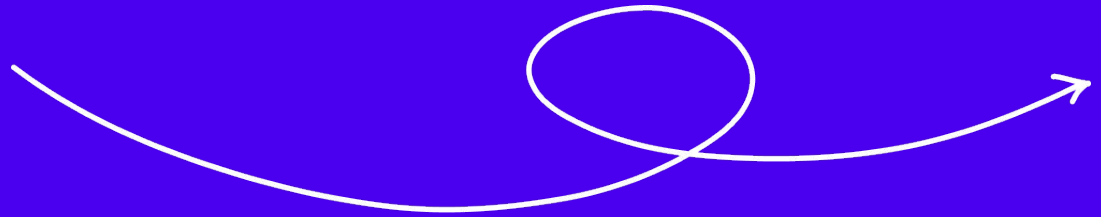


DevOps

БЛОК 4



Блок 5:
Предотвращение CRLF



Start X

Предотвращение CRLF

БЛОК 5

ВОПРОС 1

Вы устроились в банк разработчиком системы дистанционного обслуживания — ИТ-команда банка создавала его сама.

Добравшись до функции записи IP-адреса клиента в лог, вы обнаружили следующий код.

Как считаете, все ли в порядке с кодом или он потенциально уязвим?

Предотвращение CRLF

БЛОК 5

ВОПРОС 1

```php
<?php
$log  = "User: ".getClientIP().
            ' - '.date("F j, Y, g:i a").
            ' - '.$_SERVER[HTTP_HOST].$_SERVER[REQUEST_URI] .
            ' - '. file_get_contents('php://input') .PHP_EOL;
file_put_contents('./log_'.date("j.n.Y").'.log', $log, FILE_APPEND);

function getClientIP(){
    if (array_key_exists('HTTP_X_FORWARDED_FOR', $_SERVER)){
        return  $_SERVER["HTTP_X_FORWARDED_FOR"];
    } else if (array_key_exists('REMOTE_ADDR', $_SERVER)) {
        return $_SERVER["REMOTE_ADDR"];
    } else if (array_key_exists('HTTP_CLIENT_IP', $_SERVER)) {
        return $_SERVER["HTTP_CLIENT_IP"];
    }
    return '';
}
?>
```

Предотвращение CRLF

БЛОК 5

ВОПРОС 1

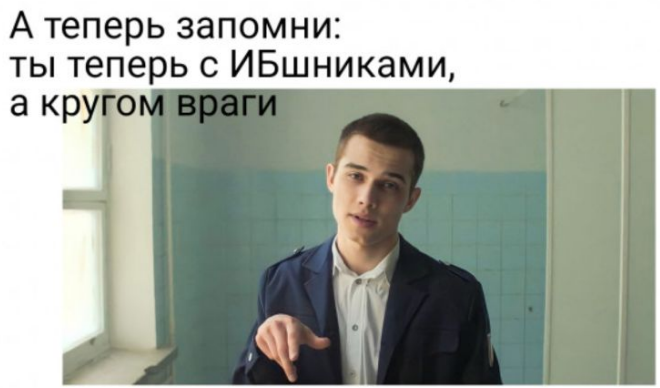
- Злоумышленник может внедрить **CRLF**, использовать знак переноса строки в заголовке **X_FORWARDED_FOR** и провести **spoofing** логов. Сервер воспримет значение как IP-адрес пользователя и запишет его в файл как есть.
- Пользователь может передать в теле запроса знак переноса строки и тем самым добавить поддельную запись в логи.
- Оба варианта верны.
- Все варианты не верны.

Читайте актуальные материалы по безопасной разработке в блоге на Хабре

Неочевидные угрозы: как защититься от атак на десериализацию, XSS и чтение произвольных файлов

10 мин 3K

Блог компании Start X (EX Антифишинг), Информационная безопасность*, Веб-разработка*, Управление разработкой*

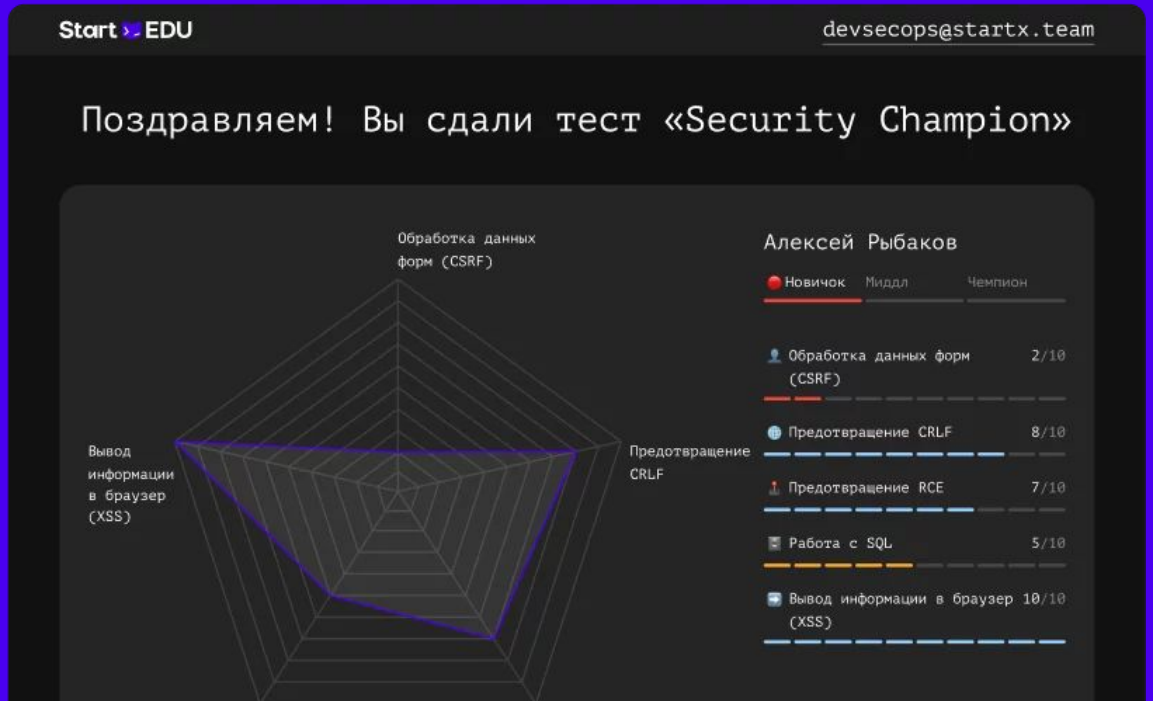


Злоумышленники могут успешно атаковать 98% веб-приложений. И это не просто громкие цифры, а данные из исследования Positive Technologies. Как такое возможно, если есть инструменты и практики типа SAST, DAST и WAF, а разработчики вроде бы нормально кодят?

Start X

Пройдите тест по безопасной разработке от Start X

В тесте Security Champion мы собрали еще больше реальных кейсов взломов известных приложений. После прохождения теста вы укрепите знания по безопасной разработке и получите личную карту компетенций.

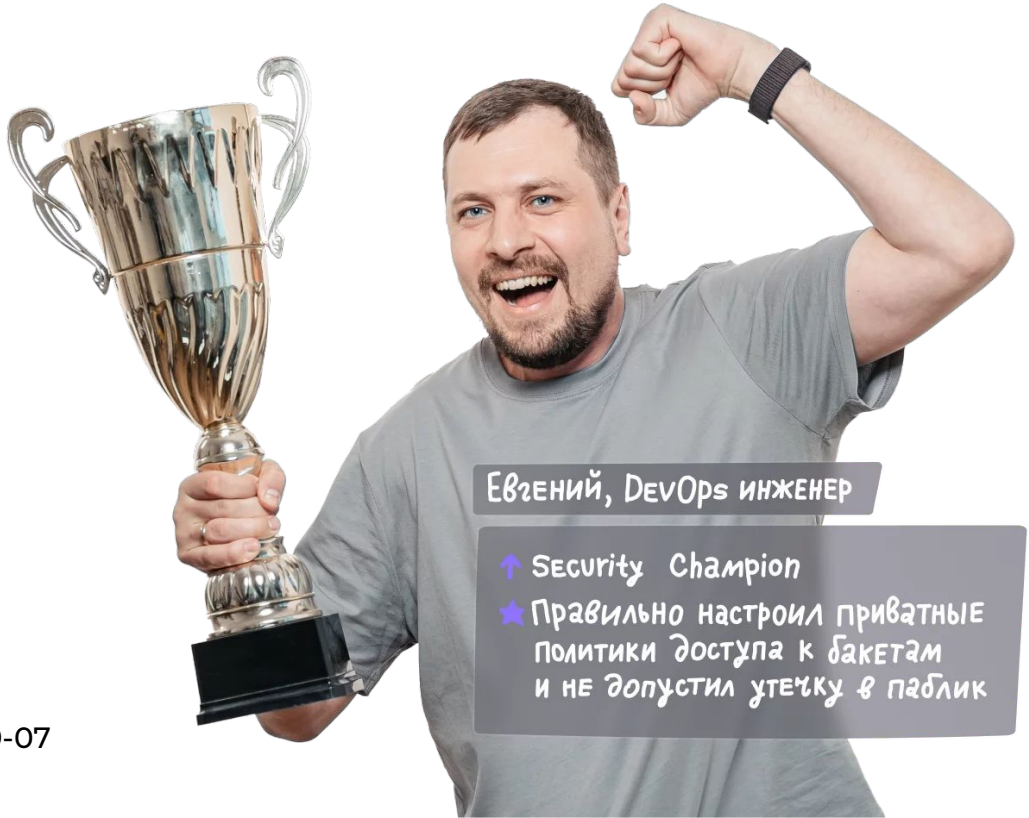


Тест Security Champion

Start X

Узнайте, как выпускать качественные продукты быстро и без уязвимостей

Start EDU — платформа по обучению продуктовых команд навыкам безопасной разработки



sales@startx.team

+7 (499) 677-19-07